THE DESIGN, DEVELOPMENT, IMPLEMENTATION, AND EVALUATION OF A FLIPPED INSTRUCTIONAL UNIT FOR AN ADVANCED PLACEMENT COMPUTER SCIENCE A CLASS

By

SHAUN T. HURLEY

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF EDUCATION

UNIVERSITY OF FLORIDA

2019
To my supportive and encouraging wife and parents
Thank you for always believing in me.
ACKNOWLEDGMENTS

I am grateful for many people that have helped me get to this point in my life.

First, I would like to thank my incredibly supportive and encouraging wife, Samantha, who always made me believe I was better than I thought I was. I cannot imagine what this journey would have been like without her. For the past three years, she picked up the slack for me, in every aspect of our lives, while I disappeared for hours on end to study. She took on virtually every role in and outside of the house to make sure I was able to spend the time I needed during this program. I will never forget the patience she had with me over these past three years. Words truly cannot begin to explain how lucky I am for having such an amazing and understanding partner.

I would also like to thank my parents, who taught me at a very young age that I could do anything I put my mind to. My mother, father, and step-father have always been there to help advise me in every aspect of my life. They have shaped me into the confident person that I am today. I could not have asked for better parenting, and I will be eternally grateful for that. I hope to be as good as them when I become a parent myself.

I am grateful for my advisor, Dr. Albert Ritzhaupt, who consistently gave encouraging and quality feedback, and was always able to do so in lightning speed. Even though he is one of the busiest people I know, I never felt like a burden, and he was always ready to pick up the phone to help me out. I would not have been able to get through this dissertation as quickly without him as my advisor. I appreciate all of his advice throughout this study.

I am also grateful for my committee members, Drs. Swapna Kumar, Danling Fu, and Cliff Haynes, as well as Dr. Kelli Peck Parrott, all of whom were supportive of my
work on this dissertation. They gave me valuable feedback and always brought with them unique perspectives that helped shape this study into what it became. I appreciate their suggestions and time spent with me on this dissertation.

I would like to thank the participating members of the focus group, who took the time out of their very busy schedules to review all of the materials in this unit. I certainly could not have completed this study without them. Working with them was such a valuable experience, and I have definitely grown as a computer science educator after working with this fantastic group.

Additionally, I am thankful for my students that agreed to participate in this study. I understand that it can be time consuming and awkward when it comes to interviews, so I sincerely appreciate these students for setting aside time in their busy schedules to meet with me and for potentially stepping outside of their comfort zones.

I would also like to thank the Educational Technology faculty members for taking a chance on me by admitting me into this program, and for all of my professors over these past three years. I was pushed further outside of my comfort zone than I thought was possible, which resulted in tremendous personal growth, and for that, I am incredibly grateful.

Lastly, I would like to thank the members of UF’s Ed Tech Cohort 5, for always being around to help with all aspects of this program. Specifically, I want to thank Tim Miller and Paul Baker for inviting me into their group. I couldn’t have asked for better classmates and friends. I look forward to an everlasting friendship.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>4</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>10</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>11</td>
</tr>
<tr>
<td>LIST OF ABBREVIATIONS</td>
<td>12</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>13</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>15</td>
</tr>
<tr>
<td>Background</td>
<td>15</td>
</tr>
<tr>
<td>Blended Learning</td>
<td>17</td>
</tr>
<tr>
<td>Flipped Learning</td>
<td>18</td>
</tr>
<tr>
<td>Computer Science</td>
<td>19</td>
</tr>
<tr>
<td>Subjectivity Statement</td>
<td>19</td>
</tr>
<tr>
<td>Research Questions</td>
<td>23</td>
</tr>
<tr>
<td>Overview of Research Design</td>
<td>23</td>
</tr>
<tr>
<td>Initial Steps</td>
<td>24</td>
</tr>
<tr>
<td>Phase I</td>
<td>24</td>
</tr>
<tr>
<td>Phase II</td>
<td>25</td>
</tr>
<tr>
<td>Definition of Terms</td>
<td>25</td>
</tr>
<tr>
<td>2 LITERATURE REVIEW</td>
<td>27</td>
</tr>
<tr>
<td>Methodology of Review</td>
<td>27</td>
</tr>
<tr>
<td>K-12 Computer Science Education</td>
<td>28</td>
</tr>
<tr>
<td>History</td>
<td>28</td>
</tr>
<tr>
<td>Research In K-12 Computer Science</td>
<td>31</td>
</tr>
<tr>
<td>AP Computer Science A</td>
<td>32</td>
</tr>
<tr>
<td>Blended Learning</td>
<td>33</td>
</tr>
<tr>
<td>General Blended Learning at the Secondary Education Level</td>
<td>34</td>
</tr>
<tr>
<td>Blended Learning in Computer Science at the Secondary Education Level</td>
<td>35</td>
</tr>
<tr>
<td>Blended Learning in Computer Science at the Higher Education Level</td>
<td>36</td>
</tr>
<tr>
<td>Attitudes and perceptions</td>
<td>37</td>
</tr>
<tr>
<td>Achievement</td>
<td>38</td>
</tr>
<tr>
<td>Recommendations</td>
<td>40</td>
</tr>
<tr>
<td>Flipped Learning in Computer Science</td>
<td>42</td>
</tr>
<tr>
<td>Flipped Learning in Computer Science at the Secondary Education Level</td>
<td>42</td>
</tr>
<tr>
<td>Flipped Learning in Computer Programming at the Higher Education Level</td>
<td>43</td>
</tr>
<tr>
<td>Engagement</td>
<td>43</td>
</tr>
</tbody>
</table>
Lessons and Sequencing

Overview

Summary

Limitations

Validity

Ethical Considerations

Data Analysis

Data Sources

Procedure

Phase I

Phase II

Research Design

Setting and Participants

Focus Group / Content Reviewers

Interviews

Field Notes and Daily Journal

Student-Generated Exercises

Socrative

CodingBat

EdPuzzle

Required Questions

Slack

Data Analysis

RQ1: After Initially Designing a Flipped Unit of Instruction, What Recommendations Were Made by the Focus Group During the Development Phase?

RQ2: To What Extent was the Unit Implemented as Designed?

RQ3: What Were Students' Perceptions of the Unit?

RQ4: Was There Any Preliminary Evidence to Support the Effectiveness of the Unit?

RQ5: What Were the Lessons Learned and Resulting Final Changes Made to the Unit?

Ethical Considerations

Validity

Limitations

Summary

4 INTERVENTION

Overview

Videos

In-Class Activities

Lessons and Sequencing
Basics of Arrays ................................................................. 92
Arrays of Objects .................................................................. 98
Copying Arrays .................................................................. 100
Objects with Arrays ............................................................ 101
Parallel Arrays .................................................................. 102
Searching .......................................................................... 103
Sorting ............................................................................. 104
2-Dimensional Arrays ........................................................... 106
Review and Test ................................................................ 107
Summary ........................................................................... 108

5 RESULTS ........................................................................... 109

Overview ........................................................................... 109
RQ1: After Initially Designing a Flipped Unit of Instruction, What
Recommendations Were Made by the Focus Group During the Development
Phase? ............................................................................. 109
Meeting #1: Initial Night – Day 5 (Basics of Arrays) ................. 110
Meeting #2: Night 5 – Day 7 (Arrays of Objects and Copying Arrays) 111
Meeting #3: Night 7 – Day 10 (Objects with Arrays and Parallel Arrays) 112
Meeting #4: Night 10 – Day 14 (Searching and Sorting) .............. 112
Meeting #5: Night 14 – Day 19 (2D Arrays) ............................. 113
Meeting #6: Student Interview Questions .................................. 114

RQ2: To What Extent was the Unit Implemented as Designed? .... 114
Time Spent ........................................................................ 115
EdPuzzle/Videos ................................................................ 116
Video Questions .................................................................. 117
Pair-Programming ................................................................ 119
Critiquing Students Solutions ............................................... 121
Socrative ......................................................................... 123
Algorithm Tracing ............................................................... 125
Slack – General ................................................................. 126
Role-Playing ..................................................................... 127
CodingBat ........................................................................ 128
Student-Generated Exercises ............................................... 130
Optional Activities ............................................................ 131
Summary ........................................................................... 132

RQ3: What Were Students’ Perceptions of the Unit? .................. 133
Effectiveness ..................................................................... 135
Engagement ...................................................................... 136
Satisfaction ...................................................................... 138
General Thoughts on the Flipped Classroom .......................... 141
Thought-Provoking Activities .............................................. 144
Peers ............................................................................. 145
Values ............................................................................. 148
Suggestions ...................................................................... 150
Summary ........................................................................... 151
RQ4: Was There Any Preliminary Evidence to Support the Effectiveness of the Unit? ............................................................................................................. 152
Socrative ........................................................................................................... 152
EdPuzzle .......................................................................................................... 153
CodingBat ........................................................................................................ 158
RQ5: What Were the Lessons Learned and Resulting Final Changes Made to the Unit? ............................................................................................................. 159
Lessons Learned .............................................................................................. 159
Final Changes Made to the Unit ....................................................................... 160

6 DISCUSSION AND IMPLICATIONS ....................................................................... 164

Summary of Study ............................................................................................. 164
Additional Limitations ...................................................................................... 165
Discussion of Results ......................................................................................... 166
   Design and Development ............................................................................. 166
   Implementation ............................................................................................ 168
   Evaluation .................................................................................................... 172
      Students’ perceptions ............................................................................... 172
      Effectiveness ........................................................................................... 176
Implications ........................................................................................................ 181
   Personal Implications .................................................................................. 181
   Implications for the Context of Flipped Computer Science Classes ............ 183
   Implications for the Field of Flipped Classrooms ........................................ 186
   Implications for the Field of Computer Science ......................................... 188
   Implications for Novice Computer Science Teachers ................................... 190
Recommendations for Future Research ............................................................. 191
Summary ............................................................................................................ 193

APPENDIX

A FIELD NOTES TEMPLATE .................................................................................. 195
B DAILY JOURNAL TEMPLATE ............................................................................ 196
C GUIDING QUESTIONS FOR THE FOCUS GROUP MEETINGS ..................... 197
D STUDENT INTERVIEW PROTOCOL .................................................................. 198
LIST OF REFERENCES .......................................................................................... 201
BIODGRAPHICAL SKETCH .................................................................................. 216
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Principles of multimedia learning.</td>
</tr>
<tr>
<td>4-1</td>
<td>Overview of pacing.</td>
</tr>
<tr>
<td>4-2</td>
<td>Basics of arrays.</td>
</tr>
<tr>
<td>4-3</td>
<td>Arrays of objects.</td>
</tr>
<tr>
<td>4-4</td>
<td>Copying arrays.</td>
</tr>
<tr>
<td>4-5</td>
<td>Objects with arrays.</td>
</tr>
<tr>
<td>4-6</td>
<td>Parallel arrays.</td>
</tr>
<tr>
<td>4-7</td>
<td>Searching.</td>
</tr>
<tr>
<td>4-8</td>
<td>Sorting.</td>
</tr>
<tr>
<td>4-9</td>
<td>Two-dimensional arrays.</td>
</tr>
<tr>
<td>4-10</td>
<td>Assessment.</td>
</tr>
<tr>
<td>5-1</td>
<td>Overview of original pacing.</td>
</tr>
<tr>
<td>5-2</td>
<td>Overview of updated pacing.</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Conceptual framework for this study.</td>
<td>61</td>
</tr>
<tr>
<td>3-1</td>
<td>Research design overview.</td>
<td>65</td>
</tr>
<tr>
<td>3-2</td>
<td>A rendering of the classroom used in this study.</td>
<td>67</td>
</tr>
<tr>
<td>3-3</td>
<td>A student’s view of one attempt on CodingBat.</td>
<td>74</td>
</tr>
<tr>
<td>3-4</td>
<td>A teacher’s view of one student’s attempt on three CodingBat questions.</td>
<td>75</td>
</tr>
<tr>
<td>3-5</td>
<td>A teacher’s view of one student’s viewing data for one video on EdPuzzle.</td>
<td>76</td>
</tr>
<tr>
<td>3-6</td>
<td>Slack, an online communication tool.</td>
<td>78</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>2D</td>
<td>Two-dimensional</td>
<td></td>
</tr>
<tr>
<td>ACM</td>
<td>Association for Computing Machinery</td>
<td></td>
</tr>
<tr>
<td>AP</td>
<td>Advanced Placement</td>
<td></td>
</tr>
<tr>
<td>CFC</td>
<td>Comment First Coding</td>
<td></td>
</tr>
<tr>
<td>CLT</td>
<td>Cognitive Load Theory</td>
<td></td>
</tr>
<tr>
<td>CS</td>
<td>Computer Science</td>
<td></td>
</tr>
<tr>
<td>CSA</td>
<td>Computer Science A</td>
<td></td>
</tr>
<tr>
<td>CSP</td>
<td>Computer Science Principles</td>
<td></td>
</tr>
<tr>
<td>CSTA</td>
<td>Computer Science Teachers Association</td>
<td></td>
</tr>
<tr>
<td>CTML</td>
<td>Cognitive Theory of Multimedia Learning</td>
<td></td>
</tr>
<tr>
<td>ESC</td>
<td>Exploring Computer Science</td>
<td></td>
</tr>
<tr>
<td>FRQ</td>
<td>Free-Response Question</td>
<td></td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
<td></td>
</tr>
<tr>
<td>ISTE</td>
<td>International Society for Technology Education</td>
<td></td>
</tr>
<tr>
<td>LMS</td>
<td>Learning management system</td>
<td></td>
</tr>
<tr>
<td>STEM</td>
<td>Science, Technology, Engineering, and Mathematics</td>
<td></td>
</tr>
</tbody>
</table>
The number of students taking computer science is on the rise, yet there are not enough qualified computer science teachers to fill the need, which may be attributing to the low retention rate in computer science programs. Since flipped learning, a form of blended learning affords the opportunity for active learning in the classroom, it was used as a foundation in an effort to improve students' levels of engagement and satisfaction in learning computer science, ideally improving their perceptions of computer science. In addition, this study was conducted to aid teachers that are unqualified or new to the field of computer science as well as to improve the quality of my own instruction. In turn, the hope was that the implications for other teachers would ultimately help towards increasing the retention rate in computer science programs in the future.

This design and development dissertation begins with an overview of the initial design of a flipped instructional unit on the topic of arrays for an Advanced Placement Computer Science A (AP CSA) class that spanned over 19 days. The unit was designed based on the cognitive theory of multimedia learning (CTML), active learning techniques, and recommendations made by previous researchers. It was then
The focus group made several suggestions that resulted in minor updates during the development phase. The main impediment to a smooth implementation was the underestimation of the anticipated time required for the activities; more time is needed for future implementations of this unit. Finally, although all activities appeared to have some value, solving programming exercises surfaced as the most favored activity.
CHAPTER 1
INTRODUCTION

The number of students taking computer science is rapidly growing (Cateté, 2018), yet there are not enough qualified computer science teachers (Qian, Hambrusch, Yadav, & Gretter, 2018), which may be responsible for the low retention rate in computer science programs (Giannakos, Pappas, Jaccheri, & Sampson, 2017). I have learned first-hand, as someone without a computer science background, that although it is possible to learn how to teach computer science, the process can be challenging; I am certainly not alone (e.g., Yadav, Gretter, Hambrusch, & Sands, 2016). This study was an attempt to fully develop an effective flipped unit of instruction on the topic of arrays in an Advanced Placement Computer Science A (AP CSA) class, with the aim of (1) improving students’ perceptions of computer science, (2) assisting teachers that are unqualified or new to the field of computer science, and (3) improving the quality of my own instruction. Ultimately, the thought was that if teachers that are unqualified or new to computer science had more guidelines on how to deliver a unit that improves students’ perceptions of computer science, the retention rate in computer science programs would eventually increase. This dissertation includes an introduction, literature review, methodology, intervention, results, and discussion and implications.

Background

In the same way students do not necessarily take math classes to become mathematicians, or history to become historians, computer science classes should not be limited to students interested in becoming computer scientists. Students with a computer science background are equipped with a skill set for careers in all areas (Repenning et al., 2015; Simard, Stephenson, & Kosaraju, 2010). Fortunately, the field
of computer science in secondary schools is growing. In addition to President Obama’s announcement of his Computer Science for All initiative in 2016, many school districts are now mandating that computer science be a graduation requirement (The United States Department of Education, 2016). Likewise, the number of students taking the AP CSA exam has grown 66% over the past five years (The College Board, 2018), which is more growth than any other subject the College Board has to offer, with the next closest subject at 59%. When taking into consideration AP Computer Science Principles (AP CSP), which was offered for the first time in the 2016-2017 school year, the combined number of students taking an AP Computer Science exam has grown by 31% just this past year, which is 17% more than any other subject.

Despite the growth, there are still many students dropping out of computer science programs due to a variety of reasons, such as the lack of a supportive learning environment, perceived usefulness of a computer science degree, and personal values (Giannakos et al., 2017). In addition, there is a lack of research in computer science teacher education (Armoni, 2011) as well as a shortage of trained computer science teachers (A. Lee, 2015; Margolis, Goode, & Bernier, 2011; Menekse, 2015; Qian et al., 2018), although efforts are being made to increase the number of qualified teachers (e.g., Lee, Dombrowski, & Angel, 2017). One study that included interviews with two dozen computer science teachers revealed that computer science teachers tend to feel isolated, lack a computer science background, and have little access to professional development (Yadav et al., 2016). This is problematic because the first computer programming course students take is considered to be crucial to the success of future computer science courses they take, making it imperative that the course is properly
designed so students develop good programming habits and positive perceptions of the field (Miliszewska & Tan, 2007; Pendergast, 2006). Properly designing the course is also important because students already tend to perceive computer programming as more difficult than similar subjects such as mathematics and physics (Aşkar & Davenport, 2009).

One solution to the imbalance of computer science offerings and trained teachers may be to implement blended learning (discussed below) classes that have one expert teacher and multiple local teachers who are less experienced and act more as facilitators (Basogain, Olabe, Olabe, & Rico, 2018). For example, teachers that are new to teaching computer science and/or lack the background could implement a blended classroom that has already been developed and contains all of the resources. Doing this would give them an opportunity to become more comfortable and confident with the content and both teachers and students will be less likely to form misconceptions since the materials are developed by more experienced teachers.

**Blended Learning**

Blended learning, also referred to as hybrid learning, involves merging traditional face-to-face learning with online learning. With so many definitions of blended learning, it would be challenging to find a learning environment that is not considered to be blended (Graham, 2006). A few definitions include: delivering between 30% and 79% of the content online (Allen, Seaman, & Garrett, 2007), combining face-to-face and computer-mediated instruction (Graham, 2006), and utilizing the best features of face-to-face and online learning (Watson, 2008). I simply view blended learning as merging face-to-face and online learning.
Flipped Learning

Flipped learning, a form of blended learning, can be thought of as “an educational technique that consists of two parts: interactive group learning activities inside the classroom, and direct computer-based individual instruction outside the classroom” (Bishop & Verleger, 2013, p.5). While I agree that students should engage in group activities, I would not say it is a requirement to be considered a flipped classroom. Additionally, I view active learning, which is an umbrella term for engaging students in the learning process (Prince, 2004), as a key component to a flipped classroom. As such, I see flipped learning as an educational technique that consists of two parts: active learning inside the classroom, and computer-based instruction outside the classroom.

Although flipped learning is a form of blended learning, to avoid confusion, from this point on it should be assumed that when the term blended is used, I am not referring to flipped unless otherwise stated. In other words, when I discuss flipped classrooms, I will explicitly use the term flipped, and when I discuss general blended learning that is not flipped learning, I will use the term blended.

The flipped environment is said to help students of all abilities, increase student-teacher and student-student interactions, diminish effects from absences, and allow students to learn at their own pace (Bergmann & Sams, 2012). This does not mean students can progress through the entire course at their own pace. Instead, students are able to watch the at-home videos at a pace that is comfortable to them since they can rewind and pause at any point, unlike direct instruction, where the pace is controlled by the teacher and to some extent, the class as a whole, depending on the questions they ask. Flipped classrooms also give students a chance to develop initial content
knowledge, procedural fluency, and problem-solving skills (Eisenhut & Taylor, 2015). This is likely because teachers are able to use the classroom time to identify student misconceptions, cultivate meaningful conversations, and support peer-to-peer instruction (Garrison & Vaughan, 2008).

**Computer Science**

Computer science, also sometimes referred to as informatics or computing (Fluck et al., 2016), takes on many definitions, ranging from broad to specific (McGuffee, 2000). I adopt the following definition of computer science: computer science is “the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society” (Tucker et al., 2006, p. 2). Computer programming, a tool in the field of computer science, is considered to be one of the more challenging subjects of computer science (Sarpong, Arthur, & Amoako, 2013) and will be the focus of this study. Computer programming is defined as the “knowledge of programming tools and languages, problem-solving skills, and effective strategies for program design and implementation” (Ala-Mutka, 2018, p. 1). An analogy to better understand the relationship between computer science and computer programming has been offered by Tucker et al. (2006) who state playing an instrument is to studying music as computer programming is to studying computer science.

**Subjectivity Statement**

I have been teaching at the secondary level for thirteen years between two public and three independent schools, all of which are in Florida. I have taught over twenty courses between the fields of mathematics, physics, and computer science. Next year will be my sixth year teaching mathematics and computer science at the school. I am
also in my second year as the head of the mathematics department. Although my B.S. is in physics and M.S. is in civil engineering, I have never taken a formal computer science course. With that in mind, the summer before arriving to the school, I attended a training to learn the basics of a computer science course, AP CSA, that I was slated to teach the following Fall. It was at this training when I realized how much I enjoyed computer science. When I started teaching at the school, computer science was only offered once every other year due to a combination of scheduling issues and a lack of interest. Fortunately, with the support of the administration, I have been able to grow the computer science program. We now offer three classes every year which include AP CSA, iOS app development, and web design and development. In addition, AP CSP is on the schedule for the 2019-2020 school year, with over 20 students already signed up for the school’s fourth computer science course.

Though the computer science program has grown, it has not been without challenges. While I certainly cannot speak for all teachers, I am well aware of my own insecurities when it comes to teaching concepts that are new to me. Naturally, I find it difficult to deliver my lessons with confidence in these situations, which I believe is rather transparent to the students. Despite my level of knowledge, my lack of confidence can sometimes impede on my ability to think clearly since I am worried about what my students are thinking and whether or not they are lacking faith in my ability to teach the material. I imagine many other teachers that are new to teaching computer science experience this, which is far from an ideal situation.

Although I initially explored the idea of flipped classrooms because a co-worker introduced me to it about seven years ago, I have found this to be a partial solution to
my lack of confidence in teaching concepts that are new to me. I have found that when I deliver new material through a video and give my students the class period to work on the exercises, I do not have a class full of students watching my every move. This alleviates a lot of pressure and allows me to think more clearly since I am able to work with a small group of students at a time. In addition, the classes tend to turn into two-way conversations with a common goal of understanding the material rather than a one-way delivery of information.

I also believe there are several other benefits to flipped classrooms. (1) Students can slow down, pause, and rewind the video at any point to collect their thoughts or if they are struggling to understand. (2) Students can re-watch the videos at any point, which is especially useful when it comes time to review for tests. (3) Absent students are at less of a disadvantage because they can still watch the videos at home. (4) Generally speaking, between taking notes and solving homework exercises, students are probably more likely to struggle with the exercises so they should have their peers and their teacher around during this process. (5) I am able to teach the material in a video faster than I would normally be able to in a traditional class because student behavior is not an issue and students are not asking questions throughout the lesson (admittedly a potential shortcoming of flipped classrooms). This is important because more time can be devoted towards practicing and getting into deeper conversations about the material. (6) The students who understand the first time around do not need to sit through all of their peers’ questions, facing potential boredom. Of course, there are disadvantages to flipped classrooms as well.
As mentioned earlier, students are unable to ask questions during the lesson, and in general, there is no interaction with traditional videos, causing a possible decrease in student engagement. Also, if students are struggling to comprehend the material through the video and cannot ask questions immediately, they may find themselves re-watching the video multiple times, ultimately increasing the time and effort required to learn. In addition, if students do not watch the video, then they will likely struggle with the in-class exercises.

With that in mind, I was not entirely convinced that a fully flipped classroom was the best option for every teacher, student, or subject/topic. At the same time, since the nature of a flipped classroom involves active learning, which I believe is superior to passive learning, I was not convinced that it had no value. As such, I attempted to be as objective as possible and to set aside my own beliefs so as to not influence the results of this study. I was interested in learning about students’ levels of engagement, satisfaction, and more generally about their perceptions of the different components of the flipped classroom. While a unit that is engaging and satisfying to students is ideal, it is important that it is effective as well, therefore I also wanted to look into how the students were performing.

Regarding the variables discussed above: Engagement is often broken up into affective, behavioral, and cognitive components, but definitions of these branches of engagement are not widely agreed upon (Skinner, Furrer, Marchand, & Kindermann, 2008). As a result, I did not attempt to distinguish between these branches and define engagement as an “emotion that ‘pulls’ students toward a subject matter and allows them to develop meaningful personal connections to a particular field of study” (Mazer,
Satisfaction can be thought of as the gratification derived from the class format and activities. Performance, in the context of this study, is defined as the student’s ability to design, implement, and analyze programs, which are three of the six main topics in the AP CSA curriculum (Computer Science A Course Description, 2014). Higher levels of performance were associated with a more effective unit/activity.

**Research Questions**

The purpose of this study was to develop a flipped unit of instruction by working with a focus group, implementing the unit with my students who were later interviewed, and then making final changes to the unit. The aim was to develop an effective unit that was both engaging and satisfying to students (1) so that students’ perceptions towards computer science would improve, (2) to help teachers new to the field of computer science, and (3) to improve the quality of my own instruction. With that being said, my research questions were as follows:

1. After initially designing a flipped unit of instruction, what recommendations were made by the focus group during the development phase?
2. To what extent was the unit implemented as designed?
3. What were students’ perceptions of the unit?
4. Was there any preliminary evidence to support the effectiveness of the unit?
5. What were the lessons learned and resulting final changes made to the unit?

**Overview of Research Design**

This design and development research focused on product development (Richey & Klein, 2014), and followed a two-phase mixed methods design. I first worked with a focus group to develop the unit and follow-up student interview questions that I initially designed. In the second phase, I implemented the unit with my AP CSA students,
collected daily field data, and interviewed students after the unit was complete. This data was then used to outline a set of lessons learned and to make final revisions to the unit.

The topic of arrays can be taught in the second half of the school year in my AP CSA course. As such, this topic was selected for the study in order to ensure enough time was set aside to work with the focus group before the unit of instruction began.

Initial Steps

Before the study started, I reached out to an AP CSA Facebook group for teachers that were willing to participate in the study, three of which responded and agreed to participate. In addition, I reached out to one of my former AP CSA students who also agreed to participate. This collaborative group served both as a focus group for developing the unit of instruction and follow-up interview questions as well as content reviewers. Before meeting with the group, the initial unit was designed based on my conceptual framework which will be discussed in Chapter 2. In addition, an initial set of interview questions was designed ahead of time (and later updated with the help of the focus group) and can be found in Appendix C.

Phase I

Before the unit was implemented with my own students, I worked with the focus group to further develop the initial unit that I designed, including the videos, activities, exercises, and follow-up interview questions. The unit was shared with the focus group and they were given time to review. We routinely met synchronously and communicated asynchronously as well. After each meeting, I made revisions which were shared with the group to validate the changes that were made.
Phase II

After the first few topics were developed and reviewed at least once by the focus group, the unit of instruction began with my students. I took field notes in class, wrote in a daily journal after class, and collected other field data throughout the course of the study. Once the unit was complete, I interviewed six of the students and later transcribed the interviews. The transcriptions and other qualitative data were analyzed using open coding, which were used to develop categories, which were finally used to generate themes. Based on the results from the data analysis, I made final revisions to the unit and outlined a set of lessons learned.

Definition of Terms

- **Active learning:** Any instructional method that engages students in the learning process (Prince, 2004, p.223).

- **Blended learning:** An educational technique that merges traditional face-to-face learning with online learning.

- **Cognitive Load Theory (CLT):** A theory based on the notion that working memory, which corresponds to short-term memory, has a fixed capacity (Sweller, 1988) whereas long-term memory is unlimited (Sweller, van Merrienboer, & Paas, 1998).

- **Cognitive Theory of Multimedia Learning (CTML):** A theory based on three principles: the human information-processing system (1) consists of dual channels (auditory and visual) and (2) has a limited capacity, and (3) a substantial amount of cognitive processing in both channels is a requirement in order for meaningful learning to transpire (Mayer & Moreno, 2003).

- **Computer programming:** The knowledge of programming tools and languages, problem-solving skills, and effective strategies for program design and implementation (Ala-Mutka, 2018, p. 1).

- **Computer science:** The study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society (Tucker et al., 2006, p. 2).
• Engagement: An emotion that ‘pulls’ students toward a subject matter and allows them to develop meaningful personal connections to a particular field of study (Mazer, 2013, p. 136).

• Flipped learning: An educational technique that consists of two parts: active learning inside the classroom, and computer-based instruction outside the classroom.

• Performance: The ability to design, implement, and analyze programs.

• Satisfaction: The gratification derived from the class format and activities.
CHAPTER 2
LITERATURE REVIEW

In this chapter, I will provide a historical context of K-12 computer science, highlight recent research studies, and discuss AP CSA in detail. Following that will be a discussion of blended learning, first at the secondary level both generally speaking and in terms of computer science, and then at the higher education level, specific to computer science. The discussion will focus on attitudes, perceptions, achievement, and recommendations, as these were the general themes that emerged as the literature was reviewed. The following sections will cover flipped learning in computer science at the secondary and higher education levels in terms of engagement, satisfaction, achievement, and recommendations. Again, these topics were not selected ahead of time, but instead surfaced as the general themes. The chapter will conclude with a discussion of two theoretical frameworks that served as a foundation for my conceptual framework, along with a discussion of my conceptual framework.

Methodology of Review

The literature reviewed in this study was identified through a variety of methods. First, databases including ProQuest, EBSCO, ERIC, ScienceDirect, and Google Scholar were used to search for existing literature that was peer-reviewed. The titles and/or abstracts needed to include three components: (a) blended, hybrid, flipped, or inverted, (b) some form of computer science (e.g., computer course, computer class, computer programming, etc.) and (c) a reference to secondary education (e.g., high school, secondary, eleventh, etc.) or higher education (e.g., university, college, etc.). Blended learning was included since there are components of blended courses that are found in flipped classrooms. General computer science (in lieu of computer programming
specifically) was included after very few studies were returned using the terms computer programming, specifically. As it turned out, many studies used the terms computer science in the title and/or abstract, but the focus was actually on computer programming. Both secondary and higher education was included because the study included high school students enrolled in a college-level course. More on this will be discussed later. In addition, the ACM and IEEE Xplore digital libraries were utilized. Since these are geared more towards computer science, I was able to be more specific with my search. Here, I searched specifically for flipped learning (rather than including blended learning) and computer programming (as opposed to general computer science). As the primary collection of articles was reviewed, relevant articles that were referenced within them were also obtained and reviewed. Additionally, I routinely received e-mails from Mendeley and ScienceDirect with suggestions for articles based on my search history and reviewed all relevant article suggestions. Several frameworks/theories emerged throughout the literature, so the same databases were again used to search for literature related to these frameworks.

**K-12 Computer Science Education**

**History**

The introduction of personal computers in the late 1970s and early 1980s brought with it computer programming education for millions of students (Resnick et al., 2009). Although, over time, enthusiasm for teaching computer programming dwindled for several reasons such as the level of difficulty of earlier programming languages, lack of consideration of students' interests/experiences, and the inability of teachers to provide proper guidance (Resnick et al., 2009). To help spread the instruction of computer science, the Association for Computing Machinery (ACM) model curriculum
was developed in 1993, but quickly became outdated mainly due to changes in the field of computer science itself (Tucker et al., 2006). While other efforts were made to develop and disseminate a computer science curriculum, there was still no widely accepted curriculum in the early 2000s. At this time, the number of students enrolling in computer science was rapidly declining, so much to the point that enrollment was cut in half over a period of only five years (McGettrick, Cassel, Guzdial, & Roberts, 2007; Moreno León, Robles, & Román-González, 2016). A few of the suggested reasons for the declining enrollment were negative perceptions of the field, a lack of understanding of what computer science was, and a lack of opportunities for students to take computer science classes in high school (McGettrick et al., 2007).

While computer science education appears to be making a comeback, there has still been a historical underrepresentation of girls, low-income students, and minority students studying computer science (Ravitz, Stephenson, Parker, & Blazevski, 2017). It has been suggested that one of the reasons for the underrepresentation of minority groups in computer science is the idea of the unconscious biases of teachers, administrators, and students (Dee & Gershenson, 2017). Dee and Gershenson (2017) have recommended several techniques for reducing these biases, such as increasing the general awareness of the biases, increasing interactions between social groups, and increasing empathy. Google Inc. and Gallup Inc. (2016) have reported both structural and social barriers that underrepresented groups face based on a recent two-year study. For example, one of the structural barriers was Black students’ lack of access to computer science classes at their schools, as compared to White students. One of the social barriers included what students were seeing on television. For
example, when students saw people on television “doing CS,” they often did not see people that were similar to them. Fortunately, many online resources such as Code.org (2018) and Khan Academy (2018), are continuing to grow and provide students with opportunities to learn how to code. In addition, initiatives such as Hour of Code (2018), which challenges students of all ages to engage in an hour of programming, and the Bebras Challenge (2018) continue to become more popular and can be effective tools for promoting problem-solving and computational thinking (Dagienė & Stupurienė, 2016). Computational thinking can be thought of as solving problems by breaking them down into logical steps and relating these steps to other situations.

While the literature appears to be lacking in the area of K-12 computer science retention rates, information is available regarding retention in computer science at the higher education level. Given the nature of the high school students enrolled in a college level course, I found these studies to be relevant and as such, included them in this discussion. As mentioned in Chapter 1, many students that enroll in a computer science program end up dropping out due to a variety of reasons, such as the lack of a supportive learning environment, perceived usefulness of a computer science degree, and personal values (Giannakos et al., 2017). Other factors such as a lack of time and/or motivation have also been reported as reasons students drop out (Kinnunen & Malmi, 2006). Likewise, introductory computer science courses that lack relevancy and are strictly limited to programming (and not the bigger picture of computer science) can lead to increased dropout rates (Biggers, Brauer, & Yilmaz, 2018). Students have additionally reported a variety of challenges in learning computer science, such as the inability to debug, managing their time, a lack of background knowledge, and access to
qualified tutors (Kinnunen & Malmi, 2006). On the other hand, factors such as attitudes, expectations, and comfort level appear to be predictors of students’ success in computer science (Kinnunen & Malmi, 2006).

While there is no widely accepted method for teaching computer programming (Leal & Ferreira, 2016; Moreno León et al., 2016), efforts are underway. The Computer Science Teachers Association (CSTA), a partner of ACM, recently developed a set of computer science standards (2017). Also, with the help of Code.org and CSTA, the International Society for Technology Education (ISTE) has developed standards for computer science educators (2018). Additionally, a committee consisting of members from ACM, Code.org, CSTA, the Cyber Innovation Center, and the National Math + Science Initiative was formed to develop a K-12 computer science framework (2016). This framework includes topics such as practices, concepts, implementation guidance, etc., and is not a set of standards, but instead a framework to help anyone interested in developing their own standards. Curriculums such as AP CSP and Exploring Computer Science (ESC) have also been developed and are recognized by many colleges (Yadav et al., 2015)

**Research In K-12 Computer Science**

Teaching computer science within K-12 programs can increase higher-order thinking skills, collaborative problem solving, and positive attitudes about computer science (Israel, Wherfel, Pearson, Shehab, & Tapia, 2015). There are two main types of programming languages found at the K-12 level: visual and text-based. Visual programming languages (e.g., Alice, Scratch) are generally used with younger/novice students since they reduce the cognitive load by limiting the syntax, allowing the students to focus more attention on the logic (Witherspoon, Schunn, Higashi, & Shoop,
2018). For example, students’ misconceptions were decreased when block-based (visual) programming languages were used in one study (Mladenović, Boljat, & Žanko, 2018). Using these visual programming languages has also be found to increase students’ attitudes toward computer science (Hayat, Al-Shukaili, & Sultan, 2017). Older students, on the other hand, tend to learn to program with text-based languages used in the real-world such as Python or Java (Israel et al., 2015).

One popular strategy that has been found to increase students’ motivation and participation in middle school computer science is to implement game design (Leal & Ferreira, 2016; Repenning et al., 2015). One such platform that has been used is Game Maker, which can help students learn to read and write simple programs based on examples and also how to identify and correct errors (Johnson, 2017).

Educational robotics programs can also be implemented in computer science classrooms to increase student engagement to a wide range of students (Witherspoon et al., 2018). Although, this can be very costly, so one solution to a lack of funds is to use virtual robotic environments, which has been found to be more feasible at the secondary level (as opposed to lower levels) and with more basic programming concepts (Majherová & Králík, 2017).

**AP Computer Science A**

The College Board currently offers two computer science courses: AP CSA and AP CSP. AP CSA focuses on programming in Java whereas AP CSP is meant to complement AP CSA by focusing more on the importance of appropriately communicating solutions in relevant ways (CollegeBoard, 2016). The six main topics in AP CSA include object-oriented program design, program implementation, program analysis, standard data structures, standard operations and algorithms, and computing
in context. Students in AP CSA have a minimum 20-hour lab requirement and sit for a three-hour exam at the end of the year which consists of 40 multiple-choice questions and four free-response questions (CollegeBoard, 2016). Over the last four years, the number of students taking an AP CS has more than doubled (39,000 to 104,000), thanks to initiatives such as CS for All and the launch of the new AP CSP course in 2016 (Cateté, 2018).

While AP CSA is a college-level course, it is important to remember that these students are still physically in high school. This is especially important to consider since the flipped classroom environment involves both in- and out-of-class activities and a high school student’s schedule likely looks very different than a college student’s schedule. For example, high school students in one flipped computer programming class were complaining about how tired they were after a packed day of classes and after-school sports (Cukurbasi & Kiyici, 2018). For this reason, literature in both secondary education and higher education was reviewed.

**Blended Learning**

Blended (or hybrid) learning, as mentioned in Chapter 1, involves merging traditional face-to-face learning with online learning, and has been around since the mid-1990s (Ferdig & Kennedy, 2018). With the prevalence of online materials available today and the number of definitions of blended learning, it would be challenging to find a learning environment that is not considered to be blended (Graham, 2006).

At the K-12 level, generally speaking, blended learning models can be classified into one of four models which include the rotation, flex, a la carte, and enriched virtual models (Staker & Horn, 2014). In a rotation model, students rotate on a fixed schedule between face-to-face and online activities. A flex model is very similar to online learning,
but with students learning at a predetermined time and location, and with a physical teacher that acts more like a tutor. In an a la carte model, students are enrolled in both face-to-face and online classes, but no single class in itself is blended. Finally, in an enriched virtual model, students split their time between face-to-face learning at a predetermined time and location and online learning which can take place remotely.

Although many aspects of blended learning in higher education are similar to those of secondary education, there are still structural differences (Staker & Horn, 2014). In addition, keeping in mind the subjects of this research are high school students enrolled in college-level classes, literature in blended learning at both the secondary and higher education levels was reviewed.

**General Blended Learning at the Secondary Education Level**

The number of classes being offered in a blended environment is rapidly growing (Miron & Gulosino, 2016), and efforts are well underway to research blended learning at the K-12 level. For example, Kennedy and Ferdig (2018) are on their second edition of their handbook of K-12 online and blended learning. This extensive collection of current knowledge in the field was made possible by the collaborative effort of more than 80 researchers. The handbook covers research on teaching, learning, student support structures, instructional design, and learning environments. It also provides a historical context and discusses the differences between different content domains, as well as online learning around the world. The handbook concludes with a discussion on some emerging issues with online and blended learning.

The literature in blended learning in secondary education generally points to positive student perceptions but contains mixed findings regarding learning outcomes. One study investigated teacher and student attitudes from 125 schools participating in
the 2011 iLearnNYC blended learning program (Hoxie, Stillman, & Chesal, 2014). The researchers found that students were generally positive about the blended experience and that teachers who created their own materials were more positive about the experience than teachers who utilized resources created by others. They also found that teachers favored the rotation model over the flex model. As a reminder, the rotation model consists of students rotating between face-to-face and online activities (e.g., flipped learning) whereas the flex model is similar to online learning, but with physical teachers acting more as tutors (Staker & Horn, 2014). Although these teachers were more positive about the rotation model, the flex model has been found to have its benefits. One study used the flex model to offer dual-credit mathematics courses which resulted in increased learning outcomes when compared to students taking the same classes face-to-face (Chao, Chen, & Chuang, 2015). Another study found that students were positive about their experiences in a blended program implementing an enriched virtual model, but found no significant differences in terms of learning outcomes (Siko, 2014). Students in this study came to class for about half of the week and spent the rest of the week learning from home. This study also stressed the importance of communication between teachers, students, and parents in blended environments. Another study looked at 47 schools implementing a blended program using the rotation model but found no evidence to support increased learning outcomes or motivation (Cavalluzzo, Lowther, Mokher, & Fan, 2012).

**Blended Learning in Computer Science at the Secondary Education Level**

Though many efforts are being made to better understand general blended learning, very little appears to be known about blended learning in computer science at the secondary level, specifically. For example, in the handbook of research on K-12
online and blended learning that was discussed earlier (Ferdig & Kennedy, 2018), the term “computer science” only appears twelve times. While that is admittedly not a sound argument for the claim that little research exists, it begins to paint a picture when that number is compared with the number of times “math” and “science” appear, which are 538 and 456, respectively. In addition, of the twelve times “computer science” was mentioned, half of them were simply found in the references or biographies of the authors. Of course, this handbook is not the only source available; still, there has been limited research in the area.

One study that used the Community of Inquiry framework (Garrison, Anderson, & Archer, 2000) to develop a blended high school computer programming course found that social presence was directly correlated with cognitive presence and positively associated with teaching presence (Pellas, 2016). Also, when given unlimited attempts to answer computer programming questions in an online environment, students’ confidence levels were found to increase over time (Basogain et al., 2018). It has also been recommended that GitHub, for example, which is an online platform for programmers to share code, is implemented for students to get help from experts (Garcia, Falkner, & Vivian, 2018). Although the field of blended computer science at the secondary level is highly under-researched, these studies demonstrate the potential for positive outcomes in blended computer science classes at the secondary level.

**Blended Learning in Computer Science at the Higher Education Level**

The literature on blended computer science at the higher education level, like the secondary level, mainly focuses on student attitudes, perceptions, and achievement. As a result of these studies, recommendations have been made for designing blended computer science classes. I will first discuss these studies in terms of attitudes and
perceptions, then achievement, and finish with a summary of recommendations found in the literature.

**Attitudes and perceptions**

It is important for students to have positive attitudes and perceptions of their learning environments since perceived satisfaction, usefulness, and interactivity in the online environment are all strong predictors of students’ self-regulated learning abilities (Cigdem, 2015). Multiple studies have found that students believe that learning computer science is done more effectively through a blended environment than a face-to-face environment (e.g., Eryilmaz, 2015; Geçer & Dağ, 2012). One study found that students perceived the online elements of a blended class as the most beneficial components (Geçer & Dağ, 2012). Also, web-based lectures were found to make computer science lectures more engaging, enjoyable, and easier to understand (Albinson, 2016). Other elements that students were satisfied with were videos and messaging tools for communication between teachers and students (Yağcı, 2017). Lastly, an online tool that provided automatic feedback was found to be highly favored amongst students when compared to paper-based materials (Pérez-Marín & Pascual-Nieto, 2012).

On the other hand, Delialioglu and Yildirim (2008) found no significant difference in students’ attitudes between blended and face-to-face environments containing the same elements. In addition, students have expressed concerns about the lack of teaching time when shifting from a face-to-face environment to a blended environment (Yağcı, 2017). Students have also claimed that the blended format requires more work when it comes to preparing for exams and participating online (Djenic, Krneta, & Mitic, 2011).
Achievement

Learning activities that were designed based on the social cognitive model of sequential skill acquisition (Zimmerman & Kitsantas, 1999) for a web-enhanced undergraduate object-oriented programming class yielded higher achievement over learning activities that were designed for traditional face-to-face learning (Georgantaki & Retalis, 2006). This model is based on the idea that students learn new skills in four steps which are observation, emulation (mimicking), self-control (comparing emulation with observation), and self-regulation (adapting emulation to match observation). Many studies in the area of blended learning have focused on self-regulation since learners with these skills are more likely to succeed in online environments (Cigdem, 2015). Self-regulated learning involves “the use of specified strategies to achieve academic goals on the basis of self-efficacy perceptions” (Zimmerman, 1989). Cigdem (2015) has recommended placing an emphasis on self-regulation during the online portion of a blended computer science course after finding no significant relationship between students’ self-regulation and their ability to learn how to program. This can be done by including elements such as online quizzes and online assignments.

In addition to self-regulated learning, co-regulated learning, which refers to the process of learners acquiring self-regulated learning skills through interactions between the experts and learners has been studied in blended computer programming classes. It was found that students that worked in teams and underwent web-based co-regulated learning activities acquired more computing skills than students that learned through traditional methods (Tsai, 2013). Since students learning computer programming are more successful when they are taught in a cooperative learning environment (Yağcı, 2016), and since students may struggle with self-regulated learning when implementing
self-regulated learning on their own (T.H. Lee, Shen, & Tsai, 2011), it appears to be more appropriate to employ a co-regulated learning techniques before self-regulated learning techniques when designing a blended computer programming class for secondary students.

Several models and tools have been developed that led to increased achievement in computer science classes. For example, a self-practice online tool (SPOT) was implemented in a blended computer programming class that increased student performance (El-Zein, Langrish, & Balaam, 2009). The SPOT is a risk-free tool for students to practice and monitor their own understanding in order to help them understand when to seek help and what kind of help to seek. An e-learning model that combined ideas of behaviorism, cognitivism, and constructivism was employed in another blended computer science course (Alonso, Manrique, Martinez, & Vines, 2011). The model was designed to increase individualized instruction, reduce space/time-restrictions, and allow students to work cooperatively. This was done by carefully considering the tools, techniques, and environments that were going to be used, as well as strategically determining what constituted “good problems” for students to solve. The model resulted in students outperforming those that were learning face-to-face, without the model. Similar to the e-learning model, the BLE-EDC (blended learning environment design using engineering design concept) model was developed and used to design a blended computer programming course (Titrakan, Kidrakarn, & Asanok, 2016). This was a five-step process that included: looking at the learning objectives, designing the learning environment, developing the learning environment and materials, implementing the course, and finally evaluating the course. The course design resulted in students
outperforming other students learning the same material in a face-to-face class. Impelluso (2009) was able to improve student achievement by using cognitive load theory (CLT) to design a blended computer programming course. In this course design, the extrinsic load was reduced by using online learning laboratories (desktop sharing) in order to leave more room for germane load. The germane load was maximized by using temporal scaffolding, specifically, by relating the programming topics to real-life examples.

Blended learning environments have not always yielded increased student achievement. Delialioglu and Yildirim (2008) used a model for learning and teaching activities (MOLTA) to design a blended computer science class and compared it with another class containing all of the same elements offered in a face-to-face environment. The only difference between the classes was the medium used to offer the elements. No significant difference was found between the two formats in terms of achievement. Likewise, Cakiroglu (2012) found no significant difference between blended and face-to-face offerings of a computer programming course in terms of students' achievement. In this study, students from the face-to-face section actually outperformed students in the blended section in a three-month delayed post-test. This could potentially be explained by the extra requirement for the students in the face-to-face section, who were required to use the whiteboard to show and explain their code, while this was not a requirement in the blended section.

**Recommendations**

Many researchers have made recommendations regarding the design of blended computer programming courses. Djenic, Krneta, and Mitic (2011) recommended placing more value on the roles of the teacher and students than on the structure of the course.
and the types of teaching materials. One such role of the teacher is that of a motivator. Since learning to program is best learned by practice, Jenkins (2001) has recommended that instructors focus their attention on motivating students to practice. One way to do this is by relating the material not only to real-world applications, but to other subjects as well (Stephenson, Gal-Ezer, Haberman, & Verno, 2005; Yadav, Mayfield, Zhou, Hambrusch, & Korb, 2014). Likewise, students can be motivated by providing them with laboratory time, projects, seminars, tutorials, peer-tutoring opportunities, group programming requirements, and problem-solving strategies (Sarpong et al., 2013). It has also been recommended that teachers raise awareness about what blended learning entails and to set clear expectations at the beginning of the class since students in a blended learning class might struggle with self-discipline, time-management, and the use of multiple technologies (Napier, Dekhane, & Smith, 2006). Hadjerrouit (2008) has suggested that it is actually more important for students new to programming to collaborate with their peers than it is for them to obtain high-order thinking skills. One way to promote this is for instructors to provide an environment that encourages students to ask and respond to questions in and outside of class. (Shen, Wang, Gao, Novak, & Tang, 2009).

Regarding programming techniques, Tritrakan et al. (2016) have recommended finding examples of code written by experts in order for students to form proper coding habits. Other techniques include comment first coding (CFC), analogies, and templates. CFC involves writing the comments before writing the code. Analogies are drawing back on known solutions. Templates are partially written code for students to expand upon.
Students implementing these techniques were found to outperform students that learned using traditional techniques (Cakiroglu, 2013).

**Flipped Learning in Computer Science**

Flipped learning, as mentioned in Chapter 1, is an educational technique that consists of two parts: active learning inside the classroom, and computer-based instruction outside the classroom. As a reminder, although flipped learning can be thought of as a form of blended learning, I will not use the term blended when discussing studies related to flipped learning. While very little literature exists specific to flipped computer science classes at the secondary level, there is a lot to be said at the higher education level.

**Flipped Learning in Computer Science at the Secondary Education Level**

One study that incorporated project-based learning by implementing LEGO applications found that students initially held negative attitudes towards the flipped classroom (Cukurbasi & Kiyici, 2018). Over time, though, their attitudes shifted as they started to see the benefits. Students reported benefits such as the ability to re-watch the videos, the increased interaction, and the efficiency of the environment. Likewise, another study found that students studying computer-aided design in a flipped environment had an overall positive experience (Chao et al., 2015). The students felt that their time was being used more efficiently, that they were actively involved in constructing their knowledge, and that the flipped environment helped them connect with their peers.

On the other hand, researchers in a study where students used an online course through Code.org to learn programming concepts found no significant difference in terms of student attitudes and their perception of how easy it is to learn programming
(Yan & Cheng, 2017). Although, they did find that the flipped classroom increased the students’ confidence in programming.

**Flipped Learning in Computer Programming at the Higher Education Level**

A review of the available literature in flipped computer science a few years ago revealed several benefits and challenges to flipping computer science classes (Giannakos & Krogstie, 2014). This review included 32 studies with various methodologies, designs, and variables of interest. The identified benefits were increased learning performance, positive attitudes, increased engagement, more discussions, supported cooperative learning, and better learning habits. The identified challenges were the increased initial cost and time commitment for the instructor, unreceptive students, and decreased attendance. Given the number of articles in flipped computer programming studies that have been published since Giannakos and Krogstie’s (2014) review, I was able to focus my review on programming, specifically. In line with the aforementioned review (Giannakos & Krogstie, 2014), the literature overwhelmingly suggests that students tend to be more engaged and satisfied when their computer programming classes are flipped. On the other hand, at best, it appears that students’ levels of achievement are also increased. At worst, the flipped environment leads to no significant difference.

The next three sections will review the literature in terms of engagement, satisfaction, and achievement. The final section will discuss recommendations that have been made by previous researchers.

**Engagement**

Many studies have reported increased engagement in flipped computer programming classes through the use of materials such as videos with quizzes, clickers,
guided practice worksheets. Active learning appears to be the central theme resulting in increased engagement. In a flipped classroom where the in-class activities were designed for collaborative experiences, students reported higher levels of engagement than traditional classrooms (Paez, 2017). The instructor utilized a combination of his own "how-to" videos and videos found online, as well as articles, books, and blog posts. These out-of-class resources were accompanied with a short set of multiple-choice questions in order to provide instant feedback to the students. In class, real-world tools and techniques were used in order to expose students to common practice. One such technique is that of retrospectives, where students learn to identify opportunities for improvement. Students in another flipped classroom that incorporated video quizzes were allowed multiple attempts on the quiz, but the average of all attempts was the score recorded to the gradebook (Hamid, 2016). Class time was used for focusing on key concepts/skills and was sometimes spent reviewing students’ work and on role-playing activities. Hamid (2016) suggested that the flipped classroom helped students of all levels in terms of their engagement and motivation. Moving beyond multiple-choice questions, an e-learning system can be used to provide semi-automatic feedback to students’ programs by running them with a set of test cases (Chen, Chen, Chang, & Yang, 2017). The researchers in this study reported that students were able to learn the concepts quicker and easier and therefore their programming skills improved. Additionally, students were writing more advanced programs and were often found walking around the room helping their peers when they were finished.

Largent (2013) was able to increase student engagement by using self-generated videos, a student response system, and active learning techniques. Students
reported that both concrete and abstract concepts could be taught through videos. Interestingly, students also felt that the text along with pictures increased their engagement, which the cognitive theory of multimedia learning (CTML) (Mayer, 2005) suggests should be done carefully. It is unclear exactly how the text was included in the video. In addition, a video of the instructor’s face was included in the corner of the video, which is considered to be bad practice according to CTML. The clickers, which were favored by the majority of the students in the class, were used at the beginning of each new section to assess the students’ current level of the material. Afterward, students worked in self-created groups of two to five students to work on the activities for the day.

In an attempt to increase student preparedness, Garcia (2018) implemented guided practice worksheets and found that students were highly motivated to complete them. These worksheets contained five components, including an overview of the reading/video, the reading/video itself, a list of basic objectives, a list of advanced objectives, and a set of problems to complete.

Simply flipping computer programming classes has not always led to increased engagement. One researcher attempted to increase motivation and interaction by incorporating pair-programming and regularly soliciting feedback for the flipped course but was unsuccessful (Towey, 2015). In line with Umapathy & Ritzhaupt’s (2017) advice that students need support and guidance with pair-programming, Towey (2015) found that simply placing students into pairs and teams was not enough to promote discourse. Students also struggled to adequately prepare questions and feedback for the
instructor, which the instructor assumed students would be able to do before the study began.

**Satisfaction**

Like student engagement, researchers have generally reported increased student satisfaction with the flipped environment. Students have reported enjoying the ability to watch the videos at their own pace, having help with the problems in class, coming to class prepared with questions, and the opportunity for active learning as opposed to passive learning (Sarawagi, 2014). In this study, students watched a 20 to 30-minute video at home. In class, the first 10 minutes was spent discussing content from the video. Students would sometimes take a short 15-minute quiz afterward, and the remaining time was spent working collaboratively on the exercises for the day. When compared to a traditional class, 95% of the students preferred the flipped class. Another study that incorporated collaborative learning had similar results and found that the majority of students were positive with their experiences in the flipped classroom (Hayashi, Fukamachi, & Komatsugawa, 2015). Students in this class were also held accountable through a quiz, but instead of taking the quiz in class the next day, students took the quiz online before arriving to class. Students in another class that were required to watch 5- to 15-minute videos containing embedded multiple-choice quiz questions every few minutes were more enthusiastic than students in a traditional class (Campbell, Horton, Craig, & Gries, 2014). Each video was created by one of the two instructors of the class. In class, students completed worksheets that contained a variety of exercises, including multiple-choice, short answer, tracing (reading/following code), and writing code using paper/pencil. They worked on these worksheets individually, but were frequently asked to discuss their responses with each other.
Although 81% of the students completed the out-of-class work, not as many attended the lectures. This may be attributed to the distribution of points since no points went towards attending the lecture. The researchers suggested that the lower attendance was not necessarily a negative consequence of the flipped classroom. They argued that many of the students that had a stronger background felt that they did not need to attend the lectures, which left fewer, weaker students in the class who were able to receive more attention. Still, the researchers suggested that there were students that did not attend that likely should have attended. One study implemented problem-based learning in class while watching videos with embedded quizzes out of class (de Oliveira Fassbinder, Botelho, Martins, & Barbosa, 2015). The in-class activities consisted of individual and group practice using pencil and paper, individually solving programming problems, and solving programming problems as a group in a contest style, which were gathered from an actual contest.

Indi (2016) carefully introduced the concept of a flipped class by initially keeping the videos short and the end-of-video tasks simple in order to increase student interest. Students appeared to be very receptive to the new environment considering 90% of them agreed that they would like to see the flipped classroom environment implemented in their other classes. Additionally, 98% of the students reported that the in-class discussions helped them improve their understanding of the concepts. While many instructors attempt to keep the length of the videos to a minimum, it appears that this is not a strict requirement. One study that implemented videos ranging from three minutes up through nearly 40 minutes still resulted in very satisfied students (Heralu, Vanhala, & Nikula, 2015).
Although many instructors create their own resources or use a combination of their own with materials found online, it is possible to flip classrooms using strictly free, online resources. One study used lectures recorded from Stanford University, which were hosted through iTunes University, for the out-of-class learning and focused on student reflections (Thomas, 2014). After watching the video, students were required to come to class with the notes they took and at least six questions they had from the video. At least three hours before class began, students were required to post their top three questions to a class forum which allowed the instructor to identify trends and to better prepare for the class. In class, each student read their two most important questions to the rest of the class. At the end of the week, they shared something useful that they learned from the previous week’s lab. The majority of students reported that they were growing intellectually and learning new skills. Jonsson (2015) used a slightly similar approach in terms of student responses driving the in-class discussions. In this study, students were required to submit their hand-written code in class (placing their name on the back of the piece of paper so they remained anonymous). The class would then review and criticize several randomly selected submissions. Students reported that the course was more meaningful and rewarding than students studying the same concepts in a traditional classroom.

While the majority of studies have found that the flipped classroom increases student satisfaction, there is at least one study that reported a decrease in satisfaction. This study utilized free, online resources and resulted in dissatisfied students that were resistant to the out-of-class activities and felt that they were being left to learn on their
own (Baldwin, 2015). The main complaint from students was the lack of consistency between the different sources.

**Achievement**

Unlike student engagement and satisfaction, which are generally reported to increase when computer programming classes are flipped, the literature provides mixed findings on student achievement. It appears that at best, student achievement is increased and at worst, there is no significant difference.

Two researchers took a three-step approach to teaching binary search and selection sort through a series of three videos (Schreiber & Dougherty, 2017). The first video focused on using intuition by relating what the students already know to connect to the material. The second video included the analysis of the algorithms along with guided examples. The third video wrapped everything up with a song and lyrics in the form of pseudocode. Students that watched the video series showed substantial improvement in their understanding of the algorithms, although there was no control group to compare the results with.

Many researchers have found that flipped classrooms that incorporate quizzes with the video tend to increase student achievement. A flipped classroom environment discussed earlier (Jonsson, 2015) that resulted in increased satisfaction also resulted in increased achievement. Students in this class were required to watch videos out of class that contained embedded multiple-choice quiz questions. The videos paused at each question and would not let students progress until the student answered correctly. They were given unlimited attempts and the quiz did not count for a grade. In class, students worked on programming problems on paper and wrote their names on the backside in order to stay anonymous. Students were encouraged to discuss their
problems with their peers as they worked on them. The instructor collected a random sample of solutions which were displayed through a projector. The class would then, as a whole, criticize each solution. Using this format, 81% of the students passed the course as compared to only 60% in the traditional class. Additionally, of those that passed the class (which required a score of 3, 4, or 5), almost twice the number of students in the flipped class did so with good grades (score of 4 or 5) than those in the traditional class. Another flipped class required students to watch a 10 to 15-minute video and answer three to ten multiple choice questions at home. In class the next day, students spent the first 30 minutes discussing misconceptions from what they learned at home and then worked on additional exercises. This resulted in students scoring, on average, 17% higher on an exam than students in the traditional offering of the class (Elmaleh & Shankararaman, 2017). There were also 20% fewer failures as well as higher minimum and maximum scores. Students in the flipped class were also given an additional set of lab exercises to work on after the class met. Another study compared two different flipped classrooms, one that contained embedded video quizzes and one that did not (Lacher, Jiang, Zhang, & Lewis, 2018). The embedded questions were a mixture of multiple-choice and free-response questions. Students had only one attempt on the multiple-choice questions and unlimited attempts on free-response questions. Both styles of questions were auto-graded, and students were not given any feedback when they were wrong with the free-response questions, other than a message indicating that they were incorrect. In line with the previous studies, the researchers found that students in the section with embedded quiz questions outperformed students in the other section. They also found the students who had a higher aptitude in
computer science benefited the most from the embedded quiz questions. Similarly, Seeling (2016), who used online textbooks that automatically graded students' answers and supplied them with feedback, found a correlation between students' performance on these exercises and how they did in the overall course. As expected, students that performed well on the exercises did better on the quizzes and in the course as a whole.

On the other hand, one study found that more students in a flipped classroom failed the midterm than students in the traditional class (Horton & Craig, 2015). Interestingly, the students in the flipped class who failed the midterm were much more likely to either completely drop the course or recover from their bad score and pass the course than students in the traditional class. In other words, the authors suggested that the flipped class had a “go big or go home” (p. 240) effect on students. Another study found that students in a flipped class performed worse than students in a traditional class on online quizzes (Seeling, 2016). Although, these students performed better on the final exam and their overall achievement in the course was comparable. Students in this class solved exercises in an online textbook that automatically graded students' answers and supplied them with feedback. Similarly, students in another flipped class performed significantly better on the final exam than students in a traditional class (Horton, Craig, Campbell, Gries, & Zingaro, 2014). On the flipside, these students were no more likely to pass the overall course. Students in this class watched about one hour of videos with embedded quiz questions per week, outside of class. The quizzes were graded only on attempts and attendance for the lectures was not required. During the lectures, students worked individually on exercises that were related to the videos.
One flipped classroom utilized 20 to 30-minute videos for students to watch before each lesson (Tyler & Abdrakhmanova, 2016). In class, the instructors gave a 5-to 10-minute overview of concepts discussed in the video and then students spent the remaining time working individually or in pairs on a few exercises that were due before the end of the class period. Occasionally, the students would take a short quiz at the beginning of the class. While student feedback was positive, the researchers found no significant difference regarding student achievement. Likewise, the study described earlier that resulted in more enthusiastic students (Campbell et al., 2014), resulted in no significant difference in terms of achievement.

**Recommendations**

As a result of the aforementioned studies as well as others that have not yet been discussed, several recommendations have been made for the design of a flipped computer programming class. Two groups of researchers focused their efforts strictly on the design and execution of the in-class activities and made several recommendations. Köppe et al. (2015) made five specific recommendations. (1) The pace of the class should be carefully controlled by setting deadlines. (2) Students’ prior experience should be considered by structuring the classroom activities based on their homework solutions. (3) Interesting demonstrations and real-world examples should be combined when giving feedback to students. (4) All students should see their own work criticized from time to time, as exemplified by Jonassen (2014). (5) Students should see multiple solutions to the same problem and discuss the strengths and weaknesses of them. A year later, Köppe collaborated with another team (Köppe, Niels, Bakker, & Hoppenbrouwers, 2016) to publish another set of recommendations. (1) Students should know that the instructor is aware of each student’s status in terms of their out-of-
class participation. (2) Optional additional resources based on the standard material should be made available for students that are struggling. (3) Optional additional resources that are more challenging than the standard material should also be made for the more advanced students, as also recommended by Hamid (2016). (4) Students should be given opportunities for individual support. (5) Feedback and instructions for small groups should be adjusted to meet the group’s ability level.

Sarawagi (2014) has suggested that passive content should be introduced in the out-of-class videos so the students can learn at their own pace and so the in-class time can be used for higher levels of learning. Since classrooms tend to have students with different levels of ability, instructors should be prepared to give the higher achieving students interesting challenges in order to keep them engaged if and when they finish the work early (Hamid, 2016). If it is important for students to complete the out-of-class activities, then they should be given motivation to do so, such as a reflection prompt requirement (Dazo, Stepanek, Fulkerson, & Dorn, 2016). Another method for providing this motivation might be through mandatory quizzes (Herala et al., 2015). Students should also be encouraged to complete the in-class activities if they are not able to finish them by the end of class, which can be done by assigning points for the completed work (Largent, 2013).

When students are working on exercises out of class, they should submit their work electronically and be provided with feedback immediately (Garcia, 2018). Garcia has also recommended adding post-exercises for students to complete after the in-class activities.
Lastly, after receiving complaints from students about a lack of consistency between videos, it has been recommended that regardless of the source, videos should be from the same source (Baldwin, 2015; Knutas, Herala, Vanhala, & Ikonen, 2016).

**Theoretical Frameworks**

Two frameworks surfaced throughout this review: active learning and the cognitive theory of multimedia learning (CTML). Active learning techniques helped to inform the structure the in-class activities whereas principles of CTML helped to inform the development of the videos.

**Active Learning**

Grounded in constructivism, active learning, which was popularized in the early 1990s (Mitchell, Petter, & Harris, 2017), is “an educational theory to improve ability by independently learning the applied-exercises involving collaboration among many students” (Hayashi et al., 2015, p.209). It can more generally be thought of as an umbrella term for “any instructional method that engages students in the learning process” (Prince, 2004, p.223), which is in contrast to traditional lecturing. Active learning has been found to positively influence students' attitudes and achievement and is generally accepted to be superior to traditional lecturing (Bonwell & Eison, 1991; Freeman et al., 2014; Prince, 2004). For example, active learning techniques have been found to be a significant predictor of student success (Serva & Fuller, 1999). Studies have also shown that students not only prefer active learning techniques over traditional techniques, but their attitudes as well as thinking and writing skills improve (Bonwell & Eison, 1991).

There are many strategies to incorporate active learning, including collaborative learning, cooperative learning, and problem-based learning (Prince, 2004).
Collaborative learning involves peer interaction, with students working together towards a common goal. Cooperative learning, where students are responsible for their own sub-goal, which are pieced together to achieve a common goal, can be thought of as a form of collaborative learning (Elazab & Alazab, 2015). While problem-based learning can be done collaboratively, cooperatively, or individually, it appears that when students work together, they tend to outperform students that work individually (Linton, Farmer, & Peterson, 2014). Other strategies include debates, role-playing, and simulations (Bonwell & Eison, 1991). Debates, where students take a standpoint and present their arguments, could be hosted in-class or even online in a forum, for example, and can be used to learn about students’ emotions (Arguedas, Daradoumis, & Xhafa, 2016). Role-playing involves students taking on a specific role within a group, and has been found to increase student achievement in introductory computer science courses (Shebaro, 2018). Simulations can be carried out on a variety of tools such as graphing calculators and web tools and are considered to be an essential component to discovering and understanding fundamental rules in computer science (Tucker et al., 2006).

In science, technology, engineering, and mathematics (STEM) classes, common approaches to incorporate active learning include worksheets, tutorials, personal response systems, and peer-instruction (Freeman et al., 2014). Worksheets can be used in or out of class and serve to reinforce concepts (Yadav et al., 2015). Tutorials can be used to teach the basics of the software that students will be using (Johnson, 2017), which may help to reduce the cognitive load on the students. One illustration of peer-instruction involves students answering a multiple choice question using clickers, which is an example of a personal response system, discussing their answers with their
peers, and then answering the question a second time (Simon, Kohanfars, Lee, Tamayo, & Cutts, 2010). In this study, which was specific to an introductory programming class, student achievement improved, and both students and teachers were generally positive about their experiences.

Active learning has been considered to be an effective approach in the field of computer science for some time (McConnell, 1996). Effective techniques in computer science include algorithm tracing, using demonstration software, and including physical activities (McConnell, 1996). Algorithm tracing might involve students working in groups to compare different sorting algorithms, for example. Students take on different roles, such as the tracing the code, keeping track of the variables, and performing the actions. Demonstration software could be projected and students could be asked to work in groups to predict what the output of a program would be based on different inputs. One physical activity that could be used is role-playing. For example, network protocols could be better understood by having students pass around a ball and only allowing students to talk when they have the ball. Pair-programming is another technique that has potential for improving student achievement (Umapathy & Ritzhaupt, 2017). Pair-programming is a form of collaborative learning since it involves one student as the driver (controlling the keyboard and mouse) and the other as the navigator, but not cooperative learning since the students do not have their own sub-goals. When implementing pair-programming, it has been recommended that students switch roles often to make sure they stay engaged (Williams, McCrickard, Layman, & Hussein, 2008).
Sometimes, it may be necessary for instructors to lecture, depending on the content/skill to be learned. Since student attention begins to decline after ten to fifteen minutes, one method to address this is to limit the lecture time to ten minutes, and then have students engage in a five-minute activity, such as discussing their notes with their partner (McConnell, 1996).

The majority of the studies mentioned earlier in this review utilized some form of active learning techniques for the in-class activities. The nature of the flipped classroom environment allows for active learning opportunities in the classroom, and given the positive effects of active learning, it served as the foundation for the in-class activities in this study.

**Cognitive Theory of Multimedia Learning (CTML)**

CTML is based on three principles: the human information-processing system (1) consists of dual channels (auditory and visual) and (2) has a limited capacity, and (3) a substantial amount of cognitive processing in both channels is a requirement in order for meaningful learning to transpire (Mayer & Moreno, 2003). Essentially, when students view a multimedia presentation, they are viewing images and seeing or hearing words. Students select the relevant images and words, organize them in their working memory, and then integrate the images and words with prior knowledge (Mayer & Johnson, 2008).

It has been suggested that principles of CTML be kept in mind to help teachers create engaging videos (Chung & Khe, 2017). There are many basic principles of multimedia learning that are commonly accepted and summarized in Table 2-1 (Mayer, 2014). These principles were followed as the videos were developed for this study.
Table 2-1. Principles of multimedia learning.

<table>
<thead>
<tr>
<th>Principle</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multimedia</td>
<td>Words should be supported with images.</td>
</tr>
<tr>
<td>Modality</td>
<td>Images are better supported with narration than with text.</td>
</tr>
<tr>
<td>Redundancy</td>
<td>Information should not be presented in multiple formats.</td>
</tr>
<tr>
<td>Segmenting</td>
<td>Presentations should be broken up into short, focused segments instead of a long, continuous unit.</td>
</tr>
<tr>
<td>Pretraining</td>
<td>The names and characteristics of the main concepts should be taught before the presentation.</td>
</tr>
<tr>
<td>Coherence</td>
<td>Extraneous material should not be included.</td>
</tr>
<tr>
<td>Signaling</td>
<td>Images and text should be highlighted, circled, etc., as they are discussed.</td>
</tr>
<tr>
<td>Spatial contiguity</td>
<td>Images and text should be physically placed close to each other.</td>
</tr>
<tr>
<td>Temporal contiguity</td>
<td>Images and text should appear at the same time.</td>
</tr>
<tr>
<td>Personalization</td>
<td>Presentations should be informal/conversational.</td>
</tr>
<tr>
<td>Voice</td>
<td>Words should be spoken in a friendly, human voice as opposed to a computer voice.</td>
</tr>
<tr>
<td>Image</td>
<td>The speaker’s image does not necessarily need to be on the screen.</td>
</tr>
</tbody>
</table>

These principles are not as straightforward as they may seem, as they should be considered as a whole, and not individually. For example, the coherence principle states that extraneous material should not be included in the presentation, but there may be an opportunity to include interesting information that is extraneous, such as making a connection between the content and its application in the real world. In this case, the extraneous material can be included, but the presentation should use the signaling principle to help focus the students’ attention (Ibrahim, 2012). In another example, the redundancy principle states that information should not be presented in multiple formats, but there are times this is okay. For instance, redundancy can support learning when text is placed close to the images (spatial contiguity principle) and when it is highlighted (signaling principle) (Mayer & Johnson, 2008).
One of the underlying theories of CTML is cognitive load theory (CLT). CLT is based on the notion that working memory, which corresponds to short-term memory, has a fixed capacity (Sweller, 1988) whereas long-term memory is unlimited (Sweller et al., 1998). Working memory is said to be split between three types of cognitive loads: intrinsic, extraneous, and germane (Sweller et al., 1998). Intrinsic load cannot be altered since it is based on the material being learned. On the other hand, extraneous and germane loads can be altered; decreasing one increases room for the other. Extraneous load is the load caused by the design of the instruction, and germane load refers to the effort required to committing ideas to long-term memory. With this in mind, instructional designers should aim to decrease the extraneous load through the presentation materials, leaving more room for germane load so students can commit their learning to long-term memory.

Computer programming involves solving complex problems, which often need to be split into sub-problems, and creating generalizable solutions, and is considered to be one of the most challenging computer subjects to learn (Zhang, Kalyuga, Lee, & Lei, 2016). This may be due to the high intrinsic load associated with learning computer programming since novice students are learning the procedures along with the language at the same time, analogous to learning math and how to read/write numbers at the same time (Margulieux, Catrambone, & Guzdial, 2016). While this is not always the case, efforts should be made to recognize the difference between high and low levels of intrinsic load in order to optimize the learning experience. For example, if graphic organizers, which have been found to improve critical thinking skills, are going to be used, then they should be given to students in situations of high intrinsic load.
Otherwise, if the intrinsic load is low, students should generate them on their own to make the learning more meaningful (Phan, 2011). In addition to the high intrinsic load already associated with learning computer programming, the potential to overload cognitive abilities increases further when taking into consideration that students are also learning how to utilize the interface that runs the code. Coupling this with a new learning environment such as a flipped classroom and the possibility of cognitive overload is even higher. Therefore, CLT and principles of CTML were kept in mind when developing the videos for the flipped computer programming course.

**Conceptual Framework**

My conceptual framework, which is essentially a set of design considerations for a flipped computer programming course, was developed based on two theoretical frameworks (active learning and CTML) and recommendations made by previous researchers. A summary of these recommendations and principles can be seen in Figure 2-1. Starting with the top left of the conceptual framework, the design considerations for the out-of-class videos are listed, which are based on the principles of CTML. Underneath is a list of common themes identified from the literature regarding recommendations made by researchers. Next, the top right contains the main categories of active learning as well as various activities specific to computer programming that have been recommended. Finally, the bottom of the conceptual framework contains a list of general suggestions based on the results of several studies, again, specific to the field of computer programming, both in terms of what the teachers and students should be doing. In Chapter 4, I will pull together the recommendations from this conceptual framework with well-established instructional design principles to describe the unit of instruction that was developed.
Figure 2-1. Conceptual framework for this study.

<table>
<thead>
<tr>
<th>Out of Class</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CTML Principles</strong></td>
<td><strong>Active Learning</strong></td>
</tr>
<tr>
<td>Multimedia</td>
<td>Collaborative learning</td>
</tr>
<tr>
<td>Split-attention</td>
<td>Cooperative learning</td>
</tr>
<tr>
<td>Modality</td>
<td>Problem-based learning</td>
</tr>
<tr>
<td>Redundancy</td>
<td>Voice</td>
</tr>
<tr>
<td>Segmenting</td>
<td>Personalization</td>
</tr>
<tr>
<td>Pretraining</td>
<td>Image</td>
</tr>
<tr>
<td>Coherence</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Videos</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Embedded quizzes</td>
<td>Pair programming</td>
</tr>
<tr>
<td>Automatic/immediate feedback</td>
<td>Animations/Simulations</td>
</tr>
<tr>
<td>From the same source</td>
<td>Physical activities</td>
</tr>
<tr>
<td>Passive content</td>
<td>Role playing</td>
</tr>
<tr>
<td></td>
<td>Critique programs</td>
</tr>
<tr>
<td></td>
<td>Personal response systems</td>
</tr>
<tr>
<td></td>
<td>Peer instruction</td>
</tr>
<tr>
<td></td>
<td>Guided practice worksheets</td>
</tr>
</tbody>
</table>

**Teachers Should:**
- Set clear expectations
- Set strict deadlines
- Communicate with everyone involved
- Motivate students to practice
- Make students aware that they can see their students progress
- Have additional, reinforcing exercises for struggling students
- Have additional, challenging exercises for advanced students
- Provide an environment that encourages students to ask and answer questions outside of class
- Consider students’ quiz responses to drive the in-class activities
- Use real-world examples

**Students Should:**
- See their work criticized
- See multiple solutions to the same problem
- Discuss strengths and weaknesses of programs written by others
- Have opportunities for individual support
- Share useful information that they learned
- Post questions they have from the video
- See code written by experts
- Write code by hand and on a computer
Summary

Due to the lack of existing literature in the specific area of flipped computer programming at the secondary level and the nature of high school students enrolled in college-level courses, this literature review included studies on K-12 computer science as well as blended and flipped learning at the secondary and higher education levels. Literature on active learning and CTML was also reviewed in order to inform the design of the in-class and out-of-class activities and videos. While the majority of the literature was situated at the higher education level, there is a lot of evidence to suggest that flipping a computer programming classes at the secondary level can result in increased satisfaction, engagement, and performance.
CHAPTER 3
METHODOLOGY

This chapter outlines the research design and general procedures for this study, including the data sources and analysis, specifically in relation to the five research questions: (1) After designing a flipped unit of instruction, what recommendations were made by the focus group during the development phase? (2) To what extent was the unit implemented as designed? (3) What were students’ perceptions of the unit? (4) Was there any preliminary evidence to support the effectiveness of the unit? (5) What were the lessons learned and resulting final changes made to the unit? This chapter also describes the setting and participants, ethical considerations, validity, and limitations.

Research Design

This study followed a design and development research design. Design and development research is “the systematic study of design, development and evaluation processes with the aim of establishing an empirical basis for the creation of instructional and non-instructional products and tools and new or enhanced models that govern their development” (Richey & Klein, 2014, p.1). This pragmatic approach allows for the development of new procedures, techniques, and tools.

This type of research typically stems from issues in the workplace, emerging technology, or general theory-based questions (Richey & Klein, 2014). With the lack of qualified teachers (Qian et al., 2018), increasing number of students enrolling in CS (Catetê, 2018), and low retention rate in CS programs (Giannakos et al., 2017), in conjunction with my own lacking CS background, I embarked on design and development research as a result of this compounded workplace issue. The aim of this
study was to (1) develop a unit that would promote positive perceptions of computer science, (2) provide advice for teachers that are unqualified or new to the field of computer science, and (3) improve the quality of my own instruction so that my own students would directly reap the benefits.

Design and development research can be broken up into two types: (1) product and tool research, and (2) model research. Product and tool research can focus on the entire design and development process: including the design, development, implementation, and evaluation, or can focus on particular components of the process. The specific products (or programs) or tools that serve as the focus of the study can be for instructional or non-instructional purposes. Model research, on the other hand, concentrates on a general model (or process or technique). Three types of model research include model development, model validation, and model use. Model development research can focus on the entire design and development process or just part of it. Model validation research focuses on the validation of already existing models. Model use research focuses on usability issues with models, so a focus might be on identifying conditions that affect the model, for example.

With my goal of developing an instructional unit, this study falls under the comprehensive product development umbrella. Comprehensive product and tool research can be comprised of methods such as case studies, content analysis, evaluation, field observations, and in-depth interviews (Richey & Klein, 2014), all of which will be implemented in this study. The initial design of the unit, which was based on the conceptual framework in Chapter 2, was completed before the study began and will be described in detail in Chapter 4 (along with the updates based on the work with
the focus group). The results of the development, implementation, and evaluation phases will be described in Chapter 5.

All research can be classified into exploratory, descriptive, or explanatory (Richey & Klein, 2014). Exploratory research deals with topics that are relatively unknown/new, descriptive research focuses on identifying facts, and explanatory research describes phenomena and seeks to find solutions to problems. With the lack of literature in secondary flipped computer science and the aim of developing of an instructional unit that promotes student engagement, satisfaction, and performance, this study was both exploratory and descriptive in nature and contained two phases. The first phase consisted of working with a focus group of three AP CSA teachers and one former AP CSA student to develop the unit (research question #1 - RQ1). The second phase consisted of implementing (RQ2) and evaluating (RQ3 & RQ4) the unit. The results from the first four research questions were taken into consideration to finalize the unit (RQ5). For the sake of clarity, it may be beneficial to review Figure 3-1 before reading about the finer details of the design.

![Figure 3-1. Research design overview.](image_url)
A qualitative approach was taken in the first phase by working with a focus group for feedback and recommendations on modifications to the unit of instruction. A focus group was used because it can help “articulate, understand, and refine a shared platform of design ideas among developers and/or experts. [It] can also provide a slightly more distanced opportunity (than interviews) for participants to discuss their perceptions and experiences of the intervention” (McKenney & Reeves, 2018, p. 147). In addition to routinely meeting synchronously online, another role of the members of the focus group was to serve as content reviewers and to communicate with me asynchronously or synchronously to report mistakes, comments, and/or questions they had. The second phase took the form of a case study. A case study approach involves exploring “a bounded system (a case) or multiple bounded systems (cases) over time, through detailed, in-depth data collection involving multiple sources of information (e.g., observations, interviews, audiovisual material, and documents and reports), and reports a case description and case-based themes” (Creswell, 2013, p. 73). The case study approach was selected because I had a clearly identifiable case with boundaries, which was my AP CSA class, and my goal was to gain an in-depth understanding of my students’ perceptions and to investigate the preliminary effectiveness so I could further develop the unit.

**Setting and Participants**

The study took place over a period of three weeks starting in January 2019 at a small independent preK-12 school in the southeastern United States. The school runs on a seven-day rotating schedule with ten periods a day, Monday through Friday, 8:00 am to 3:25 pm. The core periods last 43 minutes and meet six out of the seven days. The school has an enrollment of nearly 600 students, 25 of which are international
students, has a 13:1 student/faculty ratio, and a student attrition rate of 9%. The graduating class of 2018 had average SAT reading and math scores of 625 and 600, respectively, and 100% of the students were accepted to college, 90% of which were accepted to their first or second choice college. Of the AP exams for all subjects that were taken by students at the school in 2018, 94% were passed. Over the past four years, the average AP CSA scores for my students were as follows: 2018 – 3.0 (n=3), 2017 – 4.4 (n=17), 2016 – 3.1 (n=9), 2015 – 2.5 (n=14).

The classroom, which was converted from an athletic suite to accommodate the need for more classrooms, was on the second floor and accessible from an outside hallway and was the only classroom in the building. Other rooms in the building were the gymnasium, locker rooms, and offices. The classroom contained 16 counter-height, 4-person tables, some of which were pushed together, arranged in two rows. The tables were arranged in such a way that students could face the front of the classroom, and at any point, turn around to work with a larger group. In other words, students sitting at a 4-person table (two 2-person tables pushed together) always faced the front, while students at the 2-student tables turned around for the group activities. See Figure 3-2 for a rendering of the classroom layout. The walls were also lined with whiteboards that students were given full permission to use at any point.

Figure 3-2. A rendering of the classroom used in this study.
The class met during the first period of the day (six of the seven days) and contained eleven students. Eight of the eleven students agreed to participate in the study. Of those participating, three were female and five were male. One student was African American, one was Asian, and six were Caucasian. The Asian student was enrolled through the school's international program and had been studying at the school for over two years and planned on graduating from the school. Five of the students were Seniors and the other three were Juniors. Two of the students took my iOS app development class the year before the study, one of the students studied Java on their own for a science fair project, and the remaining five students were new to the field of computer science.

Additionally, three AP CSA teachers from other schools and one former AP CSA student, a Senior, served as the focus group. Two of the teachers were females, one was male, and the average age was 40. The teachers had an average of 13 years of general teaching experience and 9 years of experience teaching AP CSA, specifically. Two of the teachers had a B.S. in computer science and the third was in the process of working on their Ph.D. in computer science. One of the teachers was also working on their Ed.D. in Curriculum and Instruction, with a research focus on computer science. All three of the teachers had experience as AP CSA graders, with a collective total of eleven years of experience grading AP CSA exams. The group also included a chapter president of the CSTA. The former student took AP CSA during his Freshman year and earned a four out of five on the AP exam. He worked as my teaching assistant the following year and retook the AP exam at the end of his sophomore year and earned a five out of five.
Procedure

Before the study started in early January 2019, the three AP CSA teachers, former AP CSA student, and my current AP CSA students and their parents were asked to fill out a consent form. Participation was not mandatory, and students and their parents were given time to ask questions about the study before deciding. Three of the eleven students did not return their consent form, leaving eight students in the class as participants. Being both the researcher and the teacher posed a potential ethical issue, which will be addressed later in this chapter.

Phase I

The primary goal of the first phase was to further develop the initial design by considering recommendations made by the focus group (RQ1). As secondary goals, the same group also served as content reviewers and also helped to develop the follow-up student interview questions that I initially created.

Before the study began, I designed the instructional unit and student interview guiding questions based on my conceptual framework. I created a Google Site and uploaded the entire unit for the focus group, with each day and each night as a different webpage. I also created a class on EdPuzzle where all of the videos were hosted. Once the consent forms were obtained, the focus group was sent an invitation to view the website and a link to join the EdPuzzle class. After the focus group had an opportunity to review the materials, we met online using Zoom, a video conferencing tool. We had a specific agenda each time (Appendix C), which was to discuss their thoughts and recommendations for the given topic(s). After each focus group meeting, I made the appropriate revisions to the unit on the website and sent out a list of changes that were made. After the changes were made, the group was asked to review the changes to
confirm whether or not I accurately understood their recommendations. Once the group was satisfied with the changes, we moved on to the next topic and the process repeated.

The same group was invited to a private Slack team. Slack is an online communication tool that allows for conversations via text in a private forum. Multiple channels can be created in Slack to help organize conversations. One channel was devoted to mistakes so participants in the focus group could report mistakes they found with the material as they were reviewing it. Another channel was devoted to general comments, allowing the group to post general comments, concerns, or questions at any time.

**Phase II**

The goal of the second phase was to answer the remaining research questions, which were investigating the extent to which the unit was implemented as designed, the students’ perceptions of the unit, the potential effectiveness of the unit, and to outline a set of lessons learned and to make a final set of changes based on the analysis. The first part of this phase involved implementing the unit with my students. A detailed explanation of the daily and nightly activities is discussed in Chapter 4.

I took field notes during class and wrote in a daily journal after class every day, using templates that I created (Appendices A and B). The field notes focused on student behaviors, discussions, questions, and any other relevant comments made by students. By relevant, I am referring to comments that might have been valuable in answering any of the research questions, such as, “this is fun,” “I’m bored,” etc. I was unable to record every observation since I was also the teacher and did not want to hinder the
authenticity of the in-class experience by ruining the class momentum to record notes. This is discussed later as a limitation of the study.

The daily journal (Appendix B) was used to record any issues that were experienced during the lesson or any modifications that were made. In addition, I recorded general and specific thoughts I had about the study and how the day went. I also recorded any additional interview questions that came to mind as a result of my observations. Once the unit was complete, I added the additional interview questions, meet with the focus group one final time to review the questions, and made the appropriate changes. Once the interview protocol was complete, I conducted individual interviews with a sample of students.

Of the eight participating students, one left early for Spring break and could not be interviewed and another student preferred not to be interviewed. This resulted in a sample of six students which included the highest and lowest performing students, half juniors and half seniors, half females and half males, and half students with experience in computer programming and half without experience. Of the three students with experience, two were in my iOS app development class in the previous school year and one studied Java on their own. The interviews were guided by the list of questions that were developed with the help of the focus group. The questions were listed on my computer and the interviews were recorded using QuickTime (and my iPhone as a backup) and later transcribed.

Once all of the interviews were transcribed, I gave them an opportunity to review the transcription to see if it was an accurate portrayal of their thoughts. They were given time to make any comments regarding the transcriptions and given the option to request
anything to be updated/deleted. Based on the results from the data analysis, final changes were made to the unit.

Data Sources

There was an abundance of data available throughout this study. These data sources included the synchronous and asynchronous focus group conversations, interviews with students, observations, and other field data. The field data consisted of conversations on Slack, student-generated exercises and their solutions, and student response data hosted on several websites. Below, I will go into more detail on these sources.

Focus Group / Content Reviewers

As mentioned earlier, the collaborative group of AP CSA teachers and former student served as both a focus group and content reviewers to help further develop the initial unit (RQ1). This group had constant access to Slack and were tasked with asking questions, sharing comments and concerns, and reporting mistakes, all of which could take place at any point during the first phase. In addition, the audio and video from the Zoom meetings held online were recorded so I could focus my attention on the conversation and not on taking notes. The driving questions for the focus group meetings can be found in Appendix C.

Interviews

After the unit was complete, six of the students were individually interviewed as an exploratory means of better understanding the students' levels of engagement, satisfaction, and more generally about their perceptions of the flipped classroom (RQ3). The audio from the interviews were recorded on my computer using QuickTime and later transcribed. The interview protocol can be found in Appendix D.
Field Notes and Daily Journal

Field notes from observations and notes taken in my journal were recorded daily on my computer. The field notes focused on student behaviors, discussions, questions, and any other relevant comments made by students. Every effort was made to clearly distinguish between descriptions and my own interpretations. The daily journal was used to track issues and modifications made to the lesson as well as thoughts I had after the class was over. Depending on the day, I was free either the period immediately following the class or the period after that, so I was always able to begin writing within 45 minutes of the end of the class every day. This data was used to help determine the extent to which the unit was implemented as designed (RQ2) as well as the effectiveness of the unit (RQ4), which ultimately helped determine any changes that were made to the unit (RQ5).

Student-Generated Exercises

The purpose of collecting the student-generated questions was to help identify both levels of engagement and levels of performance. Throughout the unit, students were asked to generate their own exercises and to post them to a Slack channel devoted to this task. These questions were collected and evaluated to find the extent to which students were engaging with the material. More challenging questions with multiple layers were associated with more engaged students. Students were also asked to solve one another’s questions and to post their solutions to the questions, which at times, opened a dialogue. This data was also collected to identify the extent to which they were engaging in the activity and to determine any changes that should be made (RQ5).
Socrative

Another form of data came through a platform called Socrative, a student response system. With Socrative, teachers can create custom, multiple-choice questions. Students can log in to the quiz using the Socrative app on their phones or computers to answer each question. A report of the students’ responses is available after the quiz is complete. This data was also useful in determining the effectiveness of the unit (RQ4) and identifying changes that should be made to the unit (RQ5).

CodingBat

CodingBat is a website with programming exercises for students that provides immediate feedback. The exercises are open-ended and compare the solutions with multiple test cases. When students are incorrect, they can see which test cases failed. See Figure 3-3 for an example of a student’s view when they check their solution.

Figure 3-3. A student’s view of one attempt on CodingBat.

Teachers also have access to information on every question their students attempt, such as the number of attempts on each question and the rough percentage of
failed test cases on each attempt. See Figure 3-4 for an example of the teacher’s view of one student’s attempt at three different questions.

Figure 3-4. A teacher’s view of one student’s attempt on three CodingBat questions.

This data was useful in determining any trends with respect to the number of attempts students took and their success rates. While this data was available, data from other sources were needed for explanatory reasons. This data was also useful in determining any preliminary the effectiveness of the unit (RQ4) as well as the changes that should be made (RQ5).

**EdPuzzle**

EdPuzzle is a website that allows teachers to upload videos and embed their own quiz questions, which can be multiple choice or open-ended and graded or ungraded. All student responses were collected and evaluated to better understand any initial misconceptions and their levels of comprehension (RQ4) which ultimately helped
in determining changes that should be made to the unit (RQ5). In addition to the students’ responses, the number of times each section was watched was also recorded and available. Figure 3-5 shows an example of a teacher’s view for one student who watched one video that contained seven quiz questions, four of which were graded and three ungraded. This student watched the first 10% of the video twice, the second 10% three times, and the third 10% twice.

Assignment:

**BINAR Y search with FLAM ENCO dance**

<table>
<thead>
<tr>
<th>Grade</th>
<th>Video watched</th>
<th>Correct responses</th>
<th>Time spent: 4min</th>
<th>Turned in: On time - Feb 20th - 9:55 AM</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7/100</td>
<td>100%</td>
<td>2/7 answered</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Video views per portion

2 3 2 1 1 1 1 1 1 1
00:17 00:35 00:52 01:10 01:28 01:45 02:03 02:21 02:38 02:56

Figure 3-5. A teacher’s view of one student’s viewing data for one video on EdPuzzle.

**Required Questions**

After watching each video, students were required to submit their top question(s) about the content to a dedicated Slack channel named #video-questions. All of these questions were available throughout the duration of the study and were useful in better understanding the effectiveness of the videos (RQ4) and like all of the data, helpful in determining any changes that should be made (RQ5). For example, students asking questions regarding clarification were associated with less effective videos whereas
questions that extended the ideas from the videos were associated with more effective videos.

**Slack**

As mentioned above, Slack is an online communication tool and messages are stored for free, up to a given storage limit, which was never passed in this study. See Figure 3-6 for a screenshot of Slack. The different channels can be seen on the left, the main conversation in the middle, and one particular thread can be seen on the right. On the far left, each button represents a different team. All channels/conversations in a given team are only available to members of that particular team. Two teams were created for this study: one for the focus group and one for the students. In addition to the required participation on Slack, one channel was devoted towards general conversations/questions that students were able to partake in at any point. Data from Slack were useful in determining the extent to which the students were engaging in the material as well as the effectiveness of the unit (RQ4), in turn, helping to identify changes that should be made to the unit (QR5).
Data Analysis

The core elements of qualitative data analysis include (1) coding the data, (2) combining the codes into broader categories/themes, and (3) displaying and making comparisons in graphs, tables, and/or charts (Creswell, 2013). The data came from many sources, including student interviews, CodingBat, EdPuzzle, Socrative, and Slack, which incorporated the required questions, student-generated exercises and solutions, and general conversations. In addition, data were available from my own field notes and daily journal.

The data analysis began immediately, and data from everything except the interviews were organized and coded throughout the duration of the unit, rather than waiting until the study was complete, as recommended by Saldaña (2009). The data
from the field notes and daily journal were analyzed via constant comparison to help determine the extent to which the unit was implemented as designed. The interviews were coded using open coding with descriptive codes with the aim of understanding students’ perceptions of the unit (RQ3). Using the codes, categories were created, which were then finally used to identify themes.

**RQ1: After Initially Designing a Flipped Unit of Instruction, What Recommendations Were Made by the Focus Group During the Development Phase?**

A total of four synchronous focus group meetings and two asynchronous conversations were held over the duration of the unit. The synchronous meetings, which were hosted on zoom, were recorded (audio and video). After each meeting, I reviewed the comments and suggestions that were made by the group. Rather than transcribing word for word, I simply recorded the comments and recommendations holistically, since the comments and suggestions tended to be rather direct. In other words, there wasn’t much for me to “figure out” or to find the meaning of in terms of what they were saying. The asynchronous conversations took place via e-mail, so the comments were already recorded. Answering this research question was straight-forward: after each meeting, I listed all of the recommendations that were made by the focus group and the corresponding changes that were made to the unit.

**RQ2: To What Extent was the Unit Implemented as Designed?**

The data collected during the unit was used to answer this question, which mainly consisted of the field notes and daily journal. Since a template with questions was used for both the field notes (Appendix A) and the daily journal (Appendix B), the data was already somewhat organized. This streamlined the process of searching for emerging themes, potential patterns, and connections between pieces of data, as
recommended by Bogdan and Biklen (2002). Using constant comparison, I analyzed the data three times, first by considering each day at a time (including all notes for the given day), then again by considering each question/prompt at a time (including all days for the given prompt), and then a third time by considering each activity at a time. To better explain, I started by reviewing all notes (responses to the template questions) for the first day and holistically summarized the notes for the day. I also searched for any themes or connections between data within that day. This was repeated for every day to search for patterns. I reviewed the data a second time, this time looking at only one prompt at a time. For example, one question in the daily journal asked: “What would I do differently next time?” I looked at my response to this question for each day, again looking for themes. I then repeated this process for each question, holistically summarizing any findings for each question. The third time, I used conditional highlighting to highlight all occurrences of each activity, one at a time. For example, I highlighted all occurrences of “Socrative” and then reviewed every comment I made about that particular activity to attempt to identify any patterns.

In addition, before the unit started, I created a copy of the website that was shared with the focus group. As the unit progressed, the website was updated to reflect what actually happened in class and was actually assigned for homework. This resulted in two websites, one that contained the original unit that was designed and one that contained the actual unit which was carried out, which was another source to help remind me of the changes that were made.

On top of the field notes and daily journal, data from all of the other sources were considered for triangulation purposes. For example, in my field notes, I made a
comment about one group that answered a Socrative question very quickly and spent the remaining time talking about another class, which suggested that they didn’t put in a lot of effort into the question. Perhaps they simply found the question to be very easy. To get a better understanding, I checked the data from Socrative later on and found out that they both answered the question wrong. In addition, the question itself was a substantially larger question that would have required much more time than they actually put in, further suggesting that they simply picked an answer without putting in a lot of thought.

**RQ3: What Were Students’ Perceptions of the Unit?**

The data from the transcribed student interviews were used to answer this question. The transcriptions were coded openly with descriptive codes and later placed into categories. The categories were then used to identify general themes. While the focus of this question was on student perceptions, these themes were still used to help answer the other research questions by comparing the themes with findings from other sources of data for triangulation purposes. For example, if an identified theme was that students found the videos to be confusing, then I reviewed the data from EdPuzzle to see how often the videos were re-watched, and the extent to which the students’ solutions to the exercises were correct. High levels of re-watching videos and low-performance levels on the embedded exercises, in this case, would have provided more evidence that the videos were truly confusing to students.

**RQ4: Was There Any Preliminary Evidence to Support the Effectiveness of the Unit?**

The three sources used to answer this question consisted of Socrative, EdPuzzle, and CodingBat, since these platforms offered quantitative data. Simple
descriptive statistics were applied to learn about students’ success rates. The quantitative data were supported with qualitative data in an effort to help explain any phenomena that surfaced. For example, when trying to understand why success rates with some Socrative exercises were low, my field notes and the interview transcriptions offered some insight to paint a more complete picture.

**RQ5: What Were the Lessons Learned and Resulting Final Changes Made to the Unit?**

As I analyzed the data and answered the first four research questions, a list of findings was recorded in a spreadsheet. These findings emerged at different times and from different processes. For example, I recorded anything that was new to me as I analyzed the data. For instance, I assumed that all of the students already knew one another but later learned that this was not the case, so I recorded this information. While that was an example of a smaller finding, other, more comprehensive findings were recorded as well. Many of these findings stemmed from the coding process and as I was physically answering the research questions and pulling multiple data types together. These findings were combined to outline a set of lessons learned and to determine what changes should be made to the unit.

**Ethical Considerations**

The main ethical consideration in this study was my role as both the teacher and researcher, which raised two potential dilemmas: researcher-bias and coercing my students to participate. The first issue was addressed using two techniques: establishing procedures to segregate data and subjective inferences, and by triangulating data, which consisted of different types and was from different sources, as recommended by Richey and Klein (2014). I attempted to avoid coercing my students to
participate by reducing the social pressure to join, giving students and their parents time to meet with me to ask questions, and informing students that they may drop out at any point. I attempted to reduce social pressure by informing the students that I would not be discussing who joined and who did not, and by allowing students to turn in the consent forms at a time convenient to them, rather than asking for the forms to be turned in at the same time in class, where students would have had the ability to see who was and was not participating.

In addition, as I see it, as a teacher, I spent time working with other professionals to develop the highest quality instructional activities and materials within my capability and the capabilities of the participants in the focus group. Research aside, this was done with good intentions by always focusing on what was believed to be best for the students. With that in mind, all students were required to participate in the activities because at the end of the day, they were students in my classroom, and my job was to educate them to the best of my ability. Working with other professionals was a step above the best of my ability, considering I included their suggestions in the development of the activities. As a researcher, students were not required to participate in the interviews, nor was any of their data used without their permission. Students that did not agree to participate were not punished in any way. Likewise, students that participated were not rewarded.

**Validity**

Validation strategies are considered to be a strength of qualitative studies and can be thought of as “an attempt to assess the accuracy of the findings” (Creswell, 2013, p. 206). This study employed six validation strategies, including prolonged
engagement, triangulation, focus group review, clarifying researcher bias, member checking, and rich description.

- Prolonged engagement – while the study took place over only 19 days, I have taught all of these students in at least one class prior to this year. In addition, the school’s small student to faculty ratio of 13:1 allows the faculty to build close relationships and trust with the students. Also, six of the eight students were also enrolled in my AP Calculus AB or BC class at the same time this study took place. To paint a stronger picture, between the eight students, they have collectively been enrolled in 25 of my classes.

- Triangulation – Triangulation was used by making use multiple forms of data to provide corroborating evidence of findings, as discussed above in the data analysis section. Data came from a variety of sources in this study, including field notes, a daily journal, Slack conversations, follow-up video questions, student-generated questions and answers, EdPuzzle, CodingBat, Socrative, and interview transcriptions.

- Focus group review – The members of the focus group not only served to help develop the unit but also served as content reviewers by providing me with external checks regarding the instructional unit. The participants in this group were able to inform me of any mistakes they identified, or leave any general comments or questions at any point during the first phase of the study.

- Clarifying researcher bias – I attempted to clarify my bias as a researcher at the beginning of the study in Chapter 1 in the subjectivity statement. Since my aim was to develop the best possible instructional unit and since the final phase of the study involved making revisions based on the results of the data analysis, I was not out to “prove” anything or to show that this was the best unit possible. Instead, I was looking to see if there was any preliminary evidence of the unit’s effectiveness. I was also interested to see what lessons would be learned as a result of this study.

- Member checking – Member checking was performed with each student after the interviews were analyzed to check the credibility of the transcriptions. Each student was sent the transcription of their individual interview and given time to review it and to give me feedback regarding the extent to which I accurately portrayed their responses. The iterative nature of the focus group meetings was also a form of member checking considering changes that were made were brought back to the group to see if I correctly/fully interpreted their suggestions.

- Rich description – To increase the transferability of my results, I made every effort to provide a rich, thick description by including as many details as possible, within reason.
Limitations

As mentioned earlier, I was both the teacher and researcher. While participation was completely voluntary, despite my best efforts, students might have felt obligated to participate out of fear of repercussions. In addition, as a teacher, my goal was to work with my students to the best of my ability. As a researcher, my goal was to offer the most natural environment possible as well as to take as many field notes as possible. Naturally, it is impossible to do both of these to the best of one's ability, so at times, my role as the teacher or researcher suffered, considering every moment spent taking notes were moments that I was unable to work with students and every moment I was working with students I was unable to take notes. Despite my best effort, there are likely many notes that I missed.

During the interviews, it is possible that students might have told me what they thought I wanted to hear, rather than the truth. Every effort was made to ensure the students that I was only after their utmost honesty, and that I was not out to “prove” anything. In addition, not all of my students that were interviewed were necessarily equally articulate and/or perceptive. With the interviews being held at the end of the study, it is also possible that students either could not remember specific details or even reported the wrong details by accident as a result of misremembering.

Finally, the duration of the study was short. Students were not only learning new material, but in a new environment as well. It is possible that the duration of the study was not long enough to allow students to become fully comfortable with the new environment.
Summary

The purpose of this chapter was to outline the research design, describe the setting and participants, and the general procedure. Ethical considerations, validity, and limitations were also discussed. This design and development research employed a two-phase mixed methods design. The first phase consisted of developing the instructional unit and interview questions. In the second phase, the unit was implemented with my students, I interviewed a sample of the students, and analyzed all of the data. Lastly, a set of lessons learned were outlined and final revisions were made to the instructional unit, which were based on the results of the data analysis.
CHAPTER 4
INTERVENTION

Overview

This unit spanned over 19 days and consisted of eight main topics. An overview of the general pacing can be found in Table 4-1, and the specific pacing of each topic can be found in Tables 4-2 through 4-8. All activities for every day and every night are also outlined. The first time an activity is introduced in this chapter, it is explained in detail and a justification for the activity is provided. Subsequent occurrences of the same activity are not paired with as much detail or justification unless the activity is being employed for a reason other than it was originally implemented. Generally speaking, within a topic, students engaged in activities that progressed from highly structured to less structured. The initial design of the unit was driven by the conceptual framework discussed in Chapter 2.

Videos

Each new topic began with a video that students were to watch before class started. The conceptual framework suggests that passive content should be placed in the videos (Sarawagi, 2014). As such, the videos focused on general ideas as well as specific rules/syntax for each new topic, without getting into complicated ideas/problems. This allowed students to learn at their own pace and provided them with an opportunity to review the videos at any point (Bergmann & Sams, 2012). The videos laid the foundation for the activities that occurred in class. The conceptual framework also recommends including embedded quizzes in the videos (Elmaleh & Shankararaman, 2017; Jonsson, 2015; Lacher et al., 2018), so they were included and students were required to complete them, as recommended by Dazo, Stepanek,
Fulkerson, & Dorn (2016). Many of the quiz questions also provided immediate feedback, as per the conceptual framework (Garcia, 2018). In addition, the conceptual framework states that the videos should be from the same source (Baldwin, 2015; Knutas et al., 2016), so I created most of the videos myself. Some of the videos contained animations, but since I do not have a strong background creating animations, I used animations found online, although they were from the same source. Another large component of the conceptual framework are the principles of CTML (Mayer, 2005), all of which were kept in mind as I created the videos. Along the same lines, to reduce the extraneous cognitive load, each video was placed in the same location (EdPuzzle), which had an easy-to-use interface with quiz questions built directly into the video.

As recommended by Morrison, Ross, and Kemp (2006), each video began with a form of motivation for learning the material. This was typically achieved by reminding students of a past experience and relating it to the current material. At the end of each video, students were required to post at least one of their top questions regarding the content of the video, as advised by the conceptual framework (Thomas, 2014). Since the conceptual framework also recommends providing an environment that encourages students to ask and answer questions outside of class (Shen et al., 2009), students were asked to post their questions to a dedicated Slack channel and were encouraged to attempt to answer their peers’ questions. Finally, since the conceptual framework suggests making students aware that the instructor is cognizant of their out-of-class progress (Köppe et al., 2016), students were informed that I would be checking their solutions to the embedded quiz questions as well as the conversations and questions.
on Slack, both of which were paired with a completion grade. Also, each class following a video began with a discussion of the students’ top questions, which also raised their awareness of my knowledge of their out-of-class participation.

**In-Class Activities**

The in-class activities were designed based on active learning techniques discussed in Chapter 2, which are included in the conceptual framework. On the days following a video assignment, I reviewed the students’ questions from Slack as well as their answers to the quiz questions to see if I could identify any common questions and misconceptions. Doing this allowed me to prepare an example (or several examples) that would help to better explain any concepts that students struggled with. It also gave me an opportunity to modify any of the in-class activities for the day, as recommended by the conceptual framework (Thomas, 2014). Each of these class periods that followed a video assignment began with a discussion of the students’ questions, which included any examples that I may have prepared. After the discussion, students engaged in the activities designed for the day.

Students were required to complete all of the activities that were assigned in class. If they were not able to finish them in class, they were required to finish them out of class and were assigned a completion grade as a form of motivation, another element of the conceptual framework (Largent, 2013).

**Lessons and Sequencing**

The AP CSA course description (2014) lists the following topics that students are to be familiar with:

- 1-dimensional arrays
  - Traverse using for loops
  - Traverse using for-each loop
• 2-dimensional arrays
  • Understand these are stored as arrays of arrays
  • Access a row and:
    ▪ assign it to a 1D array reference
    ▪ pass it as a parameter
    ▪ use loops to traverse the row
  • For the exam, students should assume that 2D arrays:
    ▪ are rectangular
    ▪ are in row-major order
• Students should know the following algorithms:
  • Sequential search
  • Binary sort
  • Selection search
  • Insertion short
• Understand ArrayIndexOutOfBoundsException
• Initializer list: { ... }

After reviewing this list and taking into consideration my own experience teaching
the class, I categorized each topic from this list into eight main topics, including the
basics of arrays, arrays of objects, copying arrays, objects with arrays, parallel arrays,
searching, sorting, and two-dimensional arrays. Naturally, the basics of arrays had to be
discussed before anything else. Within this topic, only primitive data and Strings were
incorporated (Note: data types and code will be expressed in this
font for readability purposes). Strings were included in the basics because
they can often be thought of as primitive data since they have special handling. In
addition, since Strings are actually objects, they set the stage for the next topic, which
covered arrays containing non-primitive data. Following, students learned about
differences between making copies of arrays and setting arrays equal. This needed to
happen before students learned about creating objects with arrays in order to avoid
making common mistakes when writing constructors. In my experience, when students
write constructors that accept arrays as parameters, they tend to set the instance
variable directly equal to the input, meaning, changes made to the object’s array will
result in changes made to the original array that was passed in. In other words, they link the arrays together since the value was passed by reference. This may not be problematic depending on the intention of the student, but they need to be aware of this when writing their code. After the students learned how to create classes with arrays as instance variables, they learned about parallel arrays. This could have been taught immediately after the basics of arrays, but since it is a relatively straightforward concept, I viewed this as an opportunity to use parallel arrays with some of the more complex ideas that students would have already learned about. By this point, students were ideally comfortable working with arrays and fully understood the syntax so they were able to focus their efforts in other areas, such as understanding common algorithms, including searching and sorting algorithms. Since the sorting algorithms depend on concepts learned from searching algorithms, the searching algorithms were taught first.

Finally, two-dimensional (2D) arrays were taught last due to their higher level of complexity, based on my own experience working with my students. See Table 4-1 for an overview of the sequencing and pacing of lessons.

Table 4-1. Overview of pacing.

<table>
<thead>
<tr>
<th>Day(s)</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>Basics of arrays</td>
</tr>
<tr>
<td>6</td>
<td>Arrays of objects</td>
</tr>
<tr>
<td>7</td>
<td>Copying arrays</td>
</tr>
<tr>
<td>8-9</td>
<td>Objects with arrays</td>
</tr>
<tr>
<td>10</td>
<td>Parallel arrays</td>
</tr>
<tr>
<td>11-12</td>
<td>Searching</td>
</tr>
<tr>
<td>13-14</td>
<td>Sorting</td>
</tr>
<tr>
<td>15-17</td>
<td>Two-dimensional arrays</td>
</tr>
<tr>
<td>18</td>
<td>Review</td>
</tr>
<tr>
<td>19</td>
<td>Test</td>
</tr>
</tbody>
</table>
Basics of Arrays

Since understanding the basics of arrays is compulsory for understanding more advanced topics, more time was spent on the basics than any other topic. The aim was to reduce the cognitive load associated with learning the more advanced topics by giving students enough time to master the basics. The duration of this topic was determined based on my own past teaching experience and anticipated time the activities would take. Also, to keep things simple, this topic only included arrays with three types of primitive data, including integers, doubles, and booleans, as well as String objects, all of which students are expected to be familiar with on the AP CSA exam and had been working with for over a semester. As mentioned earlier, the String class was included because String objects tend to behave similarly to primitive data, at least for the sake of the concepts covered in this particular lesson.

Initial night: As a form of motivation, the first video discussed dilemmas the students faced earlier in the year. For example, the students wrote a class that represented a playing card and were tasked with instantiating twenty unique randomly generated playing cards. This proved to be a very laborious task that would have been much easier with the use of arrays. The video covered subtopics including indexing, initializing, and traversing through arrays using for, for-each, and while loops. Students were required to post at least one question related to the video in the Slack channel, #video-questions.

Day 1. The class began with a discussion of students’ questions, as recommended by the conceptual framework (Thomas, 2014). Once we reached a point where I felt that the majority of students were relatively comfortable with the basics of
arrays, they were randomly broken up into groups of two, and one group of three, using a random group generator. Groups were randomly generated throughout this unit to encourage students with different backgrounds to work with one another since students have their unique strengths and weaknesses. Another component of the conceptual framework is the use of personal response systems (Freeman et al., 2014; Largent, 2013). As such, we used Socrative, where I displayed multiple choice and true/false questions about the concepts discussed in the video. Students first answered individually using their computers or phones, then spoke to their group and were given a chance to change their answer, which is a technique that has been found to increase student performance and satisfaction (Simon et al., 2010). After each question, students had an opportunity to debate their answers, as advised by the conceptual framework (Bonwell & Eison, 1991). The correct answer was finally revealed and we discussed any questions students had. Getting everyone set up with Socrative for the first time took about five to ten minutes. We then spent about 30 minutes on Socrative. Each student was later given an algorithm to individually trace, as seen in the conceptual framework (McConnell, 1996), and asked to consider what the purpose of the algorithm was. After, students were asked to share their understanding of the algorithm with their group. Finally, we discussed the algorithm as a class. The purpose behind this was to give students an opportunity to see code written by an expert so that they would form proper coding habits, which is another component of the conceptual framework (Tritrakan et al., 2016). The initial plan was to include two more algorithms and four programming exercises, but unfortunately, we ran out of time.
**Night 1:** Since we ran out of time, students were assigned the two algorithms to trace as homework. In addition, students were given two programming exercises and asked to write pseudocode only, a form of comment first coding, as suggested by the conceptual framework (Cakiroglu, 2013). These were required to be turned in before class the next day.

**Day 2:** I started class by leading a ten-minute discussion where we reviewed the algorithms that students were asked to trace for homework. In addition, before class started, I purposefully selected a sample of student solutions which were then critiqued by the class, as proposed in the conceptual framework (Jonsson, 2015). Since the conceptual framework also promotes students seeing multiple solutions to the same problem and discussing their strengths and weaknesses (Köppe et al., 2015), I identified one incorrect solution and two correct solutions with different approaches for each problem. Although I was aware of whose solutions were being criticized, the students’ names remained anonymous to the rest of the class. After spending about ten minutes critiquing students’ solutions and since we ran out of time the previous day, the remaining time was spent finishing the Socrative questions. Just like the previous day, students were first required to answer each question individually and then asked to consult their group member(s) before answering a second time.

**Night 2:** Tonight, students were given two programming exercises to solve. This time, students were not only required to write the comments for both, but they were required to then write the code corresponding to their comments. Again, they were required to turn these in before class the next day.
Day 3: By this point, we were an entire day behind, so I added an extra day (today) to accommodate for the activities that had not been completed. The first ten minutes of class was spent critiquing students’ solutions that I hand-picked before class. I led the discussion and this was done as a class. Groups were then generated and we spent another ten minutes on Socrative, using the same format as the previous two days. The remaining 25 minutes was spent working on a set of programming exercises using a pair-programming approach, another component of the conceptual framework, since it has the potential for improving student achievement (Umapathy & Ritzhaupt, 2017). With this approach, students took on different roles, one as the driver who controlled the mouse and the keyboard, and the other, who acted as the navigator and assisted the driver. Students were required to take turns in different roles to keep them engaged (Williams et al., 2008). Since one of the groups contained three students, they implemented a modified pair-programming approach, where two students acted as the navigators who both assisted the driver. There was a total of four required exercises and two optional exercises. I spent my time circulating the room and offering the groups support, as recommended by the conceptual framework (Köppe et al., 2016). This was typically done in the form of answering their questions with leading questions to get them thinking in the right direction.

Night 3: Tonight, students had two more programming exercises to complete as well as any exercises that they were unable to finish from class. In an attempt to start removing the added support/structure, students were required to use the CFC approach for the first exercise but not the second exercise.
**Day 4:** Like the previous day, the first ten minutes was spent critiquing a variety of students’ solutions to the previous night’s assignment. Students were then randomly placed into groups and worked on a set of programming exercises using the pair-programming approach. There were four required exercises and two optional exercises. I again spent this time moving around the room and offering support to the groups.

**Night 4:** Since immediate feedback on students’ solutions has been recommended by many researchers and is a part of the conceptual framework (Chen, Chen, Chang, & Yang, 2017; Garcia, 2018; Pérez-Marín & Pascual-Nieto, 2012), students were assigned ten CodingBat problems for homework. While ten might sound like a lot, these were basic questions that did not require the use of any loops. Each question was intended to be solved in less than two minutes or so. By this point, students had a lot of practice with the basics of arrays so they were prepared to generate their own programming exercise. This was a requirement since student-generated content has been found to have a positive impact on learning outcomes (Carlston, D. L., 2011; Humphries & Borg, 1998; Ramirez-Velarde, Perez-Cazares, Alexandrov, & Garcia-Rueda, 2014). Students were not tasked creating their own materials until this point because it is possible that this activity could have had negative effects on learning outcomes if they did not have enough prior knowledge (Reeves-Kazelskis & Kazelskis, 1987). Students posted their exercise in the #student-exercises Slack channel. They were also required to write a solution, which was not to be posted on Slack, although they were required to turn it into me using the school’s learning management system (LMS). They were asked to turn it into me ensure they did not
change their answers after seeing their peers’ solutions to their exercises as well as to push them to fully think through their exercise.

**Day 5:** Every student was assigned another student’s exercise that was created the night before and asked to post their solution as a reply to their exercise in Slack. Each student was then assigned to a second student and asked to critique their proposed solution with a reply in the same thread. To clarify, suppose student A generated a question and posted it in Slack. Student B solved that exercise and posted their solution as a reply to student A. Student C then reviewed student B’s solution to student A’s question and offered their criticism as the third comment in the thread. Students were given the remaining class time to individually work on ten more CodingBat exercises. Although students were solving the questions individually, they were encouraged to seek help from their peers and me if/when they needed help. I spent this time walking around the room offering individual support.

Taking a step back to look at the sequencing of activities in this section, the progression was such that students were given a lot of support and structure at the beginning, which was slowly removed over time. As a recap: students were given worked examples (in the video), answered basic questions in the video and in Socrative and given immediate feedback on each answer, traced code written by experts, and wrote programs using comments only. With all of this under their belts, students finally wrote their first program using the CFC technique. Students saw those solutions criticized in front of the entire class, and then wrote more programs using a pair-programming technique. Students then wrote their own programs from scratch, and I was able to give individual attention as needed during that time. Students later created
their own exercise, solved a peer’s exercise, checked another peer’s solution, and finally provided feedback (tonight’s homework) to the student that solved their own exercise. An overview of the sequencing of activities can be found in Table 4-2.

### Table 4-2. Basics of arrays.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Watch video</td>
<td>1</td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Socrative</td>
</tr>
<tr>
<td>1</td>
<td>Trace algorithms (x2)</td>
<td>2</td>
<td>Review algorithms</td>
</tr>
<tr>
<td></td>
<td>Programming exercises –</td>
<td></td>
<td>Critique students’ solutions</td>
</tr>
<tr>
<td></td>
<td>pseudocode only (x2)</td>
<td></td>
<td>Socrative</td>
</tr>
<tr>
<td>2</td>
<td>Programming exercises (x2)</td>
<td>3</td>
<td>Critique students’ solutions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Socrative</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pair-programming exercises (x6)</td>
</tr>
<tr>
<td>3</td>
<td>Finish unanswered exercises</td>
<td>4</td>
<td>Critique student solutions</td>
</tr>
<tr>
<td></td>
<td>Programming exercises (x2)</td>
<td></td>
<td>Pair-programming exercises (x6)</td>
</tr>
<tr>
<td>4</td>
<td>Finish unanswered exercises</td>
<td>5</td>
<td>Solve a peer’s exercise</td>
</tr>
<tr>
<td></td>
<td>CodingBat (x10)</td>
<td></td>
<td>Critique a peer’s solution</td>
</tr>
<tr>
<td></td>
<td>Create an exercise</td>
<td></td>
<td>CodingBat (x10)</td>
</tr>
</tbody>
</table>

**Arrays of Objects**

**Night 5:** Students were asked to look at the comments made to the exercise they created the night before. Then, they were to post their original solution by replying to their classmate’s proposed solution and summarizing any differences between their classmate’s solution and their own solution, and commenting on both solutions in terms of correctness, efficiency, and approach. The purpose of this activity was to provide another opportunity for students to see their own work criticized, as advised in the conceptual framework (Jonassen, 2014). This activity afforded each student two opportunities to see their work criticized. First, after solving an exercise, it was reviewed by a peer, who provided feedback. Second, the original student that created the exercise provided feedback to their solution in Slack, the following night. In other words, going back to the previous example, student B received feedback from student C and
then by student A. The students were also asked to finish any unanswered exercises from class earlier that day as well as to have a total of 30 CodingBat exercises completed by the next day. Of the 30 questions, 20 were to be from the Array-1 group and 10 were to be from the Array-2 group, which typically involve traversing through an array.

Up to this point, students were only working with arrays of integers, doubles, booleans, and Strings. Tonight, they watched a video covering the topic of arrays of any data type. The video began with a reminder that a String is an object, and that they, therefore, had been working with arrays of objects the entire time. They were required to post at least one question regarding the content covered in the video.

**Day 6:** The first five minutes was spent discussing students’ questions from the video. In addition, before class started, I took all of the students’ responses to the embedded video-questions and put them in a spreadsheet with students arranged by row and responses by column (removing students’ identities). I projected the solutions to the front of the room and we went through, question by question, searching for mistakes and discussing them. This activity took about 15 minutes and the remaining time was spent on Socrative. This time, after realizing how time consuming it was for students to solve each question twice (once individually and again after consulting their group), I had students answer the questions only once and without consulting a group. The students were going to be given the rest of the period to individually work on programming exercises, but we ran out of time before we could get started. Table 4-3 provides an overview of this section.
Table 4-3. Arrays of objects.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CodingBat (x10 - a total of 30 by this point)</td>
<td>6</td>
<td>Discuss top questions Critique students’ video solutions Socrative</td>
</tr>
<tr>
<td></td>
<td>Comment on peer’s solution Watch video Post top question(s)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Copying Arrays

**Night 6:** Analogies are another component of the conceptual framework (Cakiroglu, 2013). As such, an analogy was used by relating an old concept with the new concept in the video. This was illustrated by using a basic object and reminding students of the differences between passing by value and passing by reference. The video then provided examples to demonstrate the same concept, but with arrays. The purpose of the video was to teach the students how to *link* arrays and how to *copy* arrays and to understand the differences between the two. Students were required to post at least one question to Slack. Additionally, they were required to complete the programming exercises that we did not have enough time to get to earlier that day.

**Day 7:** The first day of a new unit was typically very similar throughout this unit. On this day, students’ questions were discussed and then they individually answered questions from Socrative. The students spent the remaining time working in randomly generated groups solving programming exercises using the pair-programming approach as I walked around the room and assisted the groups as needed. See Table 4-4 for an overview of this topic.

Table 4-4. Copying arrays.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Programming exercises (x2) Watch video Post top question(s)</td>
<td>7</td>
<td>Discuss top questions Socrative Pair-programming exercises (x7)</td>
</tr>
</tbody>
</table>
Objects with Arrays

Night 7: The video began by having students recall a Student class that was created earlier in the year. This object contained one String to represent the name, and three integers to represent test scores. Students were asked to consider how this could have been written differently if the Student object needed to hold 100 test scores. The video covered several examples of objects that contained arrays. As per the norm, students posted their top question(s) about the video and finished any exercises from class that were incomplete.

Day 8: The class began with a discussion of students’ questions and then they were broken up into random groups to solve to Socrative exercises. Since there were only three Socrative questions, we reverted to the original format where students answered first individually and then again after consulting their group. Following that, students were assigned a set of programming exercises and were given the choice to either work individually or using the pair-programming approach.

Night 8: In addition to finishing incomplete exercises from class, students were assigned one programming exercise which was going to be critiqued the next day. One of the purposes of this exercise was to get students to start thinking about ArrayLists. They were essentially tasked with creating their own version of the ArrayList class, which can be thought of as a data structure with an array as an instance variable. The homework referred to this data type as a SuperArray.

Day 9: The original plan was to critique students’ solutions as a class but I went in a different direction on this day. The rationale for this decision will be discussed in Chapter 5. Instead, I broke the students up into random groups and assigned each
group two solutions to critique and to provide feedback to on Slack. They were going to be given the remaining time to individually work on the programming exercises, but there wasn’t any time leftover. See Table 4-5 for an overview of these activities.

Table 4-5. Objects with arrays.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Watch video</td>
<td>8</td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Socrative</td>
</tr>
<tr>
<td>8</td>
<td>Finish unanswered exercises</td>
<td>9</td>
<td>Pair-programming exercises (x3)</td>
</tr>
<tr>
<td></td>
<td>Programming exercise (x1)</td>
<td></td>
<td>Critique students’ solutions</td>
</tr>
</tbody>
</table>

Parallel Arrays

Night 9: Students were asked to consider how one might track student names (Strings) with their corresponding grades (integers) for a single test without creating a new data structure. The video went on to give other examples of how parallel arrays could be used along with demonstrations of how they were used. Students posted their top questions to Slack.

Day 10: We spent the first few minutes discussing students' questions from the video. In addition, the final question from the video simply asked students what parallel arrays were. I took all of their definitions, removing the students’ identities, and put them in Slack as a poll. I asked the students to vote for the definition that they thought was the best. The purpose of this was to expose the students to some of the finer details they might not have thought of. Afterwards, they were broken up into groups for pair-programming. Table 4-6 provides an overview of these activities.

Table 4-6. Parallel arrays.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>CodingBat (x10 – a total of 40 by this point)</td>
<td>10</td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Watch video</td>
<td></td>
<td>Vote for best definition</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Pair-programming exercises (x4)</td>
</tr>
</tbody>
</table>
Searching

**Night 10:** Students watched a video with flamenco dancers acting as the linear search algorithm. Several women were lined up with numbers on their backs. One woman at a time stepped out of the line to dance with a man, who had a number on his back to see if their numbers matched. If they didn’t match, the woman went back in line and the next woman stepped up to dance. No actual code was revealed in the video. Students were given the freedom to fast-forward through the video since it was a rather straight-forward concept and since video was long (it took six minutes to check six numbers). Students were then tasked with writing the linear search algorithm on their own using the CFC technique. They were also asked to post at least one question and to complete any unanswered exercises from class.

**Day 11:** The class started with a discussion of students’ questions and we then moved on to critiquing students’ attempts on the linear search algorithm. Students were given two algorithms to individually trace and then two programming exercises to individually solve. These were done individually, again because the linear search algorithm is straight-forward concept.

**Night 11:** In addition to completing any unanswered exercises from class, students were given another video to watch. The same group of flamenco dancers put together a video on the binary search algorithm. Again, no code was revealed in the video. The students were tasked with writing the binary search algorithm using CFC and as usual, posting at least one question.

**Day 12:** After students’ questions were discussed, rather than critiquing their solutions, we wrote the algorithm as a class. I projected my computer screen to the front of the class and I asked students to share their ideas. The original plan was to critique
students’ solutions like we did on the previous day, but after seeing the levels of confusion from the students (based on their solutions and questions), I thought it would be best to write the algorithm from scratch with the class. After we finished writing the algorithm, I led a brief discussion on the differences between the linear and binary search algorithms. Since the students found this algorithm to be more challenging than I expected, I had the students solve the programming exercises via pair-programming instead of individually. Table 4-7 contains an overview of the activities for this topic.

Table 4-7. Searching.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Finish unanswered exercises</td>
<td>11</td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Watch linear search video</td>
<td></td>
<td>Critique student’ solutions</td>
</tr>
<tr>
<td></td>
<td>Write linear search algorithm</td>
<td></td>
<td>Algorithm tracing (x2)</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Individual programming exercises (x2)</td>
</tr>
<tr>
<td>11</td>
<td>Finish unanswered exercises</td>
<td>12</td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Watch binary search video</td>
<td></td>
<td>Write binary search algorithm as a class</td>
</tr>
<tr>
<td></td>
<td>Write binary search algorithm</td>
<td></td>
<td>Compare search algorithms</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Pair-programming exercises (x2)</td>
</tr>
</tbody>
</table>

Sorting

**Night 12:** After completing any exercises that were unfinished from class, the students watched a very short animation, one element of the conceptual framework (Tucker et al., 2006), of the selection sort algorithm. Since the sort algorithms tend to be challenging to students based on my experience, they were asked to write the pseudocode only. An optional video was provided, this time with folk dancers acting as the selection sort algorithm. Additionally, they posted at least one question to Slack.

**Day 13:** Before discussing any of the students’ questions, the students engaged in the role-playing activity, specifically, method-acting. Everyone was given either a number or a method that was written on a piece of paper. The methods included
findMinIndex, swap, and sort. Students with numbers were placed in a random order and the three students acting as the methods were given the respective directions. After the students were ordered, they sat back down and we discussed any remaining questions. We then wrote the selection sort algorithm as a class, similar to the previous day. Following that, students were broken up into random groups for pair-programming exercises.

**Night 13:** Just like the previous night, students completed any remaining exercises from class, watched a short animation video and an optional dance video, wrote pseudocode for the insertion sort algorithm, and posted at least one question to Slack. Additionally, each student was assigned another students' pseudocode for the selection sort algorithm that was written the previous night and tasked with writing the algorithm based on the comments. If the logic behind the pseudocode was incorrect, they were to write what was wrong with it and what to change so it would be correct.

**Day 14:** This day was very similar to the previous day. All of the activities were repeated but with the insertion sort algorithm this time. The only difference was the inclusion of a class-wide discussion on the differences between the two sort methods. For an overview of these activities, see Table 4-8.
Table 4-8. Sorting.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Finish unanswered exercises</td>
<td>13</td>
<td>Role-play (method-acting)</td>
</tr>
<tr>
<td></td>
<td>Watch selection sort video</td>
<td></td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Write selection sort algorithm pseudocode</td>
<td></td>
<td>Write selection sort algorithm as a class</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Pair-programming exercises (x2)</td>
</tr>
<tr>
<td>13</td>
<td>Finish unanswered exercises</td>
<td>14</td>
<td>Role-play (method-acting)</td>
</tr>
<tr>
<td></td>
<td>Write selection sort algorithm based on peer’s pseudocode</td>
<td></td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Watch insertion sort video</td>
<td></td>
<td>Write selection sort algorithm as a class</td>
</tr>
<tr>
<td></td>
<td>Write insertion sort algorithm pseudocode</td>
<td></td>
<td>Compare sort algorithms</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Pair-programming exercises (x3)</td>
</tr>
</tbody>
</table>

2-Dimensional Arrays

Night 14: Again, each student was assigned another student’s pseudocode for the insertion sort algorithm and tasked with writing the code or correcting the pseudocode. Additionally, they were to complete any unfinished exercises from class as well as to watch a video that covered the topic of 2D arrays. Students posted at least one question they had from the video.

Day 15: As per the norm, the students’ questions from the previous night’s video were discussed as a class. Students were then broken up into groups to solve Socrative questions. This time, students only answered each question once and only after consulting their group. They were then given a set of exercises to complete via pair-programming instead of individually since 2D arrays tend to be challenging based on my teaching experience.

Night 15: Students finished any unanswered exercises from class, traced two algorithms, and completed one programming exercise.

Day 16: Since the previous night’s programming exercises was challenging, the entire class period was spent critiquing the students’ solutions. The original plan was to
spend some of the period working on an AP free response question (FRQ), but there was no time so that requirement was dropped.

**Night 16:** Students took their first attempt at an old AP FRQ. They solved the FRQ for two reasons: to practice for the AP exam and to include another component of the conceptual framework, which are templates (partially written code for students to expand on) (Cakiroglu, 2013). The FRQ that was assigned on this night already contained the method headers and the students were asked to write the bodies only. Additionally, students created their second exercise of the unit, this time with 2D arrays.

**Day 17:** The first half of the class was spent critiquing students’ solutions to the AP FRQ while the remaining time was spent solving another AP FRQ using pair-programming. As always, I led the class discussion during the critiques and assisted groups as needed during the pair-programming activity. An overview of the activities in this topic can be found in Table 4-9.

**Table 4-9. Two-dimensional arrays.**

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Finish unanswered exercises</td>
<td>15</td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Write insertion sort algorithm</td>
<td></td>
<td>Socratic</td>
</tr>
<tr>
<td></td>
<td>based on peer’s pseudocode</td>
<td></td>
<td>Pair-programming exercises (x4)</td>
</tr>
<tr>
<td></td>
<td>Watch video</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Finish unanswered exercises</td>
<td>16</td>
<td>Critique students’ solutions</td>
</tr>
<tr>
<td></td>
<td>Algorithm tracing (x2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Programming exercise (x1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>AP FRQ (x1)</td>
<td>17</td>
<td>Critique students’ solutions</td>
</tr>
<tr>
<td></td>
<td>Create an exercise</td>
<td></td>
<td>Pair-programming AP FRQ (x1)</td>
</tr>
</tbody>
</table>

**Review and Test**

**Night 17:** Students were asked to complete the AP FRQ if it was not finished in class. In addition, they were assigned one of their peer’s exercises to solve and to
respond to on Slack. Lastly, they were required to post at least one remaining question they had regarding anything related to the topic of arrays.

**Day 18:** The class started with a discussion of students’ questions from Slack. An overview of the test format was then provided and the rest of the class period was spent reviewing via practice problems.

**Night 18:** Students spent this time studying for the test.

**Day 19:** Students took the test.

An overview can be seen in Table 4-10.

<table>
<thead>
<tr>
<th>Night</th>
<th>Out of Class</th>
<th>Day</th>
<th>In Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Finish AP FRQ</td>
<td>18</td>
<td>Discuss top questions</td>
</tr>
<tr>
<td></td>
<td>Solve peer’s exercise</td>
<td></td>
<td>Overview of test format</td>
</tr>
<tr>
<td></td>
<td>Post top question(s)</td>
<td></td>
<td>Practice exercises</td>
</tr>
<tr>
<td>18</td>
<td>Study</td>
<td>19</td>
<td>Test</td>
</tr>
</tbody>
</table>

**Summary**

This chapter outlined the activities that were done both in class and out of class. All of the activities were designed based on recommendations made by previous researchers and were further developed with a focus group. In addition, the videos were designed based on principles of CTML and the activities students engaged in during class were designed based on active learning techniques, all of which were summarized in the conceptual framework in Chapter 2. Eight topics were covered, including the basics of arrays, arrays of objects, copying arrays, objects with arrays, parallel arrays, searching, sorting, and two-dimensional arrays. These topics were clustered based on my own experience teaching AP CSA and on the topics listed by the College Board (2016). Generally speaking, each unit progressed from highly-structured activities to less-structured activities.
Overview

In this chapter, I will discuss the results by answering each research question, one at a time. As a reminder, the research questions were as follows: (1) After initially designing a flipped unit of instruction, what recommendations were made by the focus group during the development phase? (2) To what extent was the unit implemented as designed? (3) What were students’ perceptions of the unit? (4) Was there any preliminary evidence to support the effectiveness of the unit? (5) What were the lessons learned and resulting final changes made to the unit?

**RQ1: After Initially Designing a Flipped Unit of Instruction, What Recommendations Were Made by the Focus Group During the Development Phase?**

A total of four focus group meetings took place online synchronously over a three-week period. Due to scheduling complications, the last two synchronous meetings were canceled and took place asynchronously via e-mail, instead. Each meeting had a specific focus in terms of the topics being discussed. The topics were as follows: (1) basics of arrays, (2) arrays of objects and copying arrays, (3) objects with arrays and parallel arrays, (4) searching and sorting, (5) two-dimensional arrays, and (6) student interview questions.

Time constraints forced the unit to begin before all of the meetings took place, therefore, there was some overlap between the meetings and the implementation of the unit. The unit started between the second and third meetings. None of the topics were covered before the respective focus group meeting took place and I had a chance to make revisions and to share those revisions with the focus group.
Finding common times for all four participants turned out to be a daunting task, and as a result, there was unfortunately only one meeting where all four participants were in attendance. When participants were not able to attend, they would attempt to e-mail me their thoughts and suggestions.

**Meeting #1: Initial Night – Day 5 (Basics of Arrays)**

Three of the four participants were in attendance. The group agreed that the quantity of content seemed to be too much for the given time span. As it turned out, they were correct because I needed to add an extra day in order to complete all of the activities in this section. More on that will be discussed later.

The group identified a few exercises that had flaws and/or could have been written more clearly. For example, one exercise involving traversing through an array had several correct answers based on the wording, so it was re-written to include the requirement “without going out of bounds,” which limited it to one correct answer.

One change was made based on a suggestion to change the CodingBat requirement from completing ten specific exercises to allowing students to select any ten exercises from the given section. The group also noticed how similar some of the exercises were and suggested that I make a couple of these questions optional, so I made this change. The group suggested that I tell the students that these optional questions might show up on the test, as well as some CodingBat questions, which I did end up doing.

An interesting conversation also took place over the requirement for students to post a question after watching the video. There was a general concern that students would post fake questions for the sake of fulfilling the requirement. Ultimately, we decided to keep the requirement, but students were given more freedom. In the event
that students did not have any questions specific to the video they watched, they were allowed to ask any question they had regarding the topic of arrays.

Meeting #2: Night 5 – Day 7 (Arrays of Objects and Copying Arrays)

All four participants were in attendance. The group agreed that the content covered in last few minutes of the first video (Basics of Arrays) should be moved to the second video (Arrays of Objects) because the focus of the conversation was on objects.

The group also agreed that the variable names should be more distinct. For example, consider the Double wrapper class. Instead of looping through an array named doubles and referring to each value as double, resulting in for (Double double : doubles), the group suggested something more along the lines of for (Double d : nums). I ultimately decided to leave it as was because my students were familiar with that style of variable naming, as they were using it all year long. A suggestion was to be more consistent with the vocabulary used in the videos and exercises. Specifically, rather than arbitrarily interchanging the terms function and method, the term, method, should always be used. I agreed and changed all written and verbal occurrences of function to method.

In addition to a few typos, the directions for some assignments were clarified as well. For example, students were asked to complete ten additional CodingBat exercises. The suggestion was to rephrase that by asking students to have a total of twenty exercises completed in the event that a student was ambitious and completed more than ten on the first day. One final suggestion and change that was made was to require students to first write the pseudocode for an exercise that asked students to fill an array with 20 randomly generated integers between 5 and 10.
Meeting #3: Night 7 – Day 10 (Objects with Arrays and Parallel Arrays)

Unfortunately, despite their best efforts, one of the members was no longer able to supply feedback due to their transitioning of roles from a high school teacher to a software developer. In addition, only one participant was able to attend this particular meeting. The majority of this meeting involved revisiting the video on copying arrays. The participant pointed out that the first nine minutes had nothing to do with copying arrays. Instead, it was setting up a class that was going to be used to demonstrate the concept of copying arrays. As a result, the video was broken up into two parts and the first video was made optional for students. There were also a couple of methods that appeared to be designed arbitrarily. It was suggested that I spend some time talking about how this was simply my decision as the programmer. In these instances, students were asked in the video to consider why a programmer might choose to design it one way or another and what other options for writing the method there might have been.

In addition, as usual, several typos were identified and corrected, as well as homework directions that were made more explicit. The final suggestion was to reconsider beginning the video on parallel arrays by stating that it was going to be an "easy topic," as that might have made students feel uncomfortable if they found the topic to be confusing, so this was removed from the video. A recap of the meeting with the suggested changes was sent out to the entire group, but no feedback was given from the absent members.

Meeting #4: Night 10 – Day 14 (Searching and Sorting)

Two of the three participants were in attendance and the third member e-mailed their comments. Regarding the role-playing activity, it was recommended that in addition to students acting as integers in the array, students act as methods as well. For
example, one student acted as the swap method and their job was to swap two students in the array. Another student acted as the minFinder, whose job was to identify the student with the smallest integer value. I originally planned for the students to act as the numbers only, so this advice was taken and implemented in the unit. The group also agreed that the role-playing activity would not be necessary for the search algorithms due to their simplicity. Instead, role-playing was only implemented with the sort algorithms. Another suggestion was to add optional videos of dancers acting out the various search and sort algorithms as a fun way for students to visualize the algorithms. I had never seen those videos before, and I included them based on their advice. The group also recognized that the lesson on binary search potentially contained too many activities, so the exercises that were going to be completed through Socrative during class were moved to the end of the EdPuzzle video that students watched the night before.

Meeting #5: Night 14 – Day 19 (2D Arrays)

Due to several failed attempts to find a common time to meet with the group and with the topic of 2D arrays on the horizon, this meeting was canceled and instead conducted asynchronously via e-mail. I sent out a list of questions and two of the three participants responded. One suggestion was to talk about traversing through non-rectangular 2D arrays. In the video, the length of the first row was used as a condition for traversing through every row. While that is okay in the AP CSA course because students are only exposed to rectangular arrays, it was recommended that each row’s length be used as a condition for that particular row, for the sake of practicing good
habits. This was talked about in class the following day. Other than that, the participants appeared to be pleased with the unit and had no other recommendations.

**Meeting #6: Student Interview Questions**

Again, due to scheduling conflicts, this “meeting” was held via e-mail. I sent the focus group a list of the interview questions I was planning on asking. Before sending out the list, I revisited my daily notes, as one of the prompts was to record any additional interview questions I thought would be useful. I copied all of those questions, as well as my initial set of questions before the unit started, on to a new document. After organizing and cutting out redundant questions, I sent the list of questions to the focus group for their feedback. As a result of the feedback, I removed several questions asking about the time spent on various activities and questions revolving around the sense of community.

**RQ2: To What Extent was the Unit Implemented as Designed?**

As discussed earlier, the field notes and daily journal were each analyzed three times using a holistic approach (Saldaña, 2009). This was first done by reviewing each day at a time (looking down one column of the spreadsheet) and summarizing/condensing the comments for each day. I then reviewed each question/prompt at a time (looking across one row) and searched for any patterns. I also reviewed comments I made specific to each activity to see if I could identify any themes within those activities. To do this, one at a time, I set up a conditional highlighting rule to highlight each cell that contained an instance of the given activity. On top of that, on the days an activity was implemented, I reviewed all of the notes I made for that day since I might not have used the actual name of that particular activity while taking notes. In terms of the extent to which the unit was implemented as designed, one main theme
quickly emerged, and that was the underestimation of anticipated time the activities were going to take. This will be discussed this in more detail below.

Lastly, students sometimes revealed relevant information in the interviews that helped me better understand how they engaged in the different activities, so some of their comments were tied in to help answer this research question.

**Time Spent**

First and foremost, the daily activities tended to take longer than expected. Specifically, Socrative, above all, took far more time than anticipated. For example, the initial plan for the first day was to cover fourteen true/false questions and five multiple-choice questions, but after spending 30 minutes on the true/false questions, I moved on to the next activity since I was running out of time. The remaining multiple-choice questions were added to the following day. In addition, the plan for the first day was to have the students trace three algorithms and solve four programming exercises. Considering the 10-minute initial setup time for Socrative and the time spent on each true/false question, there was only enough time for the students to trace one algorithm before class was over. It actually took two days to complete all of the activities that were initially planned for the first day. Rather than making changes to the rest of the unit, I simply extended the unit by one day and initially left the rest of the unit unmodified.

The school's spring break was approaching as the unit was coming to an end and I wanted to ensure I would have enough time to give the tests and put the grades in the grade book before conducting the interviews. In addition, I wanted to conduct the interviews before students left for spring break so the unit was fresh on their minds. While it was not ideal, I decided to remove one day during the topic of 2D arrays to afford me the needed time. Since days 16 and 17 were nearly identical, one of these
days was the most logical choice. Both of these involved critiquing students’ solutions to an AP CSA FRQ (free response question) and then spending the rest of the day working on another AP CSA FRQ.

**EdPuzzle/Videos**

I analyzed the comments I made on EdPuzzle and the videos as well as the data hosted on EdPuzzle’s website, which provided the percentage of the videos watched and when they were completed. One assumption going into this unit was that students would watch the EdPuzzle videos.

On three occasions, one student “completed” three of the videos by answering all of the embedded questions, but the website indicated that they did not watch 100% of the video. This was perplexing because one of the video settings prevented students from skipping the video, so it would be impossible for the students to answer all of the questions without watching the entire video. After talking to the student, they informed me that they found a way to pop the video out of the browser, which allowed them to work on other homework while the video was playing. Despite EdPuzzle’s best efforts to ensure students truly watch the video (e.g., the video automatically pauses when the user changes browser tabs), it appears that this student found a loophole. Fortunately, it looks like the website was still able to track this by only registering the percentage of the video that was watched while the student was on the actual website.

On two occasions, videos were not watched on time. One of the students failed to watch one of the videos on time but then went back and watched it later that same day. Another student failed to watch a video on time, but this student was confused with the directions and watched the optional video instead of the required video by accident and then watched the video later. There was only one student that failed to completely
watch a video, which happened on two occasions. This student did not watch the binary search video at all and completed only 50% of the video on 2D arrays, but it was the longest video and they watched it immediately before class and ran out of time. It appears that this student intended on completing the video before class, as they mentioned in their interview, “it’s just that the lengths [of the videos] varied a lot, so it’s frustrating if I didn't know beforehand because as I said to you that I do a lot of my work in the morning.” Unfortunately, this student never went back to finish the video. For the most part, students watched the EdPuzzle videos as anticipated. With ten videos over the course of the unit, only one video was not completed, one was 50% completed, three were watched late, and three were watched improperly, meaning about 98% of the videos were watched and 91% were watched properly (on time and without changing browser tabs). Still, there were other variables that I was unable to account for. For example, there was no way to know if a student turned the video on, worked on other homework that didn’t require the computer, and answered the questions as they appeared.

**Video Questions**

One of the requirements after watching each video was to post at least one question related to the video. This requirement was developed (a) so that I could identify common areas of confusion and talk about them in class the next day, preparing examples ahead of time if needed, (b) so other students might answer those questions, and (c) so students would reflect on their learning.

In terms of identifying common areas of confusion, I found that students tended to take care of this on their own by joining in on one another’s questions. For instance, questions such as, “I have the same question as [Student A]. I was also wondering if
linear search has to start at the beginning,” were not uncommon. Unfortunately, students did not always take the time to come up with a real question and sometimes posted anything for the sake of fulfilling the requirement, which was a concern by the focus group going into the unit. Take for example these quotes from three different students:

People couldn’t think of the question so you go in and like, oh I’ll just rephrase this question.

And then the video questions just, like, sometimes people would just like, BS them and just make them up just to get it done.

There were a couple times where I didn’t have a question, a couple times that I made a bad… bogus question.

Only one student took advantage of the alternative requirement of posting any question related to arrays as opposed to the video specifically. The answers to this student’s questions were always, ironically, concepts that were going to be covered in the subsequent topic. Clearly, this requirement of posting a question after each video should be reconsidered. More on this will be discussed as I answer RQ5.

As far as preparing examples ahead of time, about half of the questions that students asked required very straight-forward answers that did not necessarily need to be supported with examples. For instance, two questions that were asked were, “does anyone know the code that tells you how much time passed when you run a method in [the] main [method]?” and, “so is linear search just comparing one variable to every variable in an array to find its match?” The other half of the questions were the type of questions that certainly could have been answered with the use of examples, but I tended to answer them directly/verbally for the sake of time. For example, one student asked, “what are some examples of how you would use an array to make a more
efficiency program?” Rather than physically prepare typed examples, we simply discussed a few examples as a class. As a matter of fact, I only found myself preparing one example based on the students’ questions, which stemmed from the following post:

Before watching the edPuzzle video, I'd been using Arrays.toString(array)" to print my arrays (I found it online). When you use this, you have to import stuff (like we did with scanners) and what's printed also comes in the [] brackets. I think I just want to understand this print method more

As a result of the question, I prepared an example to demonstrate a variety of methods for printing arrays.

I initially assumed that students were going to read and attempt to answer one another's questions, but out of over 70 questions that were asked by students in #video-questions, only three questions were attempted to be answered by other students (~4%). While it was not a requirement to answer questions, students were encouraged to look back and attempt to answer their peers’ questions. I encouraged them both verbally in class and by including a reminder with the homework assignment that was posted online.

**Pair-Programming**

Pair-programming, where students worked in a team, one as the driver and the other as the navigator, went for the most part, as expected, although students needed a few reminders from time to time. For example, they had to be reminded on a few occasions of exactly what the roles of the driver and navigator were and were not. There were also times that students’ excitement got the best of them. On several occasions, I observed the navigator grabbing the computer from the driver to take over before it was their turn. It appeared that they were eager and/or frustrated with the
driver and felt they could get the job done better. The general response from the students when I reminded them of the roles was giggling and an, “oh yeah.”

One modification I made after two implementations of the pair-programming activity was to set a five-minute timer. I did this after realizing students were not trading roles as often as I hoped and found myself asking students to change roles. The decision to pick five minutes as the duration was a bit arbitrary so I experimented with different durations in subsequent pair-programming implementations. After ranging between three to five minutes over a few instances, the general consensus was that the class found three minutes to be too fast and five minutes to be a comfortable length. As a matter of fact, a five-minute timer was preferred by some students over no timer at all.

Towards the end of the unit, I forgot to start the timer, and a couple of the students asked if I was going to be using a timer so they would know when to change roles.

Although the timer was used to let students know when to change roles, that didn’t come without its challenges. Because I was both the researcher and the teacher, I was using my computer to take notes when students were not asking questions. I was also using a timer on my computer. I spent the majority of my time away from my computer circulating the room, which caused the computer to go into sleep mode, disabling the timer. Although this would have been an easy fix, being the teacher and researcher required a lot of work immediately before, during, and after class and I consistently forgot to change the setting on my computer. I also found it to be less than ideal when the timer went off and I was working with a group because I had to go back to the front of the room to reset it. Although this is again, another easy fix, it would be
ideal to utilize a timer that automatically resets and starts counting down every five minutes.

The only other modification I made on one day (Day 8), which was done out of curiosity, was to give students the option of choosing between solving the programming exercises individually or using pair-programming. On this day, only one group decided to implement pair-programming while the rest of the groups worked individually.

**Critiquing Students Solutions**

When it came time to critiquing students’ solutions as a class, students may have been reluctant to share all of their thoughts. One student said “you don’t want to just say, like, oh, you’re wrong.” While some students might have been reluctant, it appears that others were able to step outside of their comfort zone and offer their true thoughts. Another student said “I did feel a little awkward sometimes. Like, even though it’s a small class, like you know someone. So, I would be like, oh, this is wrong, this is wrong. I felt a little weird.”

Overall, the students were courteous and mature about critiquing their peers’ programs in front of the class. They were clearly cognizant of their peers’ feelings and wanted to be sure they were critiquing the programs in a professional manner. I didn’t make a single note in my field notes or daily journal regarding any inappropriate comments made during this activity. One student claimed “You need to be able to figure out how to say… it helps with kind of… just like general public speaking, I guess, because you have to figure out how to tell people what they’re doing wrong… like correctly.”

Unfortunately, this activity led to more distracted students than any other activity. The most reoccurring comment about critiquing students’ programs was about students
looking down at their computers when there was no need to be doing so. As the unit progressed, I became more aware of the distracted students and spent less time than initially anticipated on critiquing students’ programs. This was achieved by critiquing fewer solutions.

In an attempt to improve the students’ levels of engagement, I changed the format for critiquing students’ solutions on Day 6. I took all of the students’ responses to EdPuzzle and put them in a spreadsheet, removing all of the students’ names so the responses were anonymous. As a class, we looked at every solution to each question, one question at a time, with the aim of finding as many mistakes as possible. Unfortunately, it did not appear to have any impact on the students’ levels of engagement, as several students were still distracted by their computers.

On a more positive note, I made a last-minute change on Day 9. I initially planned on spending the day critiquing students’ SuperArray classes by projecting hand-picked solutions to the board and working as a class. By this time, with the somewhat limited success of critiquing students’ solutions (in terms of engagement), I randomly generated groups and assigned each group two different students’ solutions to critique and to supply feedback to on Slack. Every group appeared to be extremely engaged and students were communicating with one another more on this day than every day prior. There were only ten minutes left in class and every group was still critiquing the first solution. I forced the groups to move on to the next solution so that everyone would receive feedback. While that was not an ideal balance of time, the students were clearly engaged in the activity. As a matter of fact, the dismissal bell rang

122
and not a single student budged; they continued working. I had to tell the students to get moving to their next class.

**Socrative**

The student response system, Socrative, was employed seven times over the course of the unit. As mentioned earlier, the ratio of *actual* time spent to the *anticipated* time spent was far more for Socrative than for any other activity. I even had to take some of the Socrative questions from the first day and move them to the second day.

The initial plan was to have students first answer each teacher-paced multiple-choice question individually on their laptops and then to discuss their answer with their group and submit a second answer, making a change to their answer if they wish. This was done the first three times, but after realizing how long the activity took, the students were instructed to answer individually the next three times, without consulting a group. As it turns out, this might have been for the better, as students were more likely to change their correct answer to an incorrect answer, than they were changing their incorrect answer to a correct answer after consulting their group. More on this will be discussed later in terms of the effectiveness of the unit. Perhaps this was due to the lackadaisical view that some of the students had on the requirement to consult their group after answering individually. In an interview, one student said:

> After you submit it, you kind of figure out, oh, maybe that was wrong, and then doing it with a partner was redundant. And then you’d look at your partner and you would be like ‘A’ and then you’d hit ‘A’ and then you move on.

Another student admitted, “I found that normally when we were doing it with partners, we just did it separately.”
The last time Socrative was used, students were instructed to answer questions only after consulting their groups. They were free to choose any answer they wished, but they were encouraged to work together to determine the answer. The discussions dissipated after a few questions and I had to remind the students to discuss each question with their groups before submitting their answer. Although every group came to a consensus and picked the same answer (within their group) for all six questions, it appears that they may not have been putting much effort into the question. For example, one student said in an interview: “So there might be one person who is like, I don't know, I think I just chose that one because it looked right.”

Students guessed for a variety of reasons. On a couple of occasions, students came to class late and entered a random answer so they wouldn't hold the class up. On another occasion, one group of students answered a question very quickly and spent the rest of the time chatting about another class while the rest of the students worked on solving the question. Looking back, this particular question required students to read a large segment of code and answer a corresponding multiple-choice question. It would have been nearly impossible for the students to have fully read this question and to have put very much thought into the answer in the time they spent on the question. As it turned out, they both answered the question incorrectly. On other occasions, when students were spending too much time on a question, I started counting down from ten to expedite the process. This certainly did not promote students taking the question seriously, as one student murmured, “well I’m just going to pick one,” as I began counting down. Sometimes students were simply afraid to commit to an answer. This same student later said in an interview:
Actually, it made me realize something, for me, sometimes it's hard to put an answer up there because, like, I'll realize I don't know the information and I don't want to pick the wrong one. So that's something for me. It would be like kind of hard and then I end up guessing if I wasn't sure. Rather than actually thinking about something because I just didn't want to be wrong in front of everyone else, and be the one person that got the wrong answer even though no one would really know. It was just like, kind of hard to do that.

**Algorithm Tracing**

Out of all of the activities, algorithm tracing, which was where students were given an unfamiliar program to read and interpret, was associated with the most confusion in terms of instructions. I was to blame for this, as I was unclear with the requirement. Sometimes I was looking for a one to two sentence, big-picture summary of the algorithm, and other times I was looking for students to comment on each line of the algorithm. In addition, I was not careful enough and let a few algorithms slip by that were too revealing based on the name of the class, methods, etc. One student pointed this out in an interview, “I feel like one of the huge parts of the algorithm tracing questions was looking at the title, you know? Or like, the class name… because, of course, context matters.”

Other algorithms contained more appropriate names (for this activity), such as mystery, and forced students to really scrutinize the algorithm to determine what was going on. It appears that students got a bit too comfortable with the algorithms containing revealing names, as one student said, “sometimes the algorithm tracing would really bother me because some of the questions were easy but some I felt like I had no context for what was happening so I was just like, I don't know what they are doing.”
As the unit progressed, it appeared that students preferred simply solving the programming exercises over the other activities, so I tried to free up time in class to allow for more programming exercises. I also found myself writing in my daily journal that I should include fewer algorithm tracing requirements in the future due to an apparent lack of engagement and satisfaction. With that said, I removed several of the algorithms tracing requirements towards the second half of the unit, and instead either shared the algorithms with the students as an extra resource or turned it into a homework assignment. Interestingly, one student submitted two traced algorithms with their homework that were given as an optional resource.

Overall, the algorithm tracing did not go according to plan because I failed to carefully consider the instruction objective, which resulted in unclear directions. Providing explicit directions for students and removing reveling context within algorithms would have likely yielded more favorable outcomes.

**Slack – General**

Slack, the online communication tool, was used for a variety of reasons and had a channel for each purpose. The channels included #general, #video-questions, #algorithm-tracing, and #student-exercises.

While I expected that students would use #general as they had questions, it was rarely being utilized towards the beginning of the unit. One unanticipated finding was the increased usage of the #general channel over the duration of the unit. To help paint the picture, over the course of the unit, 18 threads, consisting of a total of 36 comments by students, were generated. Of those threads, eleven were generated during the last four days of the unit, and consisted of 25 comments. This means that approximately 69% of optional Slack conversations took place in the final 21% of the unit. Furthermore, based
on the field notes and daily journal, as the unit progressed, students were asking fewer questions in class and appeared to be becoming less dependent on me as the teacher and more open to communicating with their peers in and outside of class. One student acknowledged their group’s persistence on Day 15 by saying in their interview,

If we were just really stuck, or there was like, some error that we had no idea about, you know, [a] new [error], I think we would call you in, but it wasn’t just like the first thing that we thought of.

This group’s mentality, along with the declining number of questions that were being asked and the increasing number of optional conversations on Slack appear to suggest that over the course of this unit, students gained a sense of independence from me as the teacher. Naturally, they seem to have also gained an increased sense of confidence as a group of students.

**Role-Playing**

The role-playing activity, where students physically acted as different components of a program, was initially designed for the search and sort algorithms but after working with the focus group, the decision was made to implement it with the sort algorithms only. As it turns out, it might have been worthwhile for students to role-play the binary search algorithm for two reasons: (1) role-playing, in general, went relatively smoothly and appeared to be a good use of time and (2) the students identified the binary search algorithm as one of the more challenging concepts. Two students in different interviews noted, “coming up with binary search was hard. Like we did it like three or four times and I just couldn't figure out how to do it” and, “binary search is a little harder just because you have to have that, like, ‘mid.’ You have to change the ‘mid’ over and over.”
Only a few, minor, unexpected events occurred during the role-playing activity. One student acted as the `swap` method and their job was to swap two students when instructed. This student attempted to pull both students out and swap them at the same time, which, in retrospect, was good because it led to an important discussion on moving only one element (student) at a time and using a temporary location to get the job done. Another student, who was acting as a method, got started right away without being told to do so by another student who was acting a method, so we had to restart. The only other unexpected event was ending the insertion sort role-playing prematurely. It appeared that the students were grasping the idea (students were practically moving on their own without the method-actors telling them what to do) and we were simply wasting time, so towards the end of the algorithm, I had students go back to their seats.

**CodingBat**

CodingBat, a platform that provides students immediate feedback to their solutions, was utilized four times throughout the unit. The original plan on several occasions was to have students spend any of their remaining class-time working on the CodingBat exercises, but given the actual time required to complete the activities, no students were ever able to start the CodingBat exercises in class that were assigned for homework. With that said, there was only one day (Day 5) that students actually worked on CodingBat in class. For the most part, this went as anticipated. This particular day was one of the quietest days of the unit in terms of students talking with one another. Every student appeared to be very engaged with the exercises and asked me and their peers virtually no questions.

Students were also good about completing the CodingBat exercises on time. Over the course of the unit, the students were required to solve 40 CodingBat
exercises. One student failed to complete one and another student failed to complete two, for a 99.1% completion rate. On the other hand, two students completed two more than were required. I initially assumed that students would not give up on very many exercises, so I was surprised to see the number of exercises that were attempted by students as compared to completed (a correct solution) by students. Between the eight students, 345 exercises were attempted out of a required 320. In other words, students attempted about 8% more problems than the number of required problems.

There were a couple of minor unanticipated events and one major unanticipated event. One student did not know where they were supposed to turn in their solutions, so they had to ask about this on Slack. More explicit directions would have likely prevented this. This same student also later told me in a conversation that they preferred to put their program into Eclipse, the IDE (integrated development environment) used for this class, when they were debugging, which was going above and beyond my expectations.

Unfortunately, one student was caught cheating on the CodingBat problems. When viewing the number of times students were solving exercises on their first attempt, this student had an abundancy of correct, first-attempt answers. Upon further review, the website reported that this student was submitting answers approximately every 15 seconds. A quick Google search yielded a website that matched this student’s answers, exactly, down to the comments, word for word. On some exercises, it appears as though the student attempted to disguise the solution as their own by removing the curly braces when possible, for example. There were only three exercises that this student attempted multiple times. It seems that the student actually tried to solve two of them but gave up and ultimately submitted the solution from the website. The third
question was actually incorrect on the website they found the answers from, so the student had to modify it to get the correct solution. Aside from this one student, CodingBat went smoothly and just as planned.

**Student-Generated Exercises**

Students generated their own exercises at home on two occasions (Nights 4 and 16). The intention was for the students to come in the following day (Days 5 and 17) to have their peers solve their questions and to return home that night (Nights 5 and 17) to respond to their peers’ solutions. While the first implementation went exactly as planned, the second did not.

Two school trips took place on Day 17, leaving only two students in the class. As a result, the class was canceled but as luck would have it, the lab for the class, which met every seventh school day was scheduled for this same day, and some of the students were back on campus by the time the lab period started. Up to this point, the lab period was not utilized during this unit, with the exception one time, where students came in to take a brief quiz. This lab period met during the last period of the day and most of the students were out competing in academic challenges all day, which was likely the culprit of the low levels of motivation. One student was staring at a programming exercise and said, “I really just… can’t.” In addition to the low levels of motivation, the period was shorter than the rest, and there were still several absent students, therefore I altered the requirement. I asked students to solve the peers’ exercise that was assigned to them at home instead of in class. Knowing that the students already went through this routine, I didn’t think I had to be as explicit with the directions the second time around. I assumed that the original author of the exercise would know to respond to the peer that answered their question. I was wrong, because
there was not a single student that responded to the proposed solution, and there were therefore no conversations that took place on Slack.

On the flipside, as mentioned earlier, the first implementation went exactly as intended. Students seemed engaged in the activity and were clearly putting some thought into the proposed solutions. For example, one student, the author of the exercise, responded in Slack:

[Student B], as you already know from [Student C] I think you just misinterpreted the set up of the problem. The array of three strings is the method heading parameter. [Student C], your solution looks more efficient than mine since you are creating the new array, finding the input's length, and returning it all in one step. My solution is longer and less efficient.

Another student wrote:

That was my original solution, because we know the array's length is five, we don't have to do any loops and can just add 2 to the numbers with even indices and 1 to the numbers with odd indices. I think your solution is close though but we just need to change the nums[i] in your if statement to just i.

When students were given explicit directions, this activity went exactly as intended.

Optional Activities

There were three optional videos over the course of the unit: two videos were folk dancers acting out the sort algorithms and one was a video to set the stage for the topic of copying arrays and only contained previous concepts. Looking at these three videos, there was not a single student that watched the entire selection sort video. The percentages that were watched were 70%, 60%, 30%, and 10%. Four students did not watch any part of the video. One student watched the entire insertion sort video, one watched 50%, one watched 30%, two watched 10%, and three students didn’t look at the video at all. This means that on average, the students only watched 16.9% of the
optional dance videos. In terms of the optional video on copying arrays, only one student fully watched it, one student saw 90%, another viewed 80%, and the other five did not watch any of it. In other words, students only watched 32.5% of the video, on average.

There were also four optional programming exercises throughout the unit. While it is possible that some students completed them and never turned them in, there was only evidence of two students that worked on these exercises. One student solved all four and the other student solved one, for an overall completion rate of 15.6%.

Interestingly, there was only one student that did not engage in any portion of any optional activity. In addition, there was not a single student that engaged in some portion of all of the optional exercises.

**Summary**

Many topics were discussed in this section, including the time spent, EdPuzzle, video-questions, pair-programming, critiquing solutions, Socrative, algorithm tracing, Slack, role-playing, CodingBat, student-generated questions, and optional activities. To recap:

- More time was needed to implement the activities than was expected, specifically with Socrative.
- A timer was needed to let students know when to trade roles with pair-programming.
- A large majority of the videos were watched properly (to the extent to which I was able to determine).
- When the students were forced to ask questions in `video-questions`, their peers tended *not* to attempt to answer their questions. When the students asked questions in `general`, their peers *did* tend to attempt to answer their questions.
• At times, fake questions were posted to #video-questions. Of the questions that were asked, they tended to be surface-level questions that could be verbally answered without preparing any examples ahead of time.

• After realizing how distracted students were while students’ solutions were being critiqued as a class, I tried two other designs: (1) projecting every students’ solution for every question at once in a spreadsheet and (2) assigning solutions to groups to critique. Students were far more engaged with the later.

• Asking students to answer each question twice, once individually and again after consulting their group did not go as intended. They were more likely to change their correct answer to an incorrect answer than changing an incorrect answer to be correct. In addition, some students appeared to blow off some of the questions while others were taking more time than expected and therefore, at times, I counted down to pressure them to select an answer for the sake of time.

• The algorithm tracing did not go as intended for two reasons. The directions were not explicit enough and some of the algorithms contained too much contextual information, giving the exercises little value.

• Slack was utilized for general Q&A far more towards the end of the unit as compared to the beginning. Inversely, students asked me fewer questions towards the end of the unit as compared to the beginning.

• The role-playing (method-acting) went as expected, and students picked up the directions very quickly.

• With the exception of one student that cheated by copying answers they found online, the students engaged in CodingBat as intended.

• When students were given explicit directions regarding the student-generated exercises, they followed through, but it was made clear that students will only do the minimum that is assigned.

• Students tended not to engage in the optional activities.

RQ3: What Were Students’ Perceptions of the Unit?

The six interview transcriptions were used to answer this question. The first round of coding consisted of open coding. I went through each transcript and coded sentences/paragraphs with a few words that described the participants’ responses. Some sentences/paragraphs were assigned multiple codes. For example, one student said:
I did feel a little awkward sometimes. Like, even though it’s a small class, like you know someone. So, I would be like, oh, this is wrong, this is wrong. I felt a little weird. Like, they’re probably thinking like, why is [referring to self] saying this?

This was coded as “feeling awkward,” “considering their peers’ thoughts,” and “didn’t want to be a know-it-all.”

The second round of coding was completed after three transcripts underwent the first round of coding in an attempt to establish coding norms. I searched for commonalities amongst the codes and classified them accordingly. For example, “peers not responding” and “peers not living up to expectation” were classified as “relying on peers.” This process was repeated multiple times and the codes underwent several layers of nesting. For example, “group not agreeing” was nested in “challenges with group work,” which was nested in “group work,” which was ultimately nested in “peers.”

Due to the nature of the questions that were asked in the interview, several expected categories emerged, including effectiveness, engagement, or satisfaction. In addition to those, other categories that emerged were general thoughts on the flipped classroom, thought-provoking activities, peers, values, and suggestions.

After the eight categories were established, I went through the first round of coding with the other three transcripts and several new codes were generated. I performed another cycle of the second round of coding and placed them into their respective categories. No new categories emerged.

The third round of coding was a form of evaluation coding (Saldaña, 2009). At this point, I revisited each of the twelve categories and attempted to search for themes within them in terms of positive and negative comments. For example, within the
engagement category, I was able to classify every code as either engaging or unengaging.

Finally, I reviewed each of these sub-categories (e.g., engaging) to search for any general themes. Below, I will discuss the themes that emerged from these eight categories.

**Effectiveness**

The term “effective” is polysemous, so I wanted to get an idea of how the students viewed this term. Although there were a variety of responses, the one common response amongst all six participants was simply the idea behind learning the material. Some incorporated the idea of efficiency and said effectiveness involves quickly learning the material, while others focused on how well they understand the material as a result of the unit.

When asked which activities students found to be the most effective, all of the participants identified some form of programming exercises as their response. Furthermore, the data suggest that students perceived working alone, as compared to working with their peers, as being more effective. Only two students mentioned activities that involved working with their peers, including working as a class to write an algorithm and pair-programming. Those students also coupled those group activities with individual activities and alluded to the idea that working alone was important. One student said, “the programming exercises were nice just because it allows you sort of try on your own,” while the other mentioned that they liked “going through and doing problems, seeing if the way you think about it works out or if you need to adjust the way you're thinking.”
When directly asked about the least effective activity, there was no general consensus amongst the students. As a matter of fact, two students were unable to identify any activity as being the least effective, as one student said, “I don’t want to say anything was the least effective because I don’t think any of it was ineffective.” On the other hand, two other students identified the requirement of asking a question after each video as being the least effective. Although throughout the interviews, four of the students admitted to posting a fake question for the sake of satisfying the requirement, overall, the students didn’t appear to perceive this activity as being completely ineffective. Even one of the students who identified this activity as the least effective found it to be helpful at times and said, “I’d say it helps because you’d see what other people are thinking but then sometimes you can’t really think of a question; you kind of just ask something, that’s kind of just a dead question.”

Overall, students appeared to associate “doing” with being effective, as one student summarized this idea by referring to the most effective activity as, “the programming exercises, because you have to be writing code, you have to be working.”

Engagement

When I asked students what activities they found to be the most engaging, method-acting, creating exercises, and pair-programming surfaced as the most common responses. It looks like the students associated working with their peers as a source of engagement, as all students indicated at least one activity that involved working with their peers as being the most engaging. When asked what it was about the activities that one student identified as being more engaging than the other, their response was, “the social aspect of it rather than just sitting and listening to you.”
Another student claimed “group work was engaging because you’d hear different people’s sides of how to do it. So that’s like the most engaging.”

Conversely, students also identified creating their own exercise, which was done in isolation, as being one of the most engaging activities as well. One student commented on how creating an exercise forced them to think differently and said “having to think about how would someone ask about this, rather than just thinking how would someone answer this.” Other students simply commented that they found it to be very “interesting.”

When asked about the activities that students found to be the least engaging, algorithm tracing was the clear frontrunner, as this was identified by four of the six students. Of those four students, algorithm tracing was the only activity reported by three of them. One student suggested that it was the least engaging “just because it was difficult” while another claimed the opposite and said, “I looked at it for like, four seconds, and I would be like, okay, and then I’d go onto the next one. It wasn’t like, hard at all. It wasn’t challenging.” For one reason or another, it appears that students had the perception that algorithm tracing was an unengaging activity, as many students identified it as the most unengaging, and no one identified it as being engaging on any level.

In addition to algorithm tracing, one student expressed their thoughts on writing pseudocode and said, “that just never did it for me. I just never did it until afterwards.” Another student expressed similar feelings and said, “I feel like it’s sort of a necessary evil, because you just don’t want to do it because it’s just a bunch of time.”
Another student found CodingBat to be sometimes unengaging since it, “at times, did feel like a bit of busy work.” Other students made comments that suggested they felt the same way, for example, by referring to the CodingBat exercises as “a bit repetitive.” One student even found some of the exercises to be so similar that they were copying and pasting their code from previous exercises to solve other exercises.

Students also made comments about the lack of engagement when watching the videos. One student said, “not because they weren't informative, but because again there's that filler space where you can't do anything and you just had to wait until it's ready.” This student went on to later say,

For instance, if there's just 30 seconds or a minute of you just preparing the next, you know, thing to talk about… for instance, when I was just reviewing one of the EdPuzzle videos, I found it very useful just to see 30 seconds, just like, see, okay, this happens, and this happens, and this happens. You don't need to see the way that you do it, you know?

Another student expressed similar feelings, specifically about the dance videos. This student said,

You can't skip which is so that I can't skip to the end I guess and just not do any other activities but those were like, I don't know... like it was useful, and then it's another three minutes and you're like, okay I don't want to watch this video anymore.

Not surprisingly, students appeared to associate engaging activities with activities that were collaborative in nature, new/different, interactive, non-repetitive, and/or exercises that had appropriate levels of difficulty (not too easy, not too difficult).

**Satisfaction**

When asked about the most satisfying activities, five out of the six students reported working on the programming exercises. Of the five, two students made it a point to explicitly say they were most satisfied with the *individual* programming
exercises. One student even said that was their “favorite by far.” Two students simply referenced programming exercises, while the other student found both pair-
programming and individual programming to be the most satisfying. One student found satisfaction in making progress:

    Halfway through class we'll be given four or five exercises, and it would look like a lot and you think like, I have to do all this work, and then eventually you start doing it, and then you get through. You’d be getting in a groove and getting through them.

Several other activities surfaced as being satisfying to students throughout the interviews. These activities include CodingBat, creating and solving peers’ exercises, voting for the best solution, method acting, being paired up with random students, and the use of Slack.

Three of the students recognized CodingBat as being satisfying. Specifically, these three students liked the freedom of being able to choose which exercises they were going to solve. One student said, “I liked the freedom, I guess, of choosing my own because I would go in order and then if there's one I didn't get I could skip it.” This same student went on to say they also liked “the satisfaction of getting those check marks.”

Three students also identified creating and solving their peers’ exercises as being satisfying. One student claimed, “I personally liked it, because it also created a bit of latitude for, you know, what you wanted to do with it.” Another student appeared to be satisfied with challenging nature of the activity and said “I liked that. I liked doing that. I thought that was fun. Sometimes it was sort of a tricky think of one that was good. To look at the other side of the problem.” The third student could not quite explain why they
enjoyed it and simply said “That was something that I really enjoyed. I don’t know why. I found it fun.”

Two students commented on Slack. One student said, “I like Slack a lot. Personally, the way of formatting, it actually allows you to put the code in and it formats it in a non-goofy way.” Another student felt similarly and claimed, “Slack has been like an overall, I'd say, great experience, and just, like, sending code, very easy.”

Although students gravitated towards solving programing exercises when directly asked about the most satisfying activity, there appears to be a general theme around giving students choice. The students tended to point out the freedom they had as the reason they enjoyed an activity.

When the students were directly asked to identify the least satisfying activity, three of the participants identified the requirement to post questions after the videos. One student said it was because they didn’t “really get anything from them.” They went on to say, “It's just like, alright, let me rephrase someone else’s question.” Another student recognized the dance videos as being the least satisfying because they found that they were overanalyzing the video and paying attention to details that were not important. This student was not alone; a second student posted on Slack, “I noticed in the video, the words would get more red or green as they danced, so could it signify being closer to being equal or diverging? am i thinking too deeply about this?” This student was indeed looking too deeply into the details.

Algorithm tracing and method-acting were also brought up. Algorithm tracing was referred to as being “monotonous and boring” by one student, while another student was dissatisfied with the method-acting because they “forgot everything” and had a hard
time connecting the students’ movements with the way a computer would process the information.

Although students were directly asked to identify the least satisfying activity, all six of the students mentioned that they were frustrated with EdPuzzle at times. Three of the students commented on the restricted ability to format the code they entered as their solutions. Two of the students were unhappy with the inability to see their previous responses when the questions were connected. One student said, “you had to write a student class or whatever it was, and then it would be like, use your accessor, and I'm like, wait, I don't know my accessor anymore, and you can't go back to the old question.” One student was simply “annoyed” that they had to do the questions, although they later went on to say that the embedded questions were “good because they made me think about something that I hadn't thought about.”

It appears as though four ingredients to satisfaction are programming exercises, giving students choice, making their progress visible, and user-friendly platforms.

General Thoughts on the Flipped Classroom

The students made a variety of comments on the flipped classroom as a whole, as compared to the traditional classroom. The traditional classroom typically consisted of me projecting my computer screen to the front of the classroom while students followed along on their computers. I taught the basics of the unit over a few days using direct instruction and then gave students the following week or so to solve a set of programming exercises in and out of class. All of the exercises were due the day before the test. The students tended to work alone in class and although they were encouraged to work with one another, they rarely did. During the interviews, students had positive comments as well as criticisms of the new flipped classroom environment.
Students identified a variety of benefits with the flipped classroom environment. For one, students appeared to be more engaged during class. One student said, “I saw fewer people doing other stuff” and went on to say, “I was a little bit more engaged with it than for instance, when we did it before. So, I'd say overall, there was more engagement.” Another student commented, “I really like it because in class, I like participating more, and I feel like I got to participate more.”

Two students commented on the increased structure of the flipped classroom, which helped to prevent them from procrastinating. One student said, “I like this [flipped classroom environment] better because I procrastinate a lot. So, when you give us like a week to do like, 20 problems, I save it to the last two days.” Another commented, “at the beginning of the year I also just didn't really like how we have a certain amount of days to finish something. I like having set deadline for each thing because otherwise, I'm less motivated to… I end up doing it all the night before.”

One last benefit that was identified by a student was the efficiency. This student said, “I thought we learned it way faster this way.” This student went on to say class-time was wasted in the traditional setting because,

We’d have to spend time doing things we already knew how to do a bunch of times, and I feel like that slowed us down a lot. Like every time we built a new class, we’d have to build the constructor, and then the accessors, and then the mutators, and then we’d have to make a toString, and compareTo, and equals, which were things we already knew how to do.

On the other hand, students pointed out several negative components of the flipped classroom environment. What stood out more than anything was the increased workload as compared to the traditional classroom. One student said,

Because the idea is that you learn it at home and then you do the homework in school. However, we only had about 10 minutes to do the
actual work. So that means that just carried over to homework. So that made it, not sure if it was more than intended, or just more.

Although some students commented on the increased structure of the flipped classroom environment, one found it to be too rigid, and commented, “this being all spread out, it was like, every night I'm going to have to do something with this. And every single day, it's like I have to do more of this even though I have a ton of other stuff, it's like this has to get done.”

A comment was also made about how the traditional classroom was “easier on students in a way” because, “it's like you just kind of come into class and just listen and see what happens and you can do your homework in class and you don't really have anything outside.” This student went on to say, “we would just come into class and we would just listen and write things down, and go through it and like 45 minutes later it’s over. Like I didn't really have to actually think about anything if I didn't want to.”

The last criticism was regarding the lack of organized notes for students to return to. In the traditional setting, students were essentially provided an outlined set of notes at the beginning of each unit, considering they typed along during the direct instruction segment. They were able to refer back to these notes at any time while working on the exercises. In the flipped unit, these notes were in the form of a video, so students did not have a set of notes readily available if they did not type along with the video. One student expressed their desire for more formal notes:

Just like have it somewhere so that I know I can… like, some form of notes so that I can go back and look at it, because a lot of it was just like, work with people, or like, work on something, rather than just hear what something is. So, I wouldn't write something down because I'm like oh I'm working on it with someone.
The general consensus seems to be that students preferred the flipped classroom environment over the traditional environment. On the other hand, while some students appreciated the rigidity of the unit, not everyone did. In addition, students felt that there was more work involved, which makes sense because more activities/exercises were included in this unit than with any other unit in the past so they were undoubtedly completing more work in this unit. Regardless, it does appear that students associate the flipped classroom with increased engagement. Spanning the unit out over a few more days and adding more options to reduce the rigid nature of the unit may be solutions to some of the shared criticisms.

**Thought-Provoking Activities**

Throughout the interviews, the students shared a variety of activities that required them to think differently or to put in more thought than usual. Two activities specifically, both of which were identified by five of the six students, surfaced as the most thought-provoking activities. These activities included the requirement to post questions after the videos as well as the student-generated exercises. One student said,

> It made me actually think about a question because normally I just watch the video and be like, whatever, and I’ll move on. But if I didn't think I had a question, it made me actually think about what I saw and go back and really come up with something that I didn’t get.

Another student similarly commented, “it sort of forced me to go back over what I actually knew, and sort of evaluate where the gaps were so it was helpful in the end.”

Likewise, the student-generated exercises appeared to be thought-provoking as well:

> Sometimes I’d feel like I would have the exact same ideas as other people so I would have to, like really try to think about something to come up with. I tried to do something really intense. Like, I know the last question I put up. I was like, I would not have wanted to be the one to answer that.
Three students also commented on the embedded video questions forcing them to think as well. One student described the questions as “good because they made me think about something that I hadn’t thought about.” The same student went on to later talk about their desire to know if the solution they submitted could have worked and said, “I put it in as a solution then realize, oh, I was wrong. But it also made me wonder if the solution that I put in could still work, but I couldn’t… but I didn’t know.”

Three students also found CodingBat to be thought-provoking, specifically in terms of forcing them to think “differently,” although all three of these students still commented on CodingBat as having a number of repetitive exercises. It seems as though there is a bit of a learning curve when it comes to solving the CodingBat exercises, as the style requires students to think differently, but once students developed the type of thinking required, they found the questions to be repetitive.

One student identified the activity where students were asked to vote for the best definition of parallel arrays, as defined by the students, on Slack. This student commented, “you’d have to kind of really think about the nitty gritty details of each one, and to go into each one, like in depth.”

The common theme amongst the most thought-provoking activities, as identified by the students, seemed to be the open-ended nature of the activities. Thinking of a question to ask and creating an exercise were arguably the most open-ended / least structured activities.

**Peers**

All of the students that were interviewed had a lot of thoughts to share about their peers, despite the fact that there were very few interview questions directly related to their classmates. The most frequent comments that were made about their peers
involved pair-programming. Students also talked about difficulties they experienced in terms of depending on their peers, such as responding on Slack and generating clear, quality exercises. Generally speaking, students found it difficult to work with their classmates when it came to pair-programming. The top reason, as mentioned by three students, was because they found that different students tended to have a different thought-process on each exercise. One student said, “each person has their own train of thought, and that can cause problems.” Another expressed their frustration:

At the time, I was just like, ugh, I have to explain everything. And then like, we'll trade off in the middle. And I'm just thinking, oh, this is what we have to do. This is what we have to do. And they're trying to explain to me a totally different process that they want to do. And we've already written half of it one way, so we can't go back anymore, so it was annoying sometimes.

In addition to the varied thought-processes, students shared that they simply didn’t work well with some students:

Like some people, I just don't work well with. And it's just like, we're sitting there, and it's literally like, I'll write something and then it's... hand the computer over. There's not a lot of communication going on. And at that point, it's just inefficient for what it is.

This student went on to say, “some people I just don't work well with. And I think they clearly don't work well with me but they work well with other people.” Another student similarly said, “I don't know a lot of kids in this class. Like, I guess, I know them, but I don't know them that well. It's hard to work with somebody.” One student, when talking about a benefit of the traditional environment pointed out that they “didn't have to work with people, or ended up working with someone who I didn't know very well, which freak me out, like the random groups. I was like, please just put me with someone I know.”
Working with peers wasn’t all bad, though. One student commented, “it is good because it helped me, like, I talked to people I would never talk to.” Likewise, another student said, “at the same time I don't think it's a bad thing. I think it’s good to get kids to work together, especially if you're all in class together.”

Three students had very similar comments about Slack, specifically about their frustration with depending on their peers to respond. One student said, “I'll ask a question at like 8:00 [PM] and then someone doesn't answer until 11:45 [PM] and I'm already asleep by then, and then it doesn't help, and then I can't get what I need to get done.” Similarly, another student said, “if I needed to ask something, I just wouldn't get an answer, either at all, or the day after.” These claims are true, as there were several occasions that students’ questions posted in the #general channel (not the required questions) were either completely unanswered, answered very late at night, or within an hour of the beginning of class. Interestingly, students were far more likely to answer their peers’ questions that were posted in #general than they were in #video-questions. As mentioned earlier, student only attempted to answer about 4% of the questions that were asked in #video-questions. In #general, of the 22 questions that were asked, the students attempted to answer 17 of them (77.3%), and I answered three of them (13.6%). So, although students shared their frustrations with waiting to hear from their classmates, it appears that Slack was still useful and students attempted to help one another out.

The last set of comments that were made about the students’ peers was regarding the lack of quality self-generated exercises. One student said, “students… they’re not the best at making questions. They’re either really easy are really hard. So,
there’s not really an in-between.” Another student commented, “not everyone’s going to give you an example of how to do it because they’re just coming up with a question real quick."

This activity could potentially be improved based on one student’s recommendation to require students to include samples to help students understand the exercise. This student said, “I feel like making them do an example of it, as you see in the CodingBat exercises would really help.”

Clearly, students had higher expectations of their peers than what their peers were actually capable of or were willing to produce. Including more requirements with the student-generated exercises might be beneficial. Regarding Slack, it would be worthwhile to have a conversation at the beginning of the unit about students’ expectations when asking for help. Since it is an optional location to seek help, students should not assume that their questions will be answered. Perhaps some form of motivation could be offered in an effort to promote students answering questions. In addition, pair-programming appeared to be a challenging task for students, depending on who they were paired with. This goes hand-in-hand with comments discussed earlier hinting that students tended to prefer working alone. Regardless, the ability to work with others is an important skill to have. It might be that these students simply needed more practice, or maybe students need to be explicitly taught how to work as a team.

**Values**

I define students’ values as the extent to which they see an activity as being useful. Between all six students, the terms “helpful” and “useful” were used to describe various activities. Interestingly, between all six students, every single activity was
referred to as being either “helpful” and/or “useful,” although no one particular activity surfaced as being the most helpful/useful. To quote a few students:

I found Slack really helpful for collaborating with other classmates outside of class.

At the end [of the videos], I noticed you did some multiple-choice questions or just one final question. I think that’s a very useful, sort of way to wrap up the topic.

I found that helpful for test preparation. – CodingBat

It sort of forced me to go back over what I actually knew, and sort of evaluate where the gaps were, so it was helpful in the end. – Requirement to post questions after the videos.

I do like pair programming. But then it's nice to look on your own and to be able to look over all of it. Maybe even make changes you want to make. But it’s helpful to talk it out with somebody else.

I think Socrative and the algorithm tracing are helpful for being able to read code, which we need to be able to do.

Because you have an idea, and you have to communicate what that idea is. While thinking about it sort of, probably, like, allowed me to sort of have to think about, like, how to express it, that probably helped. – Pair-programming

Although the students were not unanimous with their perceived helpfulness/usefulness of these activities, only two activities were identified that students found to be questionable. One student expressed their concerns with pair-programming and said, “When we were working together, I'd be worried, like, would I [be able to] come up with this all on my own? Can I like, re-create this later on?” Another student commented, “We sort of already knew what the sorts and searches were, so I'm not too sure how useful the ones were where we had to trace a binary search.”
Overall, it seems as though the perception is that students found value all of the activities, but on varying levels and for different purposes.

**Suggestions**

The students were directly asked for their suggestions but also indirectly made suggestions about improving the unit throughout the interview. When students were directly asked about what changes they would to make to the unit, the time spent on the different topics tended to be the focus of most students’ responses. The sorting methods and concept of copying arrays surfaced as activities that students wanted to spend more time on. When asked where the time could come from in order to spend more time on these activities, students identified the basics of arrays and parallel arrays. Students rationalized with comments such as, “Basics of arrays, I feel like everyone got pretty quickly.” and “Parallel arrays, I think were, I think most people just understood it right away.”

On the other hand, one student recognized how much time was still left in the year and suggested adding a few more days to the unit as a whole. Interestingly, this student, who was one of the strongest students in the class, suggested spending more time on the basics of arrays, and said,

I think it took me a few days to understand how arrays worked at all, since we had never talked about anything closely, remotely, about arrays. So maybe spending one more day or two, just about like, the very basic ideas of arrays. Like a group of ints and how they work and what's [the] index? Index starts at zero. Like more of the basic stuff would be nice

This student went on to suggest spending another day on the topics of objects that contain arrays and the concept of copying arrays (linked and unlinked arrays).

When asked if there were any activities that the students would like to see continued if the class were to shift back towards a traditional setting, there was no
single activity that stood out more than the others, although many activities were mentioned. Answers included videos, asking questions after videos, Socrative, CodingBat, creating exercises, direct programming exercises, and being placed into random groups. One student responded, “honestly pretty much all of it. I really liked it a lot.” With that being said, there were no students that directly commented on their desire to continue algorithm tracing, pair-programming, critiquing solutions, role-playing, and working as a class to write algorithms. Interestingly, critiquing solutions, role-playing, and working as a class to write algorithms were the only three activities that were completed as a class and were three of the five aforementioned activities.

**Summary**

Many themes emerged as the transcripts were analyzed. These were focused on effectiveness, engagement, satisfaction, general thoughts on the flipped classroom, thought-provoking activities, thoughts about peers, values, and suggestions. Below is a summary of the students’ perceptions:

- Students associated the most effective activity with the programming exercises. Specifically, they felt that programming on their own was more effective than with their peers.

- The least effective activity perceived by students was posting questions after watching the videos. While not all students directly identified this as the least effective, the majority admitted to posting fake questions, at times.

- The most engaging activities perceived by students were method-acting, creating exercises, and pair-programming.

- The least engaging activity perceived by students was algorithm tracing.

- Students found the programming exercises to be the most satisfying, just as they found them to be the most effective.

- The least satisfying activity as identified by students was the requirement to post video questions, again, as it was perceived to be the least effective.
• Students perceived posting questions after watching the videos as well as creating their own exercises as the most thought-provoking activities.

• At times, students struggled to work with their peers both in and out of class. They talked about their lack of familiarity of their classmates and also shared their frustration with the lack of communication/answers from their peers on Slack when they sought out help.

• Every activity was valued by at least one student. The students offered their own unique rationale for finding value in the activity.

• The students felt they we should have spent more time on the unit, specifically on the concepts of sorting and copying arrays.

RQ4: Was There Any Preliminary Evidence to Support the Effectiveness of the Unit?

Three activities, including Socrative, EdPuzzle, and CodingBat were useful in determining the potential effectiveness of this unit, as their platforms provided plenty of quantitative data.

Socrative

Socrative was implemented seven times, and each question was solved using one of three different formats, including working (a) individually, (b) as a pair/small group, (c) individually and then as a pair/small group. When considering the individual responses that were followed by group responses, the students, as a whole, were correct 71.3% of the time (n = 159 responses) when they answered individually. When the same questions were asked and students consulted their group members before answering the same question a second time, this success rate increased slightly to 72.8%. When looking at all of the individual responses, regardless of whether or not they were followed up a second time with a group (formats a and c), they were correct 69.6% of the time (n = 256). When students solved exercises as a group only, they saw the questions for the first time as a group and were told not to submit an answer until
they consulted their group. They did not have to agree with their group member(s), so they were allowed to submit different answers if they disagreed. In this format, the students answered correctly 79.2% of the time (n = 48).

As mentioned earlier, students were actually more likely to change their correct answer, which was found individually, to an incorrect answer after consulting with their group, which happened with 10.7% of the responses (n = 159). Only 5.0% of the responses were changed from being incorrect to correct as a result of consulting their group. This also means that 84.3% of the responses were unchanged. Approximately 53.5% of the responses were correct both times, while 30.8% of the responses were answered incorrectly individually and then incorrectly again after the students consulted their groups. One possible explanation for this is, in addition to the haphazard attitudes of some of the students described earlier, may be due to the levels of confidence displayed by the students within the group. One student said:

I would just say that would just depends on the confidence the other person had and do they actually think they know it. So, there might be one person who is like, I don't know, I just chose that one because it looked right. Or it could have been like, you know, no, obviously, it's this one you doofus [joking]. And then it would just be like, who had bigger confidence in their answer.

With all of this being said, there appears to be some evidence to suggest that students are most successful when they work together throughout the entire problem, from start to finish.

**EdPuzzle**

Many of the embedded video-questions were designed to get students thinking about the material / motivated to learn the material rather than as a diagnostic tool. For example, in the video on 2D arrays, students were asked, “Given what you know about
initializer lists, how do you suppose you would initialize an array of arrays?” It was not an expectation that students were able to answer that correctly as a result of watching the video, as that question was embedded before it was taught. Naturally, these types of questions were not used to determine the effectiveness of the videos. On the other hand, there were many embedded questions the expectation was that students were able to answer them as a result of watching the videos. Over the span of the ten videos, there were a total of 48 questions, 27 of which were questions that students were expected to answer correctly as a result of watching the videos. These questions included true/false, multiple-choice, and open-ended formats. The open-ended questions were marked as either completely correct, partially correct, or incorrect. To distinguish between incorrect and partially correct answers, I examined the correct portion of their answer. If their answer indicated some understanding of a new concept introduced in the video, then it was marked as partially correct. For example, one student attempted to initialize an array of Strings by typing:

```java
String students = {“A”, “B”, “C”};
```

This student forgot to put rectangular brackets after String, which was a new concept. One the other hand, this student successfully used an initializer list (= {“A”, “B”, “C”};), which was another new topic in the video. As a result, this was marked as partially correct. In another example, students were tasked with replacing the middle element of an array with a new element. One student’s response was:

```java
Pens[3] = pen6;
```

This student should not have capitalized Pens, and they used the wrong index as well. The capitalization issue was possibly a function of the platform that students were
using, as they were unable to look back to see the array of pens that they instantiated earlier. Regardless, the convention is to use lower camel case for variables, which the student should have done. Either way, this student was able to demonstrate that they understood how to replace an element in an array with a different element, so it was marked as partially correct. Simply put, if the student was not able to demonstrate any new knowledge as a result of the video, it was marked as incorrect. For example, one question asked students why the binary search method could not always be used to find an element in an array. One student responded, “nothing, a binary search can always be used in a given array,” which was marked as incorrect.

With a total of 19 free-response questions, 4.1% of them were answered incorrectly, 11.6% were partially correct, and 84.3% were completely correct (n = 147 responses).

The true/false and multiple-choice questions were marked as either incorrect or correct. Between these eight questions (n = 60 responses), 28.3% of them were answered incorrectly. Breaking this 28.3% down, 18.3% came from only two questions while the other 10.0% came from the other six questions. Interestingly, these six questions were more conceptual in nature that could be answered relatively quickly, such as, “A binary search cannot always be used in a given array (true/false).” On the other hand, the other two questions involved reading a segment of code and answering a question. For example,
Consider the method below, which is intended to return true if there is at least one duplicate in the array, and false if there are no duplicates.

```java
boolean hasDuplicate(int[] nums) {
    for (int k=0; k < nums.length - 1; k++) {
        if (nums[k] == nums[k+1])
            return true;
    }
    return false;
}
```

Under which condition will the method not necessarily produce the desired result?

a. When the array is sorted in increasing order.  
b. When the array is sorted in decreasing order.  
c. When the values in the array are all positive.  
d. When the values in the array are all the same.  
e. When the array has at most 2 elements.

Looking at all of the questions together (closed and open-ended), 11.1% of the 27 questions (n = 207 responses) were answered incorrectly and 8.2% were partially correct, meaning 80.7% were answered correctly.

Another potential indicator of the effectiveness of the videos was the number of times students re-watched them. Ideally, students would have only needed to watch each video one time to understand the concept. Higher levels of re-watching videos were associated with less effective videos.

EdPuzzle is formatted in such a way that the teacher of the course can access the number of times various segments of the video were watched by individual students. By default, each video is broken up into ten equal segments, regardless of the video’s length. Therefore, between the ten videos that were assigned, there were a total of 100 video segments. Students were allowed to skip through the linear search video, so that video has been omitted for the sake of this discussion, meaning there were actually 90 required segments of video. Of the 90 segments, 29 segments (32.2%) were re-
watched at least one time by at least one student. Of those 29, 14 came from the two
sort videos, which is not much of a surprise, considering these were both less than one
minute each. Given the short duration of these videos, students were likely either more
willing to re-watch them or the videos were simply too fast.

While 29 video segments were re-watched, that number is a bit deceiving. After
considering the total number of segments for all eight students over all nine videos,
there were 720 segments to be watched. As a reminder, one student failed to watch one
entire video and half of another, meaning 705 segments were watched at least once. Of
those, there was a total of 74 segments (10.5%) that were re-watched between all of the
students, some of which were re-watched multiple times. Of the 74 segments, 49 came
from the two sort videos. Clearly, the majority of the segments that were revisited were
from the sort videos.

When removing the two sort videos from this discussion, there were 545
segments that were watched at least once. Of those, 25 segments were re-watched
(4.6%). Looking at the sort videos only, there were 160 segments, 49 of which were re-
watched (30.6%). Again, it appears that the sort videos were the least effective videos
and the design of those videos should be revisited. On the other hand, it’s possible that
the embedded questions were asked prematurely. Perhaps if the questions were asked
later in the video, students would have been more prepared to answer them. Regarding
the other videos, a mere 4.6% re-watch rate suggests that those videos were rather
effective.
Between the low re-watch rate (10.5%) and the high success rate with the embedded questions (80.7%), it appears as though the videos were relatively effective. Further developing the sort videos should certainly improve those statistics.

**CodingBat**

As a reminder, one student cheated and copied answered that were found online. Fortunately, it looks as though the rest of the students attempted the questions honestly. This discussion will only include data on these students. Between the seven students, 306 exercises were attempted and 282 were solved successfully, for a 92.2% success rate.

Generally speaking, higher rates of unsuccessful attempts were associated with giving up the exercise entirely. On average, students successfully completed the exercises in 3.5 attempts (n = 282 completed exercises) and gave up after 6.8 attempts (n = 24 incomplete exercises). Looking at the exercises that were successfully completed, on average, 38.3% of them were solved on the first attempt. Looking only at the exercises that students struggled with (unable to complete on their first attempt), they were still able to complete them in about 5.0 attempts, on average.

Interestingly, looking at the extreme cases, the story was quite different. The most attempts any one student took on any one problem ranged from 6 to 30, with an average of about 16.4 attempts (as their maximum). Looking at the exercises associated with the maximum number of attempts, six of the seven students persevered and solved it correctly. Only one student abandoned the exercise that they attempted the most. As mentioned earlier, although the students tended to abandon the exercises after they were unsuccessful too many times, it seems as though they felt a certain level of commitment to some problems. While the average number of attempts on exercises
they were unsuccessful with was 6.8, there was still a total of 32 exercises that students took seven or more attempts on to solve correctly.

Overall, with a 92.2% success rate, it looks like CodingBat was quite effective. The students that struggled had the option to solve similar problems to questions they were able to answer and the more determined/motivated students had the option of pushing themselves if they wished.

**RQ5: What Were the Lessons Learned and Resulting Final Changes Made to the Unit?**

**Lessons Learned**

- Considering the number of mistakes that were identified by the focus group, I need to spend more time reviewing the content and/or work with other teachers to check over any materials I will be using with my students.

- All activities appear to have some value. At some point, every activity was listed by at least one student as having a benefit.

- The activities took longer than expected, specifically Socrative.

- Explicit directions for every activity need to be given to students. Students failed to properly complete tasks when the directions were not explicit. While this was sometimes due to the unclear directions involved with some of the activities, at other times it was due to my own unclear expectations of the students. This was most notable when students were asked to trace various algorithms.

- Assumptions should not be made about the level of familiarity and/or comfort students have with one another. Even though the class size was only eleven, eight of which agreed to participate in this study, some of the students did not know and/or where not comfortable working with some of their peers. This caused challenges when students were randomly paired up with one another.

- The activities in this unit cause students to become more comfortable with each other. This was an interesting finding because these students had been in the same class for over a semester before the unit began. In addition, students became less dependent on me as the teacher and were more active on Slack asking and answering their peers’ questions.

- On the spectrum of working completely independently to working as an entire class, students leaned more towards the preference of working alone. While
some students appeared to be fine in the middle (working as a small group), the general consensus was that students disliked working as an entire class.

- Students have higher expectations of their peers than their peers are either capable of or are willing to produce. For example, students complained about the lack of timely responses to questions asked on Slack as well as the unclear peer-generated questions.

- Students tend to be more satisfied when they have choice. For example, students enjoyed the freedom to choose the questions they solved on CodingBat as well as the open-ended nature of generating their own, original exercise.

- Students like seeing their progress, such as the checkmarks on CodingBat, for example.

- EdPuzzle is a very robust platform, but there are two areas for improvement: (1) allowing students to see the previous questions/responses, and (2) a more user-friendly text field that allows for students to write code that is formatted nicely.

- The most thought-provoking activities are the most open-ended / least-structured. For example, while students did not tend to favor the task of coming up with a question to ask after each video, it surfaced as the most thought-provoking activity.

- When exercises that are found online (such as CodingBat), there is a chance that students will cheat and search for the solutions.

- While there are a lot of activities that can be employed, students perceive the most effective activity as simply solving programming exercises.

**Final Changes Made to the Unit**

As a reminder, the unit was implemented as outlined in Table 5-1.

<table>
<thead>
<tr>
<th>Day(s)</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>Basics of arrays</td>
</tr>
<tr>
<td>6</td>
<td>Arrays of objects</td>
</tr>
<tr>
<td>7</td>
<td>Copying arrays</td>
</tr>
<tr>
<td>8-9</td>
<td>Objects with arrays</td>
</tr>
<tr>
<td>10</td>
<td>Parallel arrays</td>
</tr>
<tr>
<td>11-12</td>
<td>Searching</td>
</tr>
<tr>
<td>13-14</td>
<td>Sorting</td>
</tr>
<tr>
<td>15-17</td>
<td>Two-dimensional arrays</td>
</tr>
<tr>
<td>18</td>
<td>Review</td>
</tr>
<tr>
<td>19</td>
<td>Test</td>
</tr>
</tbody>
</table>
Below is a list of the changes to future implementations of the unit as a result of the previous findings:

- **Changes to the time spent:**
  - I was unable to have the students begin the programming exercises on Day 6, so an extra day was added to the topic of *arrays of objects* to allow students time in class to work on these exercises.
  - Students identified the desire to spend more time on understanding how to *copy arrays* and less time on *parallel arrays*. To accommodate, I took the day that was devoted entirely to *parallel arrays* and combined it with the topic of *copying arrays*, resulting in one and a half days on *copying arrays* and half a day on *parallel arrays*.
  - An additional day was added to the topic of *sorting*. One entire day will still be devoted to *selection sort* and another day to *insertion sort*. The third day will be spent on understanding the differences between the two sort methods.
  - Due to time constraints, one of the days on the topic of 2D arrays was omitted. Moving forward, that day will be kept due to the lack of programming exercises within this unit that incorporate 2D arrays. Every year, one of the four free-response questions on the AP exam include 2D arrays, so the College Board clearly views this as an important topic.

- **Changes to specific activities:**
  - Since the binary search method was found to be more challenging to students than expected, the method-acting activity will be implemented with the binary search method.
  - All of the algorithms that students are required to trace will be stripped of any context that offers hints. For example, instead of a method name, `findMin`, it might be called `mystery1`.
  - The two sort videos, which were both less than one minute will be slowed down to half of the speed, considering the number of times they were re-watched.
  - Since students identified the topic of copying arrays to be one that they wanted to spend more time on and since the majority of the students did not watch the optional video that set the stage for the video on copying arrays, it will not be optional in the future.
  - A five-minute timer that automatically resets will be implemented with pair-programming.
• To encourage honesty on CodingBat, two unsuccessful attempted exercises will be accepted as the equivalent of one completed, correct exercise. Hopefully, with this addendum, students will feel less pressure to search online for correct solutions.

• Considering the number of students that posted fake questions after each video, this requirement has been updated. Moving forward, since there were ten videos, there will simply be a requirement to post a total of ten questions over the course of the unit. This will give students more freedom, such as posting multiple questions about one video, or posting questions about homework exercises, for example.

• When implementing Socrative, students will only answer questions after consulting their group member(s). They will be encouraged to try to come to a consensus but will be still allowed to submit different answers.

• The embedded questions in the EdPuzzle videos will be changed so that they do not depend on the students’ previous answers.

• General changes:
  
• To help students generate an organized set of notes, the code from the videos will be posted in Slack.

• Instead of assigning the unfinished classwork as homework, it will be left as an optional assignment. With that said, since the unit has been extended by three days, the class activities are more spaced out, so there are fewer exercises for students to solve in class, making it more likely that they will complete them in class.

• Students will be given more opportunities to solve programming exercises individually. Some of these will come from the pair-programming exercises and some will come from the new days that have been added. Towards the beginning of the unit, the programming exercises will be solved via pair-programming and gradually shift towards individual programming.

  To recap, in addition to some of the smaller changes, three days were added to this unit, resulting in one extra day each on the topics of arrays of objects, sorting, and 2D arrays. The day that was spent on parallel arrays has been modified to include more practice with copying arrays. For an overview of the updated pacing, please see Table 5-2.
Table 5-2. Overview of updated pacing.

<table>
<thead>
<tr>
<th>Day(s)</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-5</td>
<td>Basics of arrays</td>
</tr>
<tr>
<td>6-7</td>
<td>Arrays of objects</td>
</tr>
<tr>
<td>8</td>
<td>Copying arrays</td>
</tr>
<tr>
<td>9</td>
<td>Copying arrays &amp; Parallel arrays</td>
</tr>
<tr>
<td>10-11</td>
<td>Objects with arrays</td>
</tr>
<tr>
<td>12-13</td>
<td>Searching</td>
</tr>
<tr>
<td>14-16</td>
<td>Sorting</td>
</tr>
<tr>
<td>17-20</td>
<td>Two-dimensional arrays</td>
</tr>
<tr>
<td>21</td>
<td>Review</td>
</tr>
<tr>
<td>22</td>
<td>Test</td>
</tr>
</tbody>
</table>
CHAPTER 6
DISCUSSION AND IMPLICATIONS

This chapter discusses the results identified in the previous chapter. In addition, a set of implications are outlined for (1) me as an educator, (2) the context of flipped computer science classes, (3), the field of flipped classrooms, and (4) the field of computer science.

Summary of Study

With the number of students enrolling in computer science classes rapidly increasing (Cateté, 2018), lack of qualified computer science teachers (Qian et al., 2018), and low retention rate in computer science programs (Giannakos et al., 2017), I set out to develop a flipped unit of instruction for a computer science class to address these issues. The purpose was threefold: (1) develop an effective unit that promotes positive perceptions of computer science, (2) provide insight for teachers that are unqualified or new to the field of computer science, and (3) improve the quality of my own instruction.

A flipped environment was selected because it afforded the opportunity for active learning experiences inside the classroom. As discussed in Chapter 1 and supported in Chapter 5, active learning appears to the key ingredient for increasing student engagement. In addition to the active learning students engaged in during class, they watched videos that were developed based on principles of CTML at home.

This 19-day unit was implemented in an AP CSA class with eleven students, eight of which agreed to participate in the study. The unit focused on arrays and the topics included the basics of arrays, arrays of objects, copying arrays, objects with arrays, parallel arrays, searching, sorting, and two-dimensional arrays.
This design and development research followed a two-phase mixed methods design. After initially designing the unit, I worked with a focus group to further develop the unit in the first phase. In the second phase, I implemented the unit with my students, collecting field data along the way, and interviewed six students after the unit was over. The interviews were transcribed and the data were analyzed qualitatively and quantitatively. Once the analysis was complete, final revisions were made to the unit for future implementations.

**Additional Limitations**

Throughout the unit, several new limitations surfaced and are discussed below:

- The attendance of the focus group members was subpar, and at times, some of the members that were absent from a given meeting did not provide any input even after the meeting. In addition, one of the members transitioned roles from being a high school teacher to a software developer and was unable to participate in the second half of the study.

- Between the eight participating students, six of them were absent at least once over the 19-day unit, with a total of 16 absences. This 89.5% attendance rate could have impacted the students’ perceptions of the unit, as students that were absent likely found the material that they missed to be more challenging.

- Since there was an odd number of students in the class, pair-programming was sometimes implemented with groups of three. This was not the case every time since students were sometimes absent. Pair-programming was not designed for three students to work in a group, so this could have caused some of the frustrations that students shared.

- I was sometimes unclear with my directions, specifically when it came to algorithm tracing and pair-programming. There is a good chance that this influenced the students’ perceptions of those activities.

- It appears that EdPuzzle only reports when students watch a full segment (10%) of the video, so the statistics reported earlier are not necessarily exact, and students likely watched more than was reported.

- One final limitation was my inability to take comprehensive field notes while I was in class, which naturally resulted in fewer observations. Although some notes were taken during class, my main priority was to provide an authentic learning experience for the students, using only my downtime to record observations. On
the other hand, this limitation was slightly offset by recording notes in a daily journal within 45 minutes of the conclusion of each class.

**Discussion of Results**

In this section I will further discuss the results outlined in the previous chapter. These results will be discussed in terms of the design and development of the unit, the implementation of the unit, and the evaluation of the unit.

**Design and Development**

As discussed earlier, this unit was initially designed based on principles of CTML, active learning techniques, and recommendations made by previous researchers. Many of the activities I implemented in this study were new to me (e.g., pair-programming, role-playing, comment first coding) which I discovered as I reviewed the literature. I then applied what I learned about instructional design through the University of Florida’s doctoral program to design this unit. Still, since I was a relatively new computer science teacher (less than five years), and was lacking a formal computer science education, I worked with a focus group to further develop the unit so I could reap the benefits of their experience.

The focus group consisted of three AP CSA teachers and one former AP CSA student. With only four other members, I was very surprised to learn how difficult it was to find common meeting times. In retrospect, I feel as though it would have been wise to have considered the participants’ availability before including them in the study. When considering the six “meetings,” there were a total of eight absences between the four members, for a less-than-ideal 67.7% attendance rate. Four of the absences included in this number were attributed to the participant that changed roles from a high school teacher to a software developer in the middle of the study.
Regardless, working with the group was worthwhile considering the countless mistakes they caught. These mistakes ranged from small spelling errors/typos to larger issues, such as multiple-choice questions containing more than one correct answer, for instance. They also offered suggestions in the spirit of clarity and consistency. For example, the group recommended that I used the term *method* instead of interchanging it with the term *function*, which I was inadvertently and incorrectly doing over the past four years. This is a great example of the limitations of teachers lacking a formal computer science education, such as myself. In this particular case, I didn’t realize I was misusing the term and no one was around to tell me otherwise (before this study).

The experience of the focus group was evident, as many of their suggestions were perceived positively by the students. For example, they recommended allowing students to pick their own CodingBat questions, which several students were satisfied with. Having students act as the methods was another recommendation and was also perceived as being one of the most engaging activities.

It was also useful to have multiple sets of eyes on the videos. For example, I didn’t realize that I spent the first half of one video *preparing* for the new concept, which was pointed out by the focus group. As a result, I broke this video up into two sections and made the first part optional. Although it is no longer an optional video, I will continue to keep the videos separate considering the nature of the topics covered within those videos.

Naturally, inexperience/unqualified computer science teachers stand to learn from other experienced computer science teachers, as they possess the content knowledge, ideas, and activities that can be shared.
Implementation

Some of the activities went exactly as intended while others did not. Underestimating the actual time required for some of the activities and this unit as a whole was the largest obstacle when it came to implementation. Specifically, Socrative took far more time than initially anticipated. This was a result of requiring students to first answer the question individually and then again after consulting their group, as was done in a study by Simon et al. (2010). This did not appear to be a good use of time because the students did not tend to try very hard to come to a consensus with their groups, as mentioned by multiple students in their interviews and recorded in my field notes. While I explained to the students that they were to discuss the questions, I failed to explain to them that they were to attempt to come to a consensus before answering a second time (assuming they would). It’s possible that they might have tried harder if I was clearer with my directions. While Socrative stood out more than the other activities in the way of taking more time than anticipated, the other activities all generally took slightly more time than expected. As a result, many of the activities that were planned to occur in-class, most of which were programming exercises, were turned into homework. This is certainly not in the spirit of a flipped classroom since one of the goals of a flipped classroom is to bring the traditional “homework” into the classroom, making it imperative that more time is devoted to this unit. With that said, it is important to recognize that students still engaged in active learning every class period; they just were not able to complete all of the planned in-class activities in class.

On several occasions, students had the choice to engage in optional activities, including watching videos and solving programming exercises. There were also other, less direct opportunities to engage in optional activities, such as interacting on Slack in
the #general channel or going above and beyond the directions for any given activity. The students tended not to complete any optional exercises that were formally listed as “optional” and they generally did not go above and beyond the directions, either. This was best exemplified by considering the lack of responses to the students’ required questions posted in the #video-questions channel. As a reminder, students were required to post a question after each video but were not required to attempt to answer their peers’ questions, although they were encouraged to do so. Still, only about 4% of the students’ questions were attempted to be answered by their peers. On the other hand, students engaged in far more conversations in the #general channel, which was not associated with any form of requirement or officially labeled as “optional.” Here, students attempted to answer 77% of the questions that were asked by their peers. It seems as though when the students didn’t associate an optional activity with a formal assignment, they were more willing to engage. In this particular instance, this may have been due to the students’ perceptions that other students were posting fake questions at times and didn’t think that they were truly looking for an answer. They may have associated the #general questions with legitimate questions and were perhaps more motivated to respond.

Another interesting finding was the decreased sense of independence the students had on me as the instructor throughout the unit. Students were asking me more questions in class towards the beginning of the unit than they were by the end of the unit. Inversely, students were found spending more time interacting with each other on Slack towards the end of the unit. This is likely explained by the increased sense of comfort that students gained with their peers as a result of the group activities. As a
reminder, amazingly, some students admitted to not knowing other students in the class before the unit began. Considering all of this, I was impressed with this shift over the short duration of the unit. The students adjusted nicely to the flipped classroom environment, especially considering the first half of the year was spent in a more traditional environment.

Other hiccups included unclear directions on my part. This was most noticeable in the algorithm tracing and pair-programming activities. Unfortunately, I failed to determine exactly what I was looking for the students to do in the algorithm tracing activity, so it was no wonder why the directions were unclear. At times, I wanted them to summarize, holistically, what the given algorithm’s purpose was. Other times, I wanted them to break down the algorithm, line by line, and to write comments that explained what was happening. Both styles of tracing were appropriate at times, but it depended on the algorithm that was being traced, so the directions should have been clearly outlined before they started the exercises. For example, while on the topic of sorting, it probably would not have been valuable to have students determine, holistically, what the purpose of a given algorithm was if its purpose was to sort an array of objects. Instead, as suggested by McConnell (1996), students could work in groups to trace the algorithm, line by line, in a role-playing fashion, to compare different sorting algorithms.

Pair-programming came with its difficulties as well, which also stemmed from unclear directions. Heeding the advice from Umapathy and Ritzhaupt (2017), I attempted to clearly explain the pair-programming process and guide/support them along the way, but I fear I was unsuccessful. Recognizing that the activities were taking longer than expected, I fear that I rushed through the directions and expectations since I
found different pairs approaching the activity differently. For example, one group treated the driver role more as a stenographer with the navigator as the person providing all of the logic. When this happened, I guided them back on track but seeing this provided evidence that my directions were not clear enough.

I also made several false assumptions in the design and development phase, one of which might explain some of the students’ negative perceptions of pair-programming. I assumed, considering the small class size (and school in general), that all of the students in the class were already comfortable with each other. As mentioned earlier, this was not the case and likely impacted the students’ ability to work as a team. This assumption was the culprit for my lack of attention I paid to several guidelines I failed to follow (Williams et al., 2008). These guidelines will be further addressed later when I discuss the implications of this study. Another false assumption that I made was that students would maintain academic integrity throughout the unit. While only one student submitted solutions that were not their own, it is important to remember that there were only eight students involved in the study. There are many reasons students cheat, such as GPA pressure, peer pressure, and heavy workloads (Geddes, 2011). Regardless of the reason, it’s important to recognize that academic honesty is not guaranteed by all students, which should be taken into consideration during the instructional design process.

The overarching issue was the lack of time devoted to the unit. Allotting more time could have fixed several of the other unexpected events. For instance, without time pressure, I would have likely spent more time discussing the directions and expectations with the students on each activity. Additionally, students might have
participated more in the optional activities if more time was allocated to the unit since they would not have been as overwhelmed with classwork that spilled over into homework. As mentioned earlier, one surprising finding was the decreased sense of dependence the students had on me as the teacher and increased sense of comfort students had on their peers, which was likely derived from the forced group work throughout the unit.

**Evaluation**

Once the unit was initially designed and then developed with a focus group, it was implemented with my students. Data were collected during the implementation, and interviews were conducted with six students after the unit was complete. The data were used to evaluate the unit in terms of the students' perceptions and its potential effectiveness, both of which are further discussed below.

**Students’ perceptions**

Students in this study shared both positive and negative perceptions of all aspects of the unit. Interestingly, they associated satisfying activities with effective activities on both ends of the spectrum. Students perceived the programming exercises to be the most effective and also reported it as the most satisfying activity as well. Likewise, they perceived the least effective and least satisfying activity to be posting questions after videos. Recent research has suggested similar findings. One study found that the perceived effectiveness of a flipped classroom had a significant positive effect on students’ levels of satisfaction (Pérez, Collado, Del Mar García de los Salmones, Herrero, & San Martín, 2019). This might be explained by Hatcher, Prus, Kryter and Fitzgerald’s investment theory, which suggests that students perceive their time, energy, and effort as an investment and so they expect to see a return
Students in the present study alluded to this very idea that they wanted to see a return on their investment. For example, when they felt that the activity was a waste of time, they found themselves to be dissatisfied, making comments such as, “it would really get annoying if I'm doing the same question like ten times.” Likewise, students justified their dissatisfaction with comments suggesting that some activities were “monotonous,” or that they “forgot everything,” or that the limitations of EdPuzzle made things difficult, to mention a few examples. On the other end of the spectrum, the central theme surrounding satisfied students was the ability to complete the activities and see the return on their investment. This was mentioned in the form of students directly identifying completing the programming exercises as satisfying as well as by bringing up the satisfying nature of seeing the progress they were making.

More on the topic of satisfaction, one study that utilized free, online resources in a higher education flipped computer science course resulted in dissatisfied students (Baldwin, 2015). The students were resistant to the out-of-class activities and felt that they were being left to learn on their own. The main complaint from the students was the lack of consistency between the different sources. The students in the present study did not share similar feelings, supporting the idea that the out-of-class activities should be from the same source. As a reminder, all of the videos in the present study were from the same source (myself), with the exception of the search and sort videos, which were from the same sources. The students also appeared to be satisfied when choice was involved. For example, they were satisfied with the freedom to select any CodingBat exercises they wished (to an extent) as well as the freedom to create their
own exercise. Although giving students choice appears to result in satisfaction, too much freedom may cause problems. More on this will be addressed later as I discuss the implications of this study.

In terms of engagement, two of the three activities that students perceived as being the most engaging were method-acting and pair-programming, both of which were collaborative activities (the third was creating exercises). This finding is in line with another study where students that were learning computer programming in a flipped environment designed for collaborative experiences reported higher levels of engagement than in traditional classrooms (Paez, 2017). Although the students perceived the engaging activities to be more collaborative in nature, they did not appear to be interested in continuing the class-wide activities in the future, as there was not a single student that identified a class-wide activity that they would like to see continued in the future. Looking at these activities, which were critiquing solutions, role-playing, and working as a class to write algorithms, there does not appear to be any connection between the extent to which they were perceived as effective, engaging, and/or satisfying. The only conclusion that can be made is that the students appeared to be most engaged in the activities that were completed in small groups and individually as compared to the class-wide activities.

The students identified the most thought-provoking activities to be (1) the requirement to post a question after each video, which was perceived as one of the least effective and least satisfying activities, and (2) creating their own exercises, which was perceived as one of the more engaging activities. Of course, this is not enough to say that thought-provoking activities are perceived as engaging, ineffective, and
unsatisfying. This is backed with the number of other activities that were perceived, on some level, to be engaging, ineffective, and/or unsatisfying and not perceived as the most thought-provoking. Likewise, students perceived the requirement to create their own exercises as one of the more satisfying activities. The most that can be said is that thought-provoking activities appear to stem from open-ended and less structured activities, as both of the aforementioned activities were arguably the most open-ended and least-structured of the unit.

In one study (Yağci, 2017), students in a higher education blended computer science class expressed their concerns with the lack of teaching time when the class shifted to the blended environment. While the students in the present study did express a desire to spend more time on specific topics, namely copying arrays and sorting, the specific concern of a lack of teaching time did not surface at any point. In the aforementioned article (Yağci, 2017), it was not made explicitly clear what the instructors or students did during the class period. The only interaction that was mentioned was through a blog tool for students to communicate with their instructors. In the present study there were many different activities, all of which contained opportunities for student-teacher interaction. With that being said, anyone implementing a flipped classroom would be remiss if they were not fully present and involved with the activities the students engage in.

Students shared their concerns about the increased workload with this unit, which was also shared by students in another study who were in a higher education computer science class and felt that the blended format required more work when it came to preparing for exams and participating online (Djenic et al., 2011). As mentioned
earlier, there actually was more work in this unit as compared to the traditional units from the past, so it’s not surprising that students felt this way. The students were not alone; as the instructor that developed these materials, I experienced an increased workload as well, which has been identified by many others as a challenge for teachers who flip their classrooms (Giannakos & Krogstie, 2014). It would be interesting to see if students still felt this way in future implementations of the unit, which will be spread out over three additional days.

To recap, students in this unit tended to find satisfaction in activities that they perceived to be effective, which might be partially explained by the investment theory of students’ satisfaction. With that said, it appears as though one method for increasing levels of satisfaction is to focus on improving the effectiveness of the unit. In addition, like other studies, this study suggests that students associate engaging activities with those that are collaborative in nature. On a positive note, students did not express any concerns regarding the lack of teaching time within this unit, alluding to the idea that it is important that teachers are also engaged in the activities. On the other hand, students did share concerns with the increased workload, although future implementations of this unit will be spread out over three additional days, ideally alleviating these concerns.

**Effectiveness**

In this study, I was interested in looking at the potential effectiveness of the activities that provided quantitative data, namely Socrative, EdPuzzle, and CodingBat. As a reminder, students solved Socrative in a variety of formats: (1) answering first individually and then a second time after consulting their groups, (2) answering individually only, and (3) answering only after consulting their group. The first method yielded opposing results to another study (Simon et al., 2010) that resulted in students
performing better when they answered first individually and then again after consulting their groups. Students in the present study were actually more likely to change their correct answer to an incorrect answer after consulting their groups. Moreover, less than 16% of the students even bothered to change their answers after consulting their group. This is likely the result of the students' lack of effort and interest in solving the question a second time, which was seen at times in class and surfaced throughout the interviews. Regardless, the students performed best when implementing the third format (when they answered once, only after consulting their groups). In this format, they answered nearly 80% of the questions correctly, which is not surprising since students that work together have been found to outperform students that work individually (Linton et al., 2014). With that said, considering the time spent answering questions twice and the lack of interest, it would not be advisable to have students answer questions twice. It appears to be a better use of time to have students work as a group only and to encourage them to come to a consensus before selecting an answer.

The videos students were assigned to watch appeared to be rather effective. As a reminder, these videos were developed with principles of CTML in mind and also contained embedded quiz questions. Although watching the videos and answering the questions was a requirement, the students did not need to answer the questions correctly to earn full credit. Despite this, they still performed well on the questions, answering over 80% of them correctly. Furthermore, nearly 90% of their answers were either completely correct or partially correct, meaning they showed some new knowledge on 90% of the questions that were asked, suggesting that the videos themselves were effective. This is more significant when considering the students’
frustrations with EdPuzzle. As a reminder, they were unable to see their previous answers, but I inadvertently overlooked this platform’s shortcoming and asked questions that depended on their previous answers. Additionally, students found the task of typing code into the text box to be cumbersome. Both of these factors most likely had a negative impact on some of the answers that were marked as completely incorrect. It is also worth noting that with this high success-rate, students were found re-watching only about 10% of the videos, suggesting they tended to understand the first time around.

Also, the relatively high success rate with Socrative further suggests that EdPuzzle was effective. Socrative was typically employed the class period immediately following the night a video was assigned and tested students on their understanding of the concepts covered in the video. While this information alone does not suggest whether it was the videos alone or the combination of the videos and embedded questions that were effective, previous research (Jonsson, 2015, Lacher, Jiang, Zhang, & Lewis, 2018) suggests that it was the combination of videos and embedded questions.

CodingBat appeared to contain a blend of easy and challenging questions, but there may have been too many easy questions. Nearly one-third of the questions that were attempted by the students were solved on the first attempt with the remaining exercises taking an average of five attempts to complete. As a reminder, the students were given the option to select which questions they wanted to solve, and over 90% of the attempted questions were solved correctly. They also showed determination on the challenging questions. For six of the seven of the students (omitting the data from the student that plagiarized), the question that had the most attempts was not abandoned.
and eventually solved correctly. This demonstrates that to an extent, the students were “hooked” on these questions, so it is a bit peculiar why students did not report CodingBat as one of the more engaging activities. Looking at the data closer sheds some light on this phenomenon. Although the most extreme number of attempts corresponded with a correctly solved exercise for most students, the general trend was the greater the number of attempts, the more likely they were to abandon the problem, which may rationalize the disconnection between CodingBat and engagement.

The disconnection might be explained by this idea of “flow,” which is described as “a subjective state that people report when they are completely involved in something to the point of forgetting time, fatigue, and everything else but the activity itself” (Csikszentmihalyi, Abuhamdeh, & Nakamura, 2014, p. 230). It has been said that at least three conditions must be met in order to achieve flow: (1) the activity needs to have “a clear set of goals,” (2) there needs to be “a balance between perceived challenges and perceived skills,” and (3) there needs to be “clear and immediate feedback.” (p. 232). Since the activity contained a clear set of goals (correctly solve ten exercises) and provided students with clear and immediate feedback (outputs for specific test cases were provided), it appears that there was not a balance between perceived challenges and perceived skills. This condition essentially states that if the challenge exceeds skill, then students will become anxious and give up, and if the skill exceeds the challenge, students might become too relaxed and bored. In other words, the challenge level and skill level need to match. As a reminder, the students did not unanimously report CodingBat as unengaging. It appears that some students achieved flow while others did not. For example, students made comments such as, “I liked that
because I would try to do the first ten, wouldn’t get four of them, and then try to do the second ten, wouldn’t get two of them, and then I just keep like, going through until I could find ten that I can do,” and, “I liked the freedom, I guess, of choosing my own because I would go in order and then if there’s one I didn’t get I could skip it.” These comments exemplify the idea behind naturally balancing the challenge with skill. On the other hand, some of the students simply solved the exercises in order and may not have had the opportunity to identify an appropriate challenge. Take for instance these comments: “I didn’t realize [that I was allowed to pick any ten], I just did the first ten” and “I'm like, really lazy, so I just did the first ten.” These same two students went on to make comments by referring to CodingBat as, “a little bit too monotonous” and “a bit repetitive,” respectively. With that said, it appears that the requirement to select any ten exercises on CodingBat provides an opportunity for students to achieve flow, but it is unclear whether or not there are enough challenging questions for all students to achieve flow since some of them simply went in order and solved the first ten exercises.

One area of concern with CodingBat is the accessibility of solutions that can be found online. A quick Google search for any given CodingBat exercise will yield a multitude of solutions. Since there is not much that can be done to prevent students from searching for answers, one solution might be to simply have an open conversation with the students about cheating/dishonesty before the unit begins.

This section focused specifically on Socrative, EdPuzzle, and CodingBat because they all provided quantitative data. Students were rather successful with all three activities, which means it is likely that there were other effective activities that led to their high levels of achievement. For example, as mentioned earlier, nearly one-third
of the CodingBat questions were answered correctly on the first attempt. That means
the students were prepared going into those CodingBat questions, which had to stem
from another activity. Exactly which activity is unknown, but it is safe to say that as a
whole, this unit was relatively effective. The next step is to determine exactly which
activities, specifically, were the most effective.

Implications

This study has implications for me, for the field of flipped computer science, for
the field of flipped classrooms in general, for the field of computer science in general,
and for novice computer science teachers. I will discuss these implications below.

Personal Implications

I learned several personal lessons as a result of this study. In short, I will be
clearer with my directions, I won’t make assumptions about students, I will test new
activities before including them in the development of a unit, and I would like to find a
way to make students’ progress more visible. To better explain, I was unclear with my
directions on several occasions, which isn’t surprising since upon reflection I realized
that there were times that I didn’t have a clear image of exactly what I wanted the
students to do, namely when it came to algorithm tracing. This was a rather large
oversight on my part, as I failed to fully develop the instructional objectives in the
instructional design process, which is an important component to facilitating effective
learning (Morrison et al., 2006). I attribute this negligence to my diverted attention to the
large number of activities that I was implementing with my students for the first time. I
was also unclear with my directions with pair-programming, given the number of times I
had to remind students how to implement the activity. Moving forward, I need to spend
more time considering exactly what it is I would like the students to be doing and how I
can best communicate those directions. The first guideline for implementing pair-
programming recommended by one group of researchers was that “students need
training in pair programming” (Williams et al., 2008, p. 447). I cannot imagine this is any
different when it comes to algorithm tracing. In the future, when implementing new
activities with my students, I plan to demonstrate the activity in front of the class. For
instance, I will first show the class a video (found on YouTube, for example) of students
effectively implementing pair-programming. I will then ask two students to serve as
volunteers to try it out as the class watches. As they work together, I will point out
mistakes and highlight when they are properly implementing the activity.

I also made the mistake of making many assumptions about my students. For
example, I assumed that they all knew one another, that no one would look up solutions
online, and that they would only need to implement an activity once in order to
remember the directions. I will not make any of these assumptions from this point on. To
make sure students are familiar with one another, I will implement ice-breaker activities
at the beginning of the year (this is discussed more below). I will also be cognizant of
the solutions that are readily available online and discourage students from searching
for them by having an open conversation at the beginning of the year about academic
dishonesty. Lastly, I will continue to repeat the directions for each activity until I feel
confident they no longer need to hear them.

When I wish to include a new activity in a unit, I will test it out with my students
before the unit begins. For example, if I implemented Socrative before the unit, I would
have had a much better sense of the time it was going to take and I would have
developed the unit accordingly. Likewise, if I had students trace algorithms before, I
would have recognized that I was unclear myself of exactly what I wanted the students
to do and would have fully developed the activity before the unit began. CodingBat, on
the other hand, was employed before the unit, so I had a very good sense of how it was
going to go for the students, which was why it went smoothly.

Lastly, I would like to find a way to make students’ progress more visible to
increase the levels of engagement and satisfaction. Perhaps one method is
gamification. Gamification can be defined as “using game-based mechanics, aesthetics
and game thinking to engage people, motivate action, promote learning, and solve
problems” (Kemp, 2012, p. 10). Gamification has been infused in blended learning
classes and has resulted in increased motivation when compared to blended classes
without gamification (e.g., Tan & Hew, 2016). Several other studies have found that
gamification that included badges led to increased motivation and engagement (Gibson,
Ostashewski, Flintoff, Grant, & Knight, 2015; Kumar, 2012), as well as increased
attendance (Caton & Greenhill, 2014). Even when game-like elements such as scoring
and smileys are incorporated into learning tools that aren’t necessarily games
themselves, students have found them to be more attractive and motivating than
traditional methods (Kickmeier-Rust, Hillemann, & Albert, 2014).

**Implications for the Context of Flipped Computer Science Classes**

In addition to personal implications, this study has implications that are specific to
anyone interested in flipping their computer science classes. These implications range
from recommendations on the usage of specific platforms to general implementation of
a flipped computer science unit. Starting with Socrative, based on the results of this
study, it is advised that instructors consider having their students answer each question
only once and only after consulting their group. It is also recommended that the
instructor strongly encourages the groups to come to a consensus before submitting their answers (Simon et al., 2010). One way to promote this might be again through gamification. Perhaps individual correct answers are worth one point or badge and groups that are unanimously correct earn an additional point or a different badge.

EdPuzzle is a fantastic platform with many benefits. It is a great choice if the aim is to reduce the extraneous cognitive load, leaving more room for germane load to help students commit their learning to long-term memory (Sweller et al., 1998). The extraneous load is reduced a number of ways. For one, the quiz questions are directly embedded in the video so students do not need to leave the website to go elsewhere. In addition, the video is automatically paused when students change tabs on their browsers, somewhat preventing them from multitasking. With that said, it does not come without its limitations. Students are unable to review their answers to previous questions, so this should be kept in mind when deciding what questions to ask in the videos. Naturally, questions that depend on previous answers are not a good choice. The second limitation is specific to computer programming. The text boxes that students type their answers into is not user-friendly when it comes to writing code, so this should also be kept in mind when deciding what questions to ask. Asking students to write large segments of code is not advisable. Also, as mentioned earlier, the videos should be from the same source, when possible.

The results of this study alluded to the idea that students did not favor the class-wide activities and tended to prefer working alone or in smaller groups. Although that is the case, that does not mean the class-wide activities do not have any value and should be completely dropped. On the other hand, this is something that should be kept in mind
for instructors looking to increase levels of engagement and satisfaction, as students were most distracted and dissatisfied with the class-wide activities. As a reminder, active learning, as defined in the context of this study, is simply an umbrella term for engaging students in the learning process (Prince, 2004), which does not necessarily mean working with others. Students should be given multiple opportunities to work individually in class not just because many students preferred it, but also so they can assess their individual understanding.

It should not go unrecognized that students were satisfied when they were given choice. While this does not mean they should be given full freedom, it should be considered when developing flipped units for computer science courses and can be done a number of ways. For example, rather than asking students to create a specific object with predetermined instance variables and methods, they can be asked to create an object of their own choosing. The requirement might be a \textit{number} of variables and methods as opposed to \textit{specific} variables and methods. Another example is allowing them to pick a given number of CodingBat problems, rather than a specific set of questions. While giving students choice should increase their levels of satisfaction, this should not come at the expense of their learning. In other words, if there is not enough structure to the directions, students may not gain the full value of the activity. For instance, if one of the goals of the activity is to have students practice overloading methods (methods with the same name in the same class) and this is not explicitly outlined in the directions, then they likely won’t get the practice they need. In other words, when students are given choice, the instructor needs to carefully consider the goals and to ensure the directions align with those goals.
Finally, instructors looking to flip their computer science classes should not be overly-ambitious with the number of activities that can be completed in one class period. In this study, more often than not, I did not have enough time to complete all of the activities that I planned for. As a result, those activities were carried over as homework, which was reported as a bit overwhelming for students at times. Instead, instructors might consider leaning more towards the conservative side regarding the anticipated time the activities will take. Perhaps students could be allowed to begin working on their homework in class if they finish the class activities early. This will also allow for more individual programming opportunities, which students should receive, as mentioned earlier.

**Implications for the Field of Flipped Classrooms**

This study also has broader implications for anyone looking to flip their classes, regardless of the subject matter. These implications revolve around working in groups and the usage of Slack. As a reminder, there were only eleven students in this class, eight of which agreed to participate in this study. Remarkably, there were some students who were not familiar with other students in the class, even halfway through the year. Considering this small class size, one can only imagine that this phenomenon is exacerbated with larger classes and in larger schools. With that in mind, instructors interested in implementing a flipped unit would be ill-advised if they did not begin the year with ice-breaker activities to get the students more comfortable with one another. In addition, given the decreased dependence students had on me as the instructor and increased levels of communication they had with one another over the course of this unit, instructors might consider implementing a flipped unit early in the year so their students can reap these benefits over the entire year.
It has been recommended that instructors provide an environment that encourages students to ask questions and to respond to their peers both in and outside of class (Shen et al., 2009). Although Slack was a wonderful environment for this computer science class, there may be other, better tools for different subjects that should be considered. Regardless of the platform used, an open conversation about expectations should be held with the students before it is utilized. Specifically, the expectations students should have of one another should be discussed. In this study, students sometimes had rather unrealistic expectations of their peers, for example, by asking questions late at night about an exercise that was due the next morning. The instructor should also be present and interact with the students but should be conscious of their level of involvement. It has been suggested that it is more important for students new to programming to collaborate with their peers than it is to obtain high-order thinking skills (Hadjerrouit, 2008). As such, I view the online role of the instructor as a role model for proper interaction and not necessarily as an answer-provider. If the instructor is too involved in answering questions, I fear that students will be discouraged from attempting to answer their peers’ questions and become too dependent on their teacher. In other words, instructors should consider giving their students an opportunity to attempt to answer their peers’ questions before stepping in. It has also been recommended that the students are made aware that the instructor is monitoring their online presence (Köppe et al., 2016). One way to achieve this is to join the conversation with a simple “thumbs up” emoji to a student’s correct response to their peer’s question or to write an encouraging message about their response, for example, “Nice job Student D, I really liked the way you explained that.”
In this study, students identified the requirement to post a question to Slack after watching each video as both the least effective and least satisfying. On the other hand, they felt that it was one of the most thought-provoking activities. While I attribute the increased optional usage of Slack (#general) as the unit progressed partially to this requirement, instructors should consider revising the requirement to discourage students from posting fake questions. One method might be to simply give students a minimum number of questions that need to be asked over the duration of the unit. Another method to encourage online interaction could potentially be to change the homework grading policy. For example, students that interact on Slack about a particular homework question might be graded on effort whereas students who do not interact on Slack are graded on accuracy. Perhaps gamification is another method. The instructor might assigns points or badges for questions asked and answers provided. Ultimately, the goal is to get students interacting on Slack so they can support one another with their learning. Instructors should consider methods that promote online discourse and discourage posting fake questions.

**Implications for the Field of Computer Science**

In addition to personal implications and implications for both flipped computer science classes and flipped classrooms in general, this study has implications for the general field of computer science as well. These implications include topics of dishonesty and pair-programming. As mentioned several times before, if the solutions to an exercise are available online, instructors should not assume that students will refrain from searching for them. There are a couple of methods for discouraging students from searching for answers online. Probably the most obvious but time-consuming technique is to write original questions so solutions cannot be found online. Given the time it takes
to write quality, original questions, this approach might not be the most appealing. One potential solution to reducing the time commitment is to work with a small group of trusted teachers to develop original exercises and who also agree to refrain from posting the questions/answers online. Other than that, as mentioned earlier, another method is to have an open conversation with the students about academic integrity.

In this unit, students were not overwhelmingly in favor of pair-programming. I suspect that was partially my fault due to my unclear directions, false assumptions I made about my students’ levels of comfort/familiarity with one another, and failure to use advice from previous researchers (Williams et al., 2008). Specifically, four guidelines I inadvertently failed to follow include, “instructors should provide a systematic mechanism for obtaining students’ feedback about their partners and must act upon the feedback when indications are a student is not being an equal participant” (p.448), “instructors should attempt to maximize the chances students will work well together” (p.449), “students must understand that problems with their partner must be surfaced immediately to give the instructor a chance to correct the situation” (p. 450), “pairs should be able to comfortably sit next to each other as they work, and both should have easy access to the monitor, mouse, and keyboard” (p.450). I failed to solicit feedback from my students about their partners to identify potential issues, I randomly assigned pairs, and each pair was crowded around one laptop. As a reminder, the main complaint from the students in this study revolved around the difficulties they experienced working with their peers, suggesting that these guidelines have merit and should be followed by instructors interested in incorporating pair-programming in their classes.
Implications for Novice Computer Science Teachers

Many implications were discussed up to this point, which any teacher may stand to benefit from. So, what about teachers new to the field of computer science, specifically? While new computer science teachers certainly can benefit from the implications that have already been discussed, below is a list of implications/suggestions for these teachers, specifically. Most of these have all been discussed already, but are written more as a set of guidelines that are more straight-to-the-point and written in a language that is catered directly to these teachers.

• Talk to / work with other computer science teachers, even if they are from other schools. Request to join the “AP Computer Science A Teachers” group on Facebook to make connections. Even in my fifth year as a computer science teacher, I learned a lot from the focus group in only a handful of conversations.

• Offering computer science in a flipped environment is a very feasible option. In your first year, you might consider using videos from another source instead of creating them yourself. Although, be sure to use videos from the same source, as much as possible. This will give you time to better learn the content yourself and will offload some of the pressure on you as the instructor. As you gain more confidence, consider creating your own videos, and remember to keep the principles of CTML in mind while creating these videos.

• Do not underestimate the power of simply having students solve programming exercises individually. Students should have opportunities in class to work on these programming exercises individually.

• With that said, do not be too extreme with any one activity. Students have a variety of preferences and found value in all of the activities. A little at a time, pilot new activities with your students. Most importantly, talk to your students to get their feedback. You may be surprised to see how honest they will be with you.

• Avoid passive learning and get your students learning actively. In other words, get your students “doing.” Here is one way to think about this: if you are implementing an activity where all eyes are supposed to be on you, then many of your students are likely bored or distracted and not actively learning.

• Include the use of ice-breakers at the beginning of the year to get students more comfortable with one another. Ideally, these students will spend a great deal of
time working together throughout the year, and it cannot be assumed that they will already be comfortable with each other.

- Have an open conversation about plagiarism and academic integrity at the beginning of the year. Verify your students’ solutions from time to time to check for originality. While it’s not a perfect system, one way to check for plagiarism is to take a snippet of their code, put it inside of quotation marks, and perform a Google search. Again, this does not guarantee whether or not a student plagiarized, but you may start to identify patterns with some students.

- Slack is a fantastic platform to include with your computer science students. To get students communicating with one another outside of class, assign work that incorporates the use of Slack so they get more comfortable with it and so they begin to see the benefits. You should also be active on Slack where you should be acting as a role-model.

- Finally, relax, and enjoy the wonderful field of computer science! Know that you are not the first person to be put into this position, and you will not be the last. Also, although not ideal, know that it is most certainly possible to teach computer science without a formal education in computer science. So, be patient with yourself and don’t expect to be perfect the first time around. Each year will be easier than the previous year. Good luck!

**Recommendations for Future Research**

Throughout this study, several questions emerged and paved the way for areas of future research. Below is are several areas of research to consider.

This study focused solely on the development of a flipped unit on the topic of arrays in an AP CSA class. This study can be used as a template to develop other topics in computer science. With the number of unqualified teachers that are being asked to teach computer science, pulling together our efforts as a community of researchers will help shed some light on how these teachers can best approach teaching various computer science topics.

Since the majority of this study was qualitative in nature, there is not much to be said in terms of the generalizability of the extent to which the unit can be implemented as developed. In other words, this unit was developed and later modified, but it is
unknown how well the unit would work with other teachers and their students in different contexts. For example, how well would this unit hold up in a class containing 40 students in a school with a low graduation rate? This unit can be implemented with a larger sample of students and quantitatively analyzed to further investigate the levels of effectiveness, engagement, and satisfaction of the various activities and the unit as a whole.

As discussed earlier, there appeared to be value in all activities that were implemented in this study. It would be interesting to dig deeper into each activity to better understand the extent to which these have value to ultimately determine an appropriate balance of time to devote to each activity. For example, Socrative consumed a lot of time. Perhaps that time could have been better spent on another activity.

It would be worth determining whether or not CodingBat provides enough challenging problems for all students (or at least the vast majority) to achieve “flow.” It would also be interesting, with more participants and exercises, to see if there exists a threshold in terms of the number of attempts that students make and stay committed to a problem. In other words, I wonder if students are most likely to stay committed to an exercise if they attempted a problem at least 20 times, for example. If that were the case, it might be worth updating the CodingBat requirement to either complete an exercise or attempt it a minimum of the threshold count.

It looks as though gamification has a promising future in blended learning. Future researchers might consider how gamification can be incorporated into flipped learning in a way that promotes achievement, engagement, and satisfaction, in the context of
computer science classes. There have been many studies in the area of gamification, some of which have been discussed earlier, that should be reviewed before undertaking this task.

**Summary**

This design and development study started with an initial design based on my conceptual framework which consisted of CTML, active learning, and recommendations by previous researchers. After working with a focus group to further develop this unit, students engaged in a variety of activities and later shared their perceptions of the different components of the unit. For the most part, the unit went as expected with the exception of the anticipated time versus actual time spent on the activities. Students identified activities that they perceived to be effective, engaging, and satisfying, as well as activities that were on the other end of the spectrum. In addition, students’ perceptions aside, the potential effectiveness of unit was investigated quantitatively. Combining the results, a set of lessons learned and final revisions to the unit were outlined.

As a reminder, this study attempted to fully develop an effective flipped unit of instruction on the topic of arrays in an AP CSA class, with the aim of (1) improving students’ perceptions of computer science, (2) assisting teachers that are unqualified or new to the field of computer science, and (3) improving the quality of my own instruction. As to the first point, students shared positive and negative perceptions of the unit that was designed and then developed with the focus group. Based on these perceptions, changes were made to the unit in an effort to further improve students’ perceptions of computer science. These changes were outlined in Chapter 5 of this dissertation. On the second point, a set of implications were outlined earlier in this
chapter for teachers that are unqualified or new to the field of computer science. Certainly, teachers of all backgrounds could also stand to benefit from these implications as well. These implications have been arranged for those that are interested in (1) flipped computer science classes, specifically, (2) general flipped classrooms, (3) general computer science, and/or (4) teachers new to the field of computer science. Finally, I am confident that the quality of my own instruction has improved as a result of this study, not only in terms of this particular unit, but as a general educator as well.
# APPENDIX A
## FIELD NOTES TEMPLATE

<table>
<thead>
<tr>
<th>Class Number</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>In-Class Activities</td>
<td></td>
</tr>
<tr>
<td>Topic</td>
<td></td>
</tr>
<tr>
<td>Student Questions</td>
<td></td>
</tr>
<tr>
<td>Student Comments</td>
<td></td>
</tr>
<tr>
<td>Student Behaviors</td>
<td></td>
</tr>
<tr>
<td>Student Discussions</td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>Algorithms Traced</td>
<td></td>
</tr>
<tr>
<td>Class Number</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---</td>
</tr>
<tr>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>In-Class Activities</td>
<td></td>
</tr>
<tr>
<td>Topic</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Were there any absent students?</td>
<td></td>
</tr>
<tr>
<td>What were the groups?</td>
<td></td>
</tr>
<tr>
<td>Did I prepare any examples based on students’ questions from Slack?</td>
<td></td>
</tr>
<tr>
<td>Did I cover everything that I planned to cover?</td>
<td></td>
</tr>
<tr>
<td>Did I make any modifications today?</td>
<td></td>
</tr>
<tr>
<td>Were there any issues/special events that occurred (e.g., fire drill)?</td>
<td></td>
</tr>
<tr>
<td>What are my general thoughts about the day?</td>
<td></td>
</tr>
<tr>
<td>What are my specific thoughts about the day?</td>
<td></td>
</tr>
<tr>
<td>How engaged did the students appear to be?</td>
<td></td>
</tr>
<tr>
<td>How satisfied did the students appear to be?</td>
<td></td>
</tr>
<tr>
<td>How much did the students appear to be comprehending?</td>
<td></td>
</tr>
<tr>
<td>Is there any other relevant information?</td>
<td></td>
</tr>
<tr>
<td>What would I do differently next time? / What changes to the unit should I make?</td>
<td></td>
</tr>
<tr>
<td>Are there any additional interview questions that I would like to add?</td>
<td></td>
</tr>
<tr>
<td>Copy and paste the Slack conversations that took place from yesterday to now here.</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C
GUIDING QUESTIONS FOR THE FOCUS GROUP MEETINGS

- **Time:**
- **Date:**

**Questions**

- Did you find any mistakes with the materials?
- What are your thoughts on the current state of instructional unit?
  - How about the pacing? Is enough / too much time being spent on each topic?
  - How about the workload? Are the students receiving enough / too much work?
- What are your thoughts regarding the individual activities?
  - Do you think any activities are redundant?
  - Do you think any new activities should be included?
  - Do you think the activities are appropriate (e.g., Should a particular topic be practiced / taught using a different approach?)
- What are your thoughts regarding the videos?
  - Do you see any issues regarding the video and principles of CTML?
  - Is there enough / not enough information?
  - Overall quality?
  - Any recommendations for improvement?
- What are your thoughts on the guiding questions for the student interviews?
  - Are there any redundant questions?
  - Are there enough questions to help me understand their levels of:
    - Satisfaction?
    - Engagement?
    - Comprehension?
Student Number:

Time:

Date:

Location:

Opening Description:
- The purpose of this interview is to help me better understand your thoughts about the unit that we just completed on arrays in a flipped classroom environment. As a reminder, a flipped classroom consists of two parts: active learning inside the classroom, and computer-based instruction outside the classroom.
- This is not an evaluation of me as a teacher or you as a student. Instead, my goal is to develop the most effective unit possible that is both engaging and satisfying. I’m doing this both for my own use with my future students and for other AP computer science teachers that are either interested in either flipping their classroom or are new to the field of computer science. I would like to take the information that I collected throughout the unit, as well as your responses to the interview to further develop this unit.
- This interview will be recorded and later transcribed so that it can be analyzed. After I finish analyzing it, I will provide you with a copy of my findings and give you a week to read it over to check to make sure I accurately understood and captured your thoughts. You can either make comments directly on the document and/or meet with me to discuss any comments or questions you have. I will remove anything that you are uncomfortable with, and/or make edits to anything that I misunderstood.
- All of your responses will be confidential, so no one will be able to identify you.
- Feel free to ask any questions throughout the interview.
- If there are any questions you would prefer not to answer, that is okay can let me know that you would rather not answer.
- Do you have any questions before we begin?

Specific Activities
- Tell me about your experiences with / thoughts on:
  - Slack.
  - posting your required questions to the videos.
  - discussing students' questions from the videos in class.
  - generating your own exercises.
  - solving other students' exercises.
  - solving the book exercises in class.
  - having your code analyzed in front of everyone.
  - critiquing your classmates' code in front of everyone.
o pair-programming.
o CodingBat.
o Socrative.
o algorithm tracing.
o comment first coding (pseudocode).
o the videos you watched at home.
  ▪ What did you think about the embedded exercises?
  ▪ How about the duration?

The Flipped Environment
• As a reminder, the goal of this study is to develop an effective unit that is both engaging and satisfying. How would you define
  o Effective?
  o Engaging?
  o Satisfying?
• What are your thoughts on this flipped unit as compared to previous units in this class in terms of the environment?
• In terms of specific exercises and/or activities, what component of this unit would you say:
  o was the most engaging? Least engaging?
  o was the most satisfying? Least satisfying?
  o was the most effective? Least effective?
• At one point, I had you vote for the best solution, what are your thoughts on that?
• What are your thoughts on the quantity of exercises?
• What do you see as the benefits/downsides of the various activities? Including: talking as a class vs activity solving problems in pairs / individually

Engagement - Distractions
• At times, I noticed students looking down at their computers when there didn’t seem to be a reason to be doing so. Was that you at any time? If so:
  o How often did you find this happening?
• How about distractions at home?

Engagement - Optional Activities
• Did you go back to re-watch any of the videos after they were assigned? Why or why not?
• Did you watch any of the optional videos? Why / why not?
• Did you use any other resources? Tutors, Khan Academy, etc.
• Did you work on anything outside of class with other students? If, so:
  o With whom?
  o How often?
  o Do you remember what you worked on with them?
• Did you complete any of the optional exercises?
  o If so,
    ▪ What motivated you to do them?
    ▪ How many did you do?
• Did you find them to be worthwhile?
  o If not:
    ▪ Why not?
    ▪ What would have motivated you to do them?

Community
• What are your thoughts on being randomly paired up with students as compared to selecting your own partner?

Comprehension
• Are there any exercises you can remember that stand out as being extremely difficult?
  o Do you still not know how to solve it (them)?

Overall
• If you could make any changes to the unit, what would they be?
• Moving forward, if we didn’t continue to learn in the flipped classroom environment, are there any components that you like to continue to see implemented? If so, what are they? Why?

Final remarks
• I have no more questions for you. Is there anything else you would like to add?
• As a reminder, I will be sending you my analysis of the interview and you will have an opportunity to check it over to see if you would like anything removed or changed.
LIST OF REFERENCES


Bergmann, J., & Sams, A. (2012). *Flip your classroom reach every student in every class every day*. https://doi.org/10.1111/teth.12165


CollegeBoard. (2016). *AP Computer Science A.*

*Computer Science A Course Description.* (2014). College Board.


BIOGRAPHICAL SKETCH

Shaun Hurley earned a Bachelor of Science degree in physics from the State University of New York, College at Oneonta in 2005. He then earned his Master of Science degree in civil engineering from Florida Atlantic University in 2009. He later earned a Doctor of Education degree in curriculum and instruction with a concentration in educational technology from the University of Florida in 2019. He is an Apple Distinguished Educator and a Florida professionally certified teacher in mathematics and physics. He has been teaching in the classroom for thirteen years and is currently the Mathematics Department Chair at a small independent preK-12 school in the southeastern United States. As of Fall 2019, he will be serving as his school's Academic Technology Coordinator for the upper school.