Middle School Computer Science for Low-SES Schools

By

Rebecca Thomas

A Master's Thesis Presented to the

FACULTY OF THE USC ROSSIER SCHOOL OF EDUCATION

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

MASTER OF EDUCATION

May 2021

https://beccaannthomas.wixsite.com/mysite

Acknowledgments

A big thank you to the professors I have had throughout the Learning Design and Technology masters program. As the whole world has seen over the last few years, there are so many challenges to creating a virtual learning experience and it is through your hard work that I have learned so much. Your constant support and commitment to challenging me to go above and beyond have helped me reach this point. A special thanks to Dr. Kenneth Yates whose guidance helped me bring everything together in this capstone project. Most importantly thank you all for building a wonderful learning community.

Additionally, I would like to thank everyone in my cohort for the role they played in our learning community. Whether it was class discussions, group projects, breakout sessions, or answering questions over slack, you were always there to support me and my work. It was never just about a grade, but instead about learning from each other. Thank you for your genuine interest. There are so many valuable resources and connections that I would not have without you. I will never forget the friends I have made throughout this program.

I would also like to thank my coworkers for putting up with me through everything. It is not easy to work a full time job while completing a masters. You were not just supportive but eager to hear about what I was learning. Another thank you for allowing me to bounce ideas off of you given your contextual knowledge.

Finally, I would like to thank my family and friends. I know I have neglected you while working on my degree. Thank you for putting up with my weird hours even if you had trouble remembering my class schedule. I have spent so much time in front of a computer screen that I look forward to seeing you again in person soon.

Table of Contents

List of Tables

List of Figures

Glossary

Term	Definition
Algorithm	A set of steps to solve a problem
API	An interface that defines a set of supported functions for a piece of code
Array	An ordered collection of elements
Breakpoint	A marker that pauses running code at a particular point for debugging
Bug	An error in a program that results in incorrect behavior
Bug Tracker	A tool for tracking bugs that have not yet been fixed
Build Error	An error that surfaces when building a program before it is run
Class	A piece of code that defines a type of object with set properties and functions
Conditional	A statement that allows a program to execute different code based on a condition
Counter	A tool for counting how many times a particular piece of code is executed
Data	Information used by and manipulated in a program; can be a complex object or a primitive type
Data Structure	An object used to organize data
Debug	The process of finding the cause of a bug or error in a program
Debugger	A software tool with features designed to speed up the process of finding an error in a program
Diff	A line by line comparison between two pieces of data; often used to compare a previous version of a program's code with the current version
Documentation	A formal write up of the behavior and supported functions of a piece of code
Function	A piece of code that takes an optional input and performs a set of actions to produce an optional output
Input	Data passed into a program or function
Linked list	A data structure that stores data in an ordered list where each object points the next object

Logs	Files containing information and errors output by a program while it is run
Loop	A piece of code used to repeat a set of actions
Map	A data structure used to map a key to a value
Null Pointer Exception	A type of error that is thrown when a program attempts to access or perform an action on an object that does not exist
Output	The data that is returned by a program of function
Primitive Types	Types of data that cannot cannot be broken down into smaller types of data
Print Statement	A piece of code that prints information to the program logs
Programming Language	A language used to create machine readable instructions for an algorithm
Queue	A data structure that stores data in an order where the first data in is the first data out
Refactor	The process of reorganizing the code in a program
Runtime Error	A type of error that is thrown while the program is running
Stack	A data structure that stores data in an order where the last data in is the first data out
Stack Trace	A log that shows the order of functions that were called prior to an error being thrown
Syntax	The rules for how to properly format code in a given programming language
Testing	The process of checking that a program runs properly
Ticket	An entry in an issue tracker describing a bug in a program
Unit Test	A test designed to verify a small piece of code without running the entire program
Variable	A name used to reference a data object in a program
Variable Assignment	The process of setting a variable to point towards a specific data object

Abstract

The number of jobs for computer scientists is expected to grow by 22% by 2030 (U.S. Bureau of Labor Statistics, 2021). The median income for computer scientists is three times the median income for all workers and has potential for reducing the generational wealth gap if students from low-socioeconomic status (low-SES) backgrounds are able to capitalize on this growth. However, only 40% of principals report that their schools offer computer science classes (Gallup, 2016). This curriculum will provide a foundation in computer science for students in low-SES schools and help them recognize it is a viable career option. Designed using the ADDIE model, this course builds off a social constructivist theory of learning. This semester-long, 14 unit course for middle school students is designed to take place in a traditional classroom setting without relying on 1:1 student computers. At the completion of the course, learners will be able to apply computer science principles to solve problems, take a systematic approach when solving problems, and understand the value proposition of their continued engagement in the field of computer science. Students will complete a two part summative assessment involving a practical problem to solve and an essay connecting the principles to a real world problem. Rather than including the implementation for all 14 units, a single lesson on debugging is included as an example. An evaluation plan is included to measure how well the course outcomes are met and provide a framework for stakeholders to understand the benefits of implementing the course at other schools.

Needs Assessment

This course is designed to teach computer science principles to middle school students in low-socioeconomic status (low-SES) schools without relying on computers and internet access. Computer science is a rapidly growing field with many high paying job opportunities that could help reduce the income gap that exists for these students. While a majority of parents, teachers, and principals believe that computer science is an important subject for students to learn, about 40 percent of principals report that their school does not offer any computer science courses (Gallup, 2016). Students from low-SES backgrounds are less likely to have access to computers at home (Wood & Howley, 2012). Additionally, these schools are less likely to have access to high-speed internet and up-to-date technology, like modern programming robots, popular in high-SES schools. As a result of this course, learners will be able to describe the main computer science principles, solve problems using these principles, and discuss future career options that use computer science. The need for instruction is clear from the number of principals who report that their school does not offer any computer science courses. Parents, teachers, and principals have all expressed that there is a need for this instruction through their responses on the Gallup (2016) poll.

A problem model of needs assessment will be used initially to ensure a complete understanding of the barriers preventing low-SES schools from having computer science classes. This will reveal whether there are organizational barriers, like finding a qualified teacher, that need to be addressed together with the instruction that is being designed. In addition to the problem model, an innovation model of needs assessment will be used for this course since there is no existing program for computer science instruction at these schools. The innovation model

focuses on creating new instruction in response to change (Smith & Ragan, 2005). In this case, the expectation that students would be introduced to computer science during their K-12 education is a relatively recent expectation.

Description of the Learning Environment

Learners for this course are middle school students and instruction will take place in a formal classroom. However, a learning environment is made up of many aspects beyond the physical space where instruction takes place, such as the instructors, the technology resources available, the learning practices, and the larger institutional processes (Lombardozzi, 2015). Instructors for this course have prior experience teaching but may not have experience with computer science. Those who do not have experience with computer science may be hesitant about new technology, teaching strategies that differ from other courses they have taught, and unfamiliar concepts. Additionally, the challenges that exist in finding qualified computer science teachers may mean some schools will have teachers who are not interested in computer science teaching the course.

While the schools do not have an existing computer science course, there may be other curricula that the course will need to be aligned with. Some schools or districts may have computer science courses at the high school level that this course will need to prepare students for. There may also be a larger science, technology, engineering, and mathematics (STEM) or science, technology, engineering, art, and mathematics (STEAM) curriculum that the school or district has in place. Most states have some form of the common core standards that the computer science course will need to fit within.

The technology available in the classroom varies significantly from school to school. The target schools for this course have limited access to computers and high-speed internet. While they might have access to a laptop cart or a set of iPads that can be checked out for the classroom, these resources are shared with other classes and not available for use the entire time the class will meet. These computers may be older hardware and unable to run the latest software, or the cost of upgrading may be too expensive. Other elements of the classroom facilities that may vary between schools implementing this course include the class size, beliefs about group work, and whether the teachers consider students to be active learners. A large class size can create challenges finding the physical space needed for group work, and if the equipment is shared, there may not be enough for each student to have their own. Computer science is often considered a hands-on discipline and pair programming is a common activity. If the classroom climate does not promote group work or active learning, these tasks may be unfamiliar to learners.

At the organizational level, schools implementing this course are looking to prepare their students for the future. This means not just preparing students for a job in the future but also for the expectations they will face in high school. While these schools have an interest in offering a computer science program, they must balance it with the other courses needed to prepare students for the future. Some schools may not be able to treat this course as a main discipline and instead have to offer it as a special or elective course. All of these elements of the learning environment require careful consideration throughout the design of this course.

Primary and Supporting Typologies

The learning environment for this course can be classified based on several primary and supporting typologies of learning environments. The primary typologies for this learning environment are that it is high stakes and physical with learning happening synchronously in a formal classroom setting. The environment is formal since the learning goals are set by the instructor and not the students. While students will complete homework assignments outside of class, new knowledge will be presented in class through small and large group discussions making the environment synchronous. Supporting typologies for this environment are its non-adaptivity and mix of open and closed elements which are derived from the primary typologies. The learning environment is non-adaptive because students will be working primarily in groups preventing individual learners from having complete control over the pace and content. There are elements of the learning environment that are both open and closed. The materials (primarily paper based) used in the course are provided in an open format for schools considering implementing the course to freely peruse and adapt. However, the actual implementation of the course is closed because only students enrolled in the school and course are part of the learning community.

Design Elements that Promote Learning

The main goals of the learning environment are to support students in learning computer science principles so that they can apply them in new situations and to affect students' attitudes so that they see computer science as a field where they can succeed. While these goals could be accomplished outside of a formal classroom setting, there is high demand for computer science to be taught in schools and so the design of the learning environment begins with the assumption

that learning will occur in a physical classroom environment with a credentialed instructor. Additionally, using a virtual or blended learning environment would prevent many individuals in the target population from participating since low-SES individuals are less likely to have access to computers and the internet at home (Wood & Howley, 2012). The decision to offer a formal course in a physical environment also impacts the level of synchronization that will occur in the course. The instructor's time is limited and teaching synchronously allows them to work with more students at once. Another benefit of teaching computer science synchronously in a physical classroom is that the instructor can provide immediate feedback to all students at once. Some students may not know how to formulate their questions and could benefit from the answers to others' questions. The ability to provide immediate feedback is one key element that Clark et al. (2010) suggest considering when choosing between physical and virtual environments. In addition to these decisions about where and when learning will take place in space and time, the learning environment has been designed to support particular types of activities.

The tools and materials available to students, the level of collaboration, and the content are all selected to support the learning environment goals. Students will not work directly with computers because they are expensive for schools to purchase and maintain while also adding extraneous load that may distract students from the computer science principles that are being taught. Instead, students will work with paper materials and personal whiteboards to remove distractions and the need to provide instruction on how to use the required software. While the course could be taught with paper alone, personal whiteboards have the advantages of being quickly erasable for activities where students need to keep track of information that is changing like variables in their program. Personal whiteboards create an environment students may

perceive as more fun, increasing their willingness to engage. According to Gunawardena et al. (2018), small and large group collaborative activities encourage analysis, synthesis, and evaluation, which correspond to higher-level cognitive processes on Bloom's Taxonomy (Anderson et al., 2001). The learning environment is designed to be collaborative and use small and large group work to support these types of learning. By working collaboratively, students are able to build a shared understanding and further refine their understanding of the details of computer science principles. Another benefit of collaborative learning is its ability to foster an openness to diversity (Adams Becker et al., 2017). This is an important benefit for this course as many learners may have preconceived notions of what computer scientists look like. Working with a diverse group of students can help expand their view of who can succeed in the field of computer science. Finally, the content of the learning activities will use real-world problems and examples to highlight potential role models.

Constraints and Limitations

One limitation of the learning environment is the conceptual authenticity that is lost by not using computers. Programming is usually done on computers with specific software like Integrated Development Environments (IDEs). To address this limitation, the instructor will have a classroom computer and projector to present software in the cases where a particular digital tool is needed to teach the concept. Another limitation of the environment is physical space in which it occurs. The target schools are low-SES schools and likely do not have money to be able to design a space specifically for this class. Instead, they will reuse existing classroom spaces and may not have the ability to configure the room as effectively for small group work as they would be able to with a newly designed classroom with flexible seating. The small group

activities do not require a particular number of students and can be adapted to fit whatever size of small group the space allows.

Learner Profile

Instructional designers must take the time to understand their learners in depth before beginning to design their instruction (Smith & Ragan, 2005). An analysis of the learner helps design instruction that is relevant to the learner, will keep the learner engaged, and enables the design of equitable instruction. There are many aspects of the learner that an instructional designer can consider when building a profile of their learners. Smith and Ragan (2005) provide a comprehensive list of potential characteristics, but they vary in their utility for instructional design. However, for this course, the learners' cognitive characteristics, motivation, and social characteristics are the focus of the learner profile. A more in-depth discussion of each of these constructs and how they apply to the specific learners in this course follows. It will cover the cognitive characteristics of ability, prior knowledge, developmental level, and visual literacy; the motivation characteristics of value, self-efficacy, and interest; and the social characteristics of the learners' tendency towards collaboration or competition; and potential issues with power, equity, and inclusion.

The target learners for this course are middle school students from low-SES schools. This is the first computer science course these schools will be offering at the middle school level so learners' prior experience with computer science is limited. The levels of these constructs will be determined using student surveys administered prior to the start of the course. Additionally, existing student demographic data will be collected from the schools interested in implementing the course including age, gender, ethnicity, and previous STEM grades. It is important for the

data to be disaggregated by each social characteristic to ensure that no group is being disadvantaged in the instruction and to further improve on the course design.

Demographic Characteristics

The target learners are middle school students at low-SES schools. This means that the expected age of the learners is 10 to 14 years old. The gender breakdown of the students will vary from school to school but it is expected to be approximately equally split between boys and girls. Learners are expected to have diverse racial and ethnic backgrounds as communities of color are more likely to have a lower SES.

Physiological Characteristics

Given the physiological characteristics of the target learners, it is not expected that accommodations are needed for the group as a whole. However, there may be individual learners who need accommodations. Course lessons will need to be designed with auditory and visual accommodations prepared so implementing schools do not have to create original accommodations for individual students with impairments. While the course is designed to not depend on computers, any supplemental activities that use computers must work with assistive technologies for learners with auditory, visual, or motor impairments.

Cognitive Characteristics

As an instructional designer, it is important to understand the cognitive characteristics of the learner prior to designing the instruction (Smith & Ragan, 2005). Failure to do so can lead to the creation of instruction that the learner cannot possibly learn from. Some learners may be confused in this situation while others may be disengaged. While there are many cognitive

characteristics to consider, the focus here is on ability, prior knowledge, developmental level, and visual literacy, which are critical to consider when designing computer science instruction.

Ability. There are many different types of capabilities that impact individuals' ability to engage in instruction. Some of the capabilities discussed by Smith and Ragan (2005) that are relevant for this course are the abilities of the human information processing model, sensory capabilities, and the learners' aptitude for the domain of computer science. Instructional designers must understand the ability of learners so that they can design instruction at the right level of difficulty for the learners. Given the target learners for this course are beginners, the instruction will depend heavily on the use of strong instructional strategies, like scaffolding, to ensure students do not experience cognitive overload. While this course does not specifically target learners with any type of disability, there may be individual students with disabilities who will need alternative accommodations. For example, a student with a visual impairment might be unable to see the software an instructor presents. Additionally, it is expected that learners will be able to easily recognize patterns which is an important skill for many computer science processes that will be covered in the course.

Prior Knowledge. Prior knowledge is existing knowledge that learners have that can guide knowledge construction and allow for more information to be held in working memory (Mayer, 2011). If instructional designers do not take the time to understand the prior knowledge of their learners they could end up creating instruction that covers what the learner already knows or that the learner does not have sufficient prior knowledge to understand. Learners' prior knowledge of computer science will be measured through a pre-assessment for the course described in Appendix A.

Developmental Level. The developmental level of the learners is important to understand to ensure that the concepts being taught can be understood by the learners. Some computer science concepts are more abstract than others and may require learners to be in Piaget's formal operational stage rather than the concrete operational stage. According to Piaget, adolescents enter the formal operational stage around 11 to 15 years of age (Santrock, 2017). If this course is being offered to students in younger middle school grades (e.g. 5th or 6th) the learners are likely to still be in the concrete operational stage while those in older middle school grades (e.g. 7th or 8th) are likely to have entered the formal operational stage. The design of this course will lean towards the older grades.

Visual Literacy. Similar to developmental level, visual literacy will be important for determining what topics to teach and how in depth to go on the topics. Some topics, like data structures, rely heavily on visual literacy to illustrate how different data is stored. Students in this course should be able to process information from graphics like a simple flow chart or decision tree and understand that different graphics can be used to present the same information.

Motivation

Motivation is a critical part of the learner profile because it helps instructional designers understand why learners will engage in the instruction that they design. Elements of motivation impact whether learners will engage with the task, the effort they put in, and the likelihood they will persist at the task (Mayer, 2011; Pajares, 2009). For the purpose of this course, the focus is placed on the interest, value, and self-efficacy beliefs learners have towards learning computer science.

Value. According to Wigfield (2002), value is the relative attractiveness of succeeding or failing on a task. Value is a key piece of motivation for an instructional designer to understand as it represents the reason the learners start and persist on the learning tasks. While individual learners will vary in the intrinsic interest and attainment value they have for this course, the utility and extrinsic value are generally high for the target population. According to Gallup (2015), 90% of students expect to have a job in the future where it is somewhat or very likely they would need to know computer science. This expectation shows that students have a high utility value for computer science. The chance of receiving a good high paying job represents one type of extrinsic value that learners might place on learning computer science.

Self-efficacy. Self-efficacy is defined as the belief one has in their capability to take the actions needed to produce a desired outcome (Bandura, 1997). Understanding the self-efficacy beliefs of the learners is important because it impacts how much effort learners will put in during instruction. Low self-efficacy may cause learners to avoid investing effort in something they see as hopeless. The target learners for this course lack mastery experience because they have not taken computer science courses before and likely lack vicarious experience through role models already in the field of computer science. These factors combined with the social persuasions that signal people like them do not belong in computer science result in an anticipated low self-efficacy for the target population.

Interest. Learners may have an existing interest in computer science based on their previous experience with technology. Students with smartphones are likely to be familiar with various applications and understand that computer science and programming are an important piece of creating those applications.

Social Characteristics

Understanding the social characteristics of the learners is an important first step for instructional designers as they affect what experiences the learners have had and how they approach situations. As Thomas et al. (2002) discuss, it is not sufficient to design instruction in a culturally neutral way, which can cause unintentional harm, instead instructional designers must aim to be culturally sensitive in the instruction they design. For this course, the focus is on the social characteristics of gender, race and ethnicity, socioeconomic status, and the learners tendency towards collaboration or competition.

Tendencies towards Collaboration or Competition. Learners in the course may differ in their tendencies towards collaboration or competition. Learners who tend towards competition may focus on outperforming their peers and be unwilling to ask for help. In computer science, a small error can often cause a great deal of confusion and it is helpful to have peers that can bring new perspectives to the problems. It is also common in computer science education to see learners who tend towards collaboration to step back and let others take over. When designing the instruction, it will be important to ensure learners who tend towards both collaboration and competition are effectively prepared to handle collaborative and individual work.

Potential Issues with Power, Equity, and Inclusion. There are many issues of power, equity, and inclusion that exist in computer science instruction and computer science as a field. Care needs to be taken in the design of this course to not further these issues or introduce new issues. One common issue is the gender representation of the computer science field and the power issues that arise in academic spaces as a result. Women in computer science classes are often outnumbered and temper their participation in classes because they doubt their ability

(Eddy et al., 2014; Micari et al., 2007). The target learners for this course are not just men or women so it is important to ensure students of all genders are able to participate equally. In addition to gender, people of color have been historically underrepresented in computer science (Rooney & Khorram, 2020). The target population contains students from marginalized races and ethnicities that may not have seen role models that look like them in computer science. To keep these students engaged and boost their self-efficacy, it will be important to center real world examples and problems that are relevant to them. Depending on whether schools implement this course as a required course or as an elective it will be important to ensure that the student demographics match those of the school. If the course is an elective and students from particular demographic groups are underrepresented, this could suggest there are biases in the way the course is implemented.

Additionally, there are potential issues around power, equity, and inclusion that arise because the target population for this course is students at low-SES schools. Students who come from families with low SES are less likely to have access to technology at home so some technology and computer science concepts may be less familiar to them than other students (Wood & Howley, 2012). These students are also more likely to have working parents and may have to take on responsibilities outside of academics, limiting the time they are able to spend outside of class.

Task Analysis and Overall Learning Goal

Course Analysis

A learning goal is a broad statement of what the course hopes to teach students at the completion of the course. The overall learning goal for this course is to teach students the basic

principles of computer science, how to apply the principles, and in what situations they can be applied. By showing students how many different situations there are in which computer science principles can be applied, the hope is to increase the value they associate with computer science. According to Gagné's types of learning described by Smith and Ragan (2005), this goal is a mix of declarative knowledge, intellectual skills, and attitudes. Some examples of declarative knowledge learners will need to be able to recall include key terminology like variables, data structures, conditionals, and algorithms. The most common type of learning required to complete this goal is intellectual skills. Learners will need to understand many defined concepts (e.g. what items fall under each of those key terms), principles (e.g. variable assignment), procedures (e.g. testing and debugging), and problem solving. Finally, the goal also covers attitudes by attempting to increase learners' motivation by building an understanding of the situations in which computer science can be applied.

Course Outcomes

Course outcomes represent the changes in student behavior that are expected as a result of the student completing the course. For this course, the outcomes are that students will:

1. Apply computer science principles to solve problems in future classes and real world situations (Intellectual skills).
2. Take a systematic approach when solving unfamiliar problems or encountering unexpected results from a solution they have designed (Cognitive strategies).
3. Continue to engage with the field of computer science either by continuing their learning or becoming employed in the technology industry (Attitudes).

Course Objectives

The learning goal for this course is to teach students the basics of computer science, how and when to use different concepts, and build the motivational ideas that will encourage learners to continue in the field of computer science. The learning objectives stated in this section represent more specific objectives that work together to accomplish the overall learning goal. Each learning objective is categorized based on its knowledge dimension and cognitive process dimension according to Anderson and Krathwohl's Revised Bloom's Taxonomy (Anderson et al., 2001).

By the end of the course, learners will be able to:

1. Differentiate between common computer science principles (eg. loops, conditionals, data structures, variables; Conceptual, Analyze).
2. Describe when common computer science principles (eg. loops, conditionals, data structures, variables) might be used (Conceptual, Analyze).
3. Write code that makes use of the computer science principles taught in this course to solve a computer science problem (Procedural, Create).
4. Propose a solution to a real world problem they care about that could benefit from computer science (Procedural, Create).
5. Explain in a journal entry their ability to navigate between the designing, implementing, testing, and debugging phases when solving a computer science problem (Metacognitive, Evaluate).

Major Tasks

The exact tasks needed to solve a computer science problem will depend on the problem that is being solved and not all problems will require all tasks to be completed. Additionally, the tasks do not need to be completed sequentially, but learners can move between tasks and repeat tasks as needed. The major tasks that will be taught in this course follow in roughly the order in which they are most commonly performed:

1. Understanding the problem scope
2. Recognizing what data is involved and the format
3. Designing an algorithm for solving a problem
4. Creating a series of steps to implement the designed algorithm
5. Using conditionals to choose between steps based on a condition
6. Using loops or functions to perform repeated steps
7. Testing the program steps
8. **Debugging issues with the program**

The tasks are derived from my experience as a software engineer. They represent key processes involved in all successful software development. Additional tasks have been added around integrating key concepts (eg. loops and conditionals) that experts will perform as part of the other tasks, but which novices can benefit from focusing on separately. These tasks are used as a foundation when creating the units for this course. Rather than discuss all tasks and units in detail, this paper will focus on Task 8 in future sections.

Unit Overview

The units in this course build directly from the major tasks and course outcomes. The sequencing of the units follows the Elaboration Model (Reigeluth, 1992) beginning with an initial example introducing topics that will be covered in more depth later in the course (Smith & Ragan, 2005). The first unit will introduce a simplified real world problem as an epitome and preview key concepts like data structures, testing, and debugging. Then the next two units will cover key principles and problem solving skills that are generally applicable to all types of problems. The remaining units will go into more depth on topics previewed in the first lesson and are ordered based on their increasing complexity and frequency of use. For example, the units on conditionals and loops presented after the first data unit, instead of grouping all data units together, because conditionals and loops are more frequently used and allow students to solve more types of problems. By structuring the units in this way, students are presented with the basic knowledge they need to be able to solve problems in early units and additional knowledge for more complex problems just in time. Major tasks are generally covered in a single lesson, however there are enough major data structures to cover that it requires splitting into multiple units.

List of Units

Following the structure described in the previous section, the units are listed below:

1. Defining the problem scope
2. Algorithms
3. Testing
4. Data - Variables & primitive types

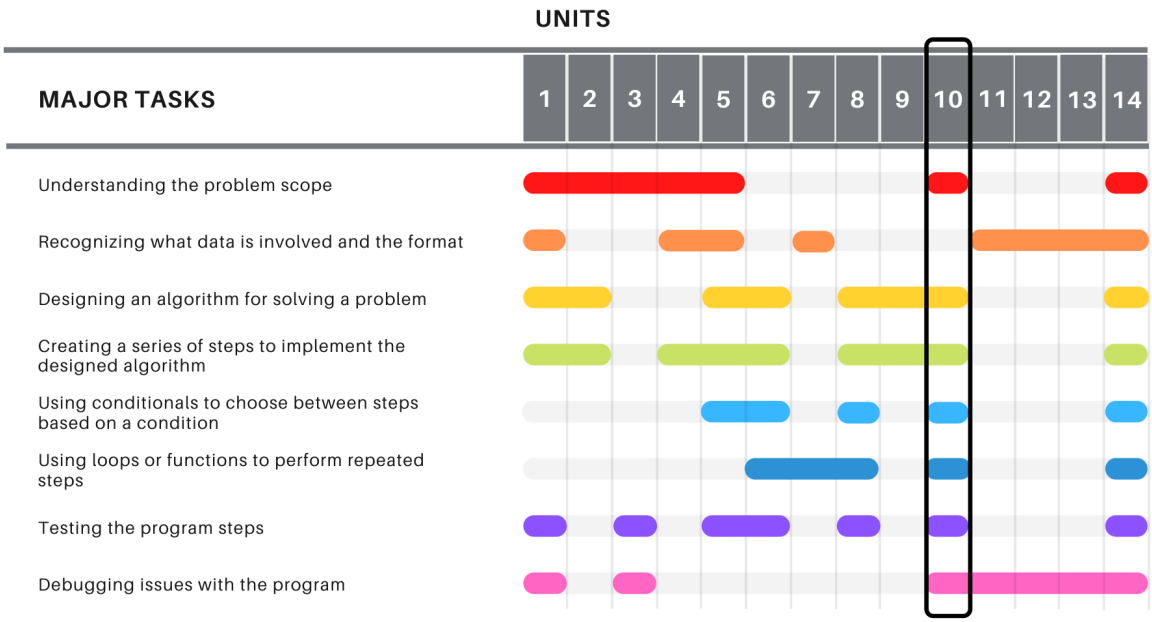
5. Conditionals
6. Loops
7. Data - Arrays
8. Functions
9. Classes
10. Debugging
11. Data - Linked lists
12. Data - Maps
13. Data - Stacks & queues
14. Real world problems

Visual Overview of the Units

A visual representation of the course units is provided in Figure 1. Each major task is identified with a different color marking which units contribute to students' mastery of the task. The following sections of this paper focus on a single example lesson for clarity which is highlighted in Figure 1.

Figure 1

Visual Representation of Course Units



Scope and Sequence Table

The scope and sequence of the units is presented in Table 1, indicating whether the skills required for each outcome are previewed (P), introduced (I), reinforced through application and discussion (R), reinforced only through application (r), or mastered (M) during the unit. The unit being designed in this paper is highlighted in gray within Table 1. This is a beginner course and learners are expected to reach a certain level of mastery for all tasks but can continue to develop their experience with each task beyond this course.

Table 1

Scope and Sequence

	Units													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Outcome 1: Apply computer science principles to solve problems in future classes and real world situations														
Describing the problem scope	I	R	R	R	R	r	r	r	r	R	r	r	r	M
Recognizing what data is involved and the format	P			I	R	r	R	r	r	R	R	R	R	M
Designing an algorithm for solving a problem	P	I	R	r	R	R	r	R	R	R	r	r	r	M
Creating a series of steps to implement the designed algorithm	P	I	r	R	R	R	r	R	R	R	r	r	r	M
Using conditionals to choose between steps based on a condition					I	R	r	R	r	R	r	r	r	M
Using loops or functions to perform repeated steps						I	R	R	r	R	r	r	r	M
Outcome 2: Take a systematic approach when solving unfamiliar problems or encountering unexpected results from a solution they have designed														
Testing the program steps	P		I	r	R	R	r	R	r	R	r	r	r	M
Debugging issues with the program	P		P							I	R	R	R	M
Outcome 3: Continue to engage with the field of computer science either by continuing their learning or becoming employed in the technology industry	I	R	R	r	r	r	r	r	r	R	r	r	r	M

Note. P = Previewed; I = Introduced; R = Reinforced (discussion & application); r = reinforced (application); M = Mastery

Lesson Analysis for One Lesson: Debugging

Unit 10: Debugging

Lesson Analysis

This lesson will focus on the skills needed to debug issues in computer programs. When creating or modifying a program in computer science it is rare to have the program work perfectly the first time. This lesson was chosen because it is critical that students have the skills to debug their programs in order to iterate on the design until they have a working program. Using Gagné's (1984) learning outcome types, the learning outcome for this lesson is categorized as problem solving which is a subtype of intellectual skills. In developing this lesson, I completed a cognitive task analysis (CTA) involving an interview with a subject matter expert (SME). The SME interviewed for this cognitive task analysis was Sonny Zylstra, a Staff Software Engineer at Google. Sonny has over ten years of experience as a software engineer and has moved into a managerial role in recent years. As a Tech Lead and a manager, she spends a significant amount of time helping her team debug complex software problems making her an expert at the task. Prior to conducting the CTA, I built on my prior knowledge of debugging through informal conversations with other software engineers and through observing how several problems were debugged.

Terminal Learning Objective

At the conclusion of this lesson, learners will be able to find and determine the cause of a bug in a program. When a program does not work as expected, the ability to identify where a bug

is in the program and the underlying cause is critical to fixing the issue. Those debugging skills will be covered in this lesson. The skills required to fix the bug will be taught in other units.

Cue

There is a bug in the program. That is, the actual result of a program given a set of inputs does not match the expected result.

Condition

For this task to occur, learners must first have a program that they are running or testing. They must have an understanding of what the expected result is of the run or test and this run or test must give an unexpected result. The unexpected result can be either in the form of an unexpected value or an incomplete run or test. The task is completed with two individuals, the primary learner who is helping to debug a program the secondary individual is working on creating. The physical environment should be a work or classroom environment where the secondary individual was working on the problem and the primary individual should be familiar with computer science and present at the time.

Standards

There is no time standard for completing this task as different types of bugs will take inherently different amounts of time to find and determine the cause of. Instead, learners will need to identify the specific step in the program where the bug is occurring or identify someone with additional subject matter knowledge to continue debugging. The task is considered complete when they have identified where the error is occuring in their code, identified what about the code results in the unexpected behavior, and proposed a solution that will result in the expected behavior. In the event that the learner's experience limits their ability to find the error

the task is considered complete after identifying someone with more experience to help debug the error.

Equipment

The equipment required to complete the tasks varies based on the format of the program that contains the error. Learners will need access to the program code which may mean access to a computer in some situations. However, if they are working on a program where the code is not on a computer there is no requirement that they have access to a computer. The strategies and processes of debugging are the same whether performed on a computer or off. This unit will teach the strategies and processes without computers before briefly demonstrating some tools that can be used to decrease the time required to complete the task when the code is on a computer.

Cognitive Task Analysis

The task chosen for this lesson is how to debug an issue with a program. This task is commonly performed individually and with a partner. For this lesson analysis, a CTA was performed with a focus on how to debug a problem in a program or piece of code with a partner. Ideally, a cognitive task analysis would be performed with an expert with similar characteristics to the target learner. However, given the limited access to computer science courses in schools, there are not many middle school students who are experts at debugging programs. Instead, an interview was conducted with Sonny Zylstra, a software engineer and subject matter expert (SME). The CTA is presented from the perspective of the SME who was interviewed. The next steps for this CTA would be to translate it to an appropriate level of complexity for middle school students and present it to a sample of middle school students to identify any potential gaps

in understanding that need to be addressed (e.g. terminology to be defined). The steps of the CTA follow with action steps listed with numbers and decision steps listed with letters. A visual representation of the CTA is also provided in Figures 2, 3, 4, and 5. Unless otherwise noted, all decision steps should be completed in order and individuals should skip to the next decision step if the criteria for the step does not apply.

1. **Define** the scope and context of the program or code including details about what the program should do, what language the program is written in, are there any specific technologies or tools being used, and are they modifying an existing program or writing a new one.
    a. **IF** the scope of the program is broad, **THEN** narrow it down to what piece of the program they are working on before moving on to the next step.
    b. **IF** you have experience with the context, **THEN** move on to step #2.
    c. **IF** you do not have experience with the context, **THEN** ask questions about the context until you feel comfortable moving on to step #2.
2. **Ask** questions about the problem. Some example questions are: How much do you know about the problem? What is the expected result? What is the actual result? Do you know where the problem is happening? When was the last time it worked?
    a. **IF** you are not comfortable with the language or technology, **THEN** help develop questions to ask an expert with more contextual experience and hand off the problem. In this case, you are done performing the task.
    b. **IF** the problem is in existing code, **THEN** evaluate the cost-benefit ratio of fixing the problem and decide whether to continue to the next step (potentially pulling in

more people). How bad is the problem? Is it affecting real users? How many users? If the problem is not worth solving, you are done performing the task.

c. **IF** there is an error message, **THEN** move on to step #3.

d. **IF** it is a rare problem that only happens sometimes, **THEN** determine what about the situation could be unique (eg. action steps before the error or something about the user).

e. **IF** you feel comfortable with your understanding of the problem, **THEN** move on to step #4.

3. **Identify** context clues in the error message.

    a. **IF** the error message is specific and you have seen it before, **THEN** offer guidance based on past experience and move on to step #9.

    b. **IF** it is a common error message (eg. null pointer exception), **THEN** discuss common mistakes that can generate that error message and move on to step #9.

    c. **IF** it is a build error, **THEN** determine if there is something missing or incorrect syntax (eg. typo, missing semicolon, wrong number of method parameters) and move on to step #9.

    d. **IF** there is a line number and file name, **THEN** take a look at the specific line of code and move on to step #9.

    e. **IF** there is a stack trace, **THEN** start at the most specific piece and work out until you find a line you are familiar with (eg. function name) then move back in looking for additional context clues (eg. the type of variable there is a problem with). Move on to step #9.

   f. **IF** the error message is not specific, **THEN** return to step #2.

4. **Ask** what has already been done to try and find the error.
   a. **IF** they have not done something you think could be helpful (eg. checking documentation, looking up how the API is meant to be used, finding existing examples, or looking at the program logs), **THEN** direct them to those resources and allow them to explore on their own if they no longer need help. If they do not need help, you are done performing the task.
   b. **IF** they need help with the strategy, **THEN** demonstrate how to use the strategy and move to step #3 if it produces an error message.
   c. **IF** they still need help, **THEN** move to step #5.
5. **Narrow down** where in the program the problem is occurring.
   a. **IF** it is a new failure in code you are updating, **THEN** look at what has been changed since the last successful run and see if anything stands out as incorrect. Move on to step #9 if something looks incorrect.
   b. **IF** it is a multi step program, **THEN** run sub components until you find one that has an unexpected output and move on to step #7.
   c. **IF** the language has good debugging tools (eg. android development), **THEN** attach a debugger, set breakpoints, and move on to step #7.
   d. **IF** you cannot use a debugger, **THEN** use another method to find intermediate outputs (eg. print statements, logs, or counters). If you are out of strategies move on to step #6.
6. **Run** at least one test.

   a. **IF** there is an existing test that can be run and fails, **THEN** return to step #3.
   b. **IF** there is an existing test that passes but the code fails, **THEN** move on to step #8.
   c. **IF** there is no existing test and one can be easily created, **THEN** add a new test and return to step #3.
   d. **IF** there is no existing test and one cannot be easily added, **THEN** move to step #7.

7. **Move** through the program line by line and check variables.
   a. **IF** the values of the variables are expected, **THEN** move on to the next line and repeat this step.
   b. **IF** the values of the variables are unexpected, **THEN** move on to step #9.
   c. **IF** you step through all the lines and do not find anything unexpected, **THEN** consider pulling in another person for additional eyes and experience and return to to step #5.
8. **Verify** the data.
   a. **IF** you ran a test that passed, **THEN** compare the format of the actual data and test data. If the data format does not match move on to step #9. If the data does match, identify someone with more experience to take over the task of helping debug. After identifying someone with more experience, the task is considered complete and the new individual will take over.
   b. **IF** the program outputs all the same values but the input data is not all the same, **THEN** return to step #5.

9. **Design** a fix.
    a. **IF** there is a coding error (eg. typo or incorrect loop), **THEN** update the identified line and move to step #10. This is the most common type of error.
    b. **IF** a piece of code is being used incorrectly, **THEN** update the usage to match the intended use. Consult the documentation if needed before moving on to step #10.
    c. **IF** the data format is incorrect, **THEN** reformat the input data and move on to step #10 with properly formatted data.
    d. **IF** the program steps do not match the desired algorithm, **THEN** rewrite the code to match the algorithm and move on to step #10.
    e. **IF** it is unclear how to fix the issue, **THEN** identify someone with more experience to take over helping to debug. After identifying someone with more experience, the task is considered complete and the new individual will take over.
10. **Test** the fix.
    a. **IF** the program still does not work, **THEN** return to step #2.
    b. **IF** there was a test that should have caught the issue, **THEN** update the test to cover the conditions of the problem. At this point you have completed the task.
    c. **IF** the code is easily tested, **THEN** add a new test to catch the problem in the future. At this point you have completed the task.
    d. **IF** the code is not easily tested, **THEN** add a comment, create a ticket in the appropriate bug tracker, or refactor to be more testable in the future. At this point you have completed the task.

Figure 2

Cognitive Task Analysis Flowchart

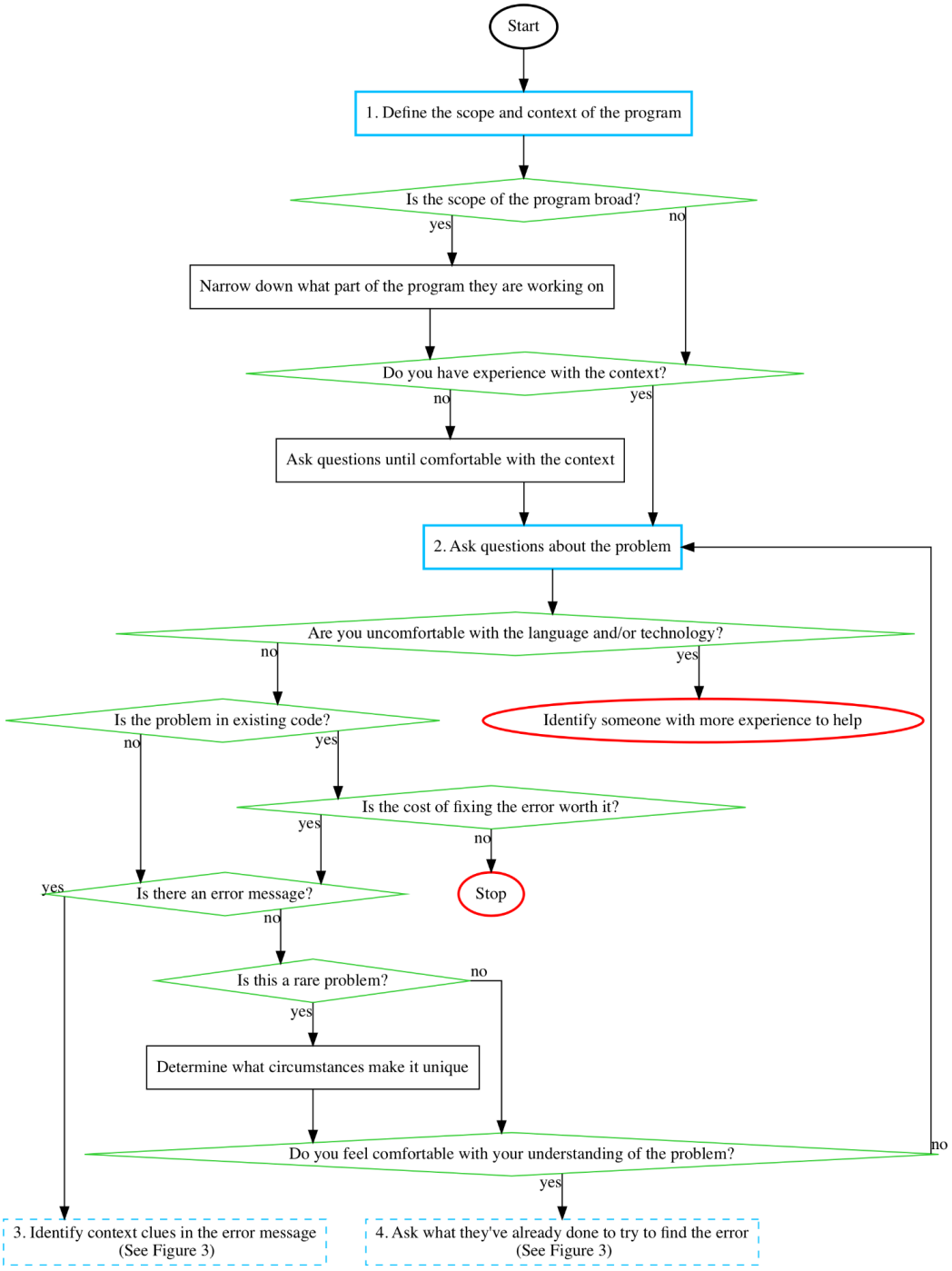


Figure 3

Cognitive Task Analysis Flowchart

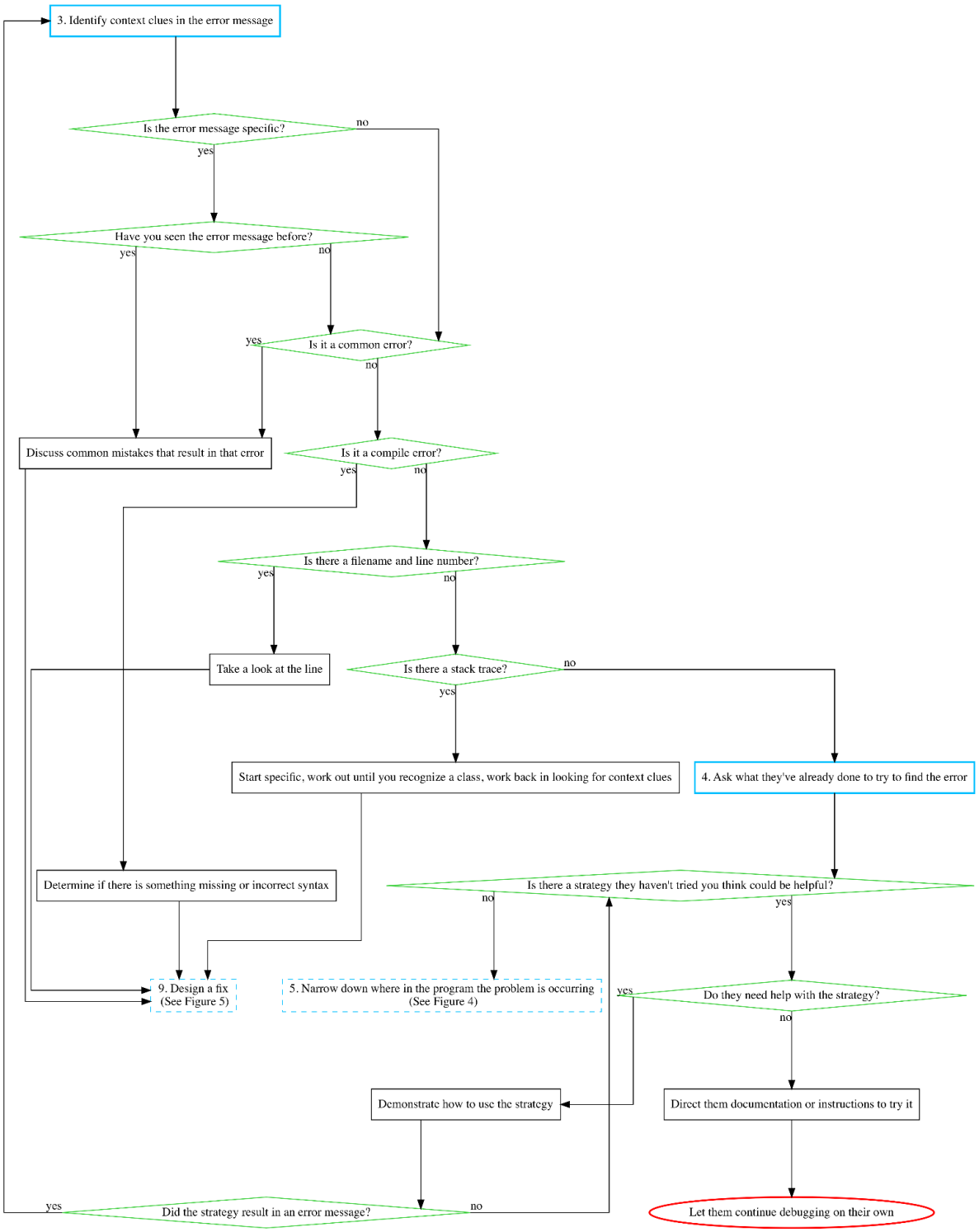


Figure 4

Cognitive Task Analysis Flowchart

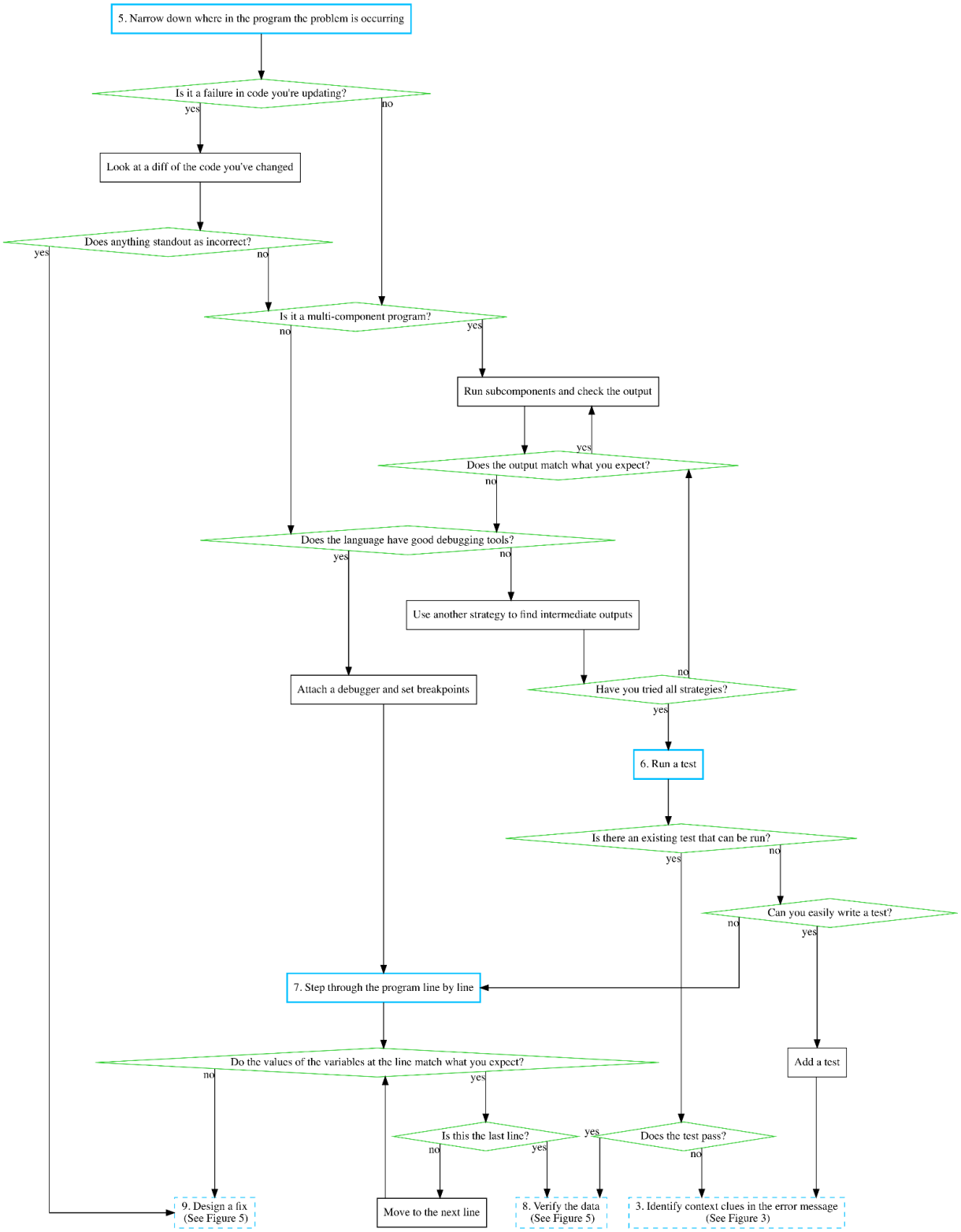
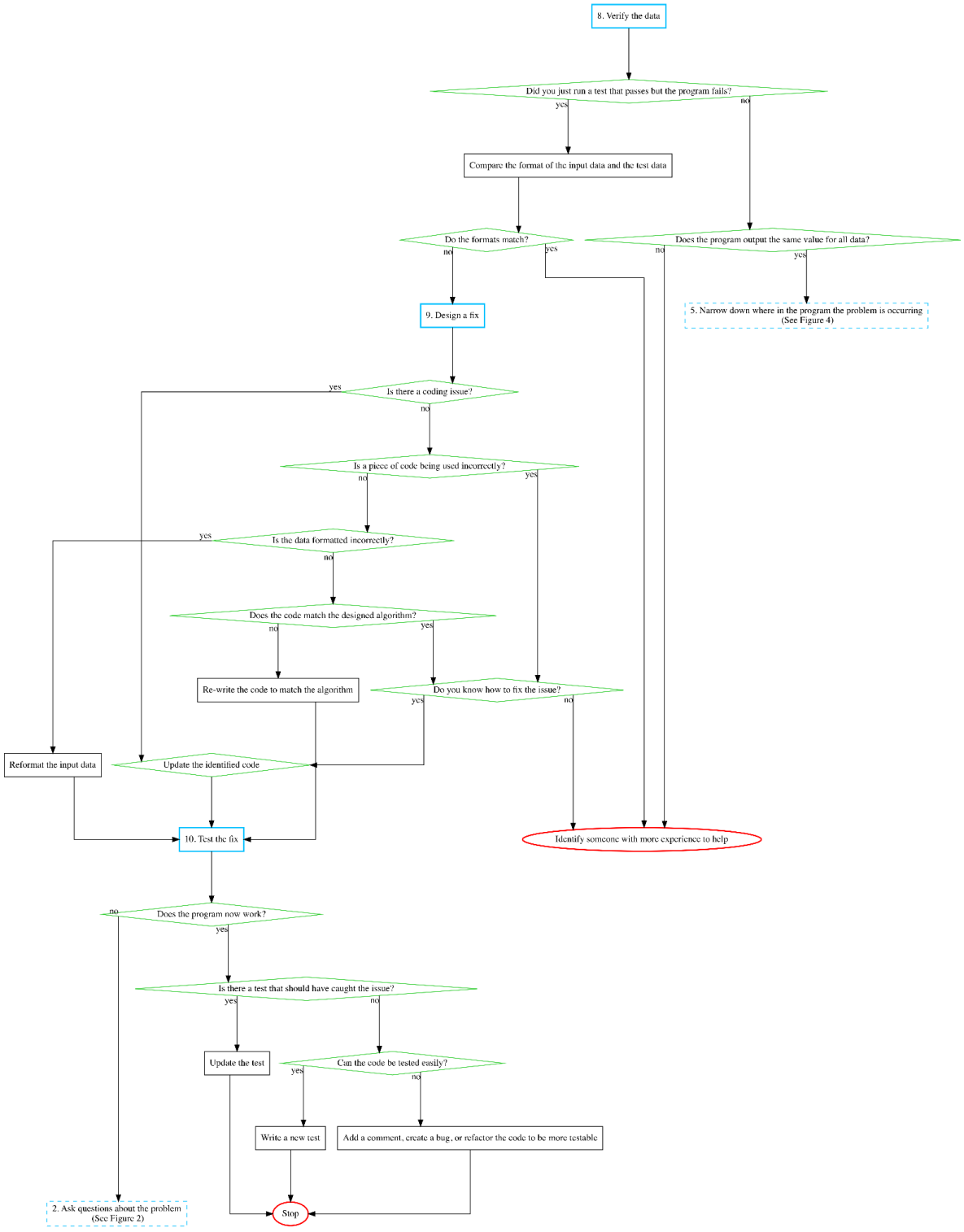


Figure 5

Cognitive Task Analysis Flowchart



Prerequisite Analysis

Given the complexity of the chosen task, there is a significant amount of prerequisite knowledge that the learner must have in order to complete the task. Each action step requires the learner to have some prerequisite knowledge to complete the described action. A complete list of this knowledge follows with the classification of the knowledge according to Anderson and Krathwohl's (2001) matrix:

1. **Define** the scope and context of the program or code including details about what the program should do, what language the program is written in, are there any specific technologies or tools being used, and is it existing code that is being modified or new code.
    a. ***Define*** what a programming language is. (Remember)
    b. ***Define*** what an algorithm is. (Remember)
    c. ***Explain*** how different programming languages impact the design of algorithms. (Understand)
    d. ***Differentiate*** between different types of programs and the types of problems they can produce. (Analyze)
    e. ***Explain*** how new code and edited code can produce different types of problems. (Understand)
    f. ***Explain*** that computer science knowledge is context specific and their knowledge may not be helpful in all situations. (Understand).

2. **Ask** questions about the problem. Some example questions are: How much do you know about the problem? What is the expected result? What is the actual result? Do you know where the problem is happening? When was the last time it worked?
    a. ***List*** five example questions that can reveal different elements of a problem. (Remember)
    b. ***Compare*** expected results and actual results of a program. (Apply)
    c. ***Describe*** how build errors, runtime errors, and coding or design errors result in different types of actual results. (Understand)
    d. ***Explain*** how the last time a program ran successfully can be used to narrow down when an error was introduced. (Understand)
    e. ***Explain*** that programs can be made up of multiple subparts. (Understand)
3. **Identify** context clues in the error message.
    a. ***Describe*** what types of error messages a program can output. (Understand)
        i. ***Define*** what a build error is. (Remember)
        ii. ***Define*** what a runtime error is. (Remember)
        iii. ***Define*** what a null pointer exception is. (Remember)
    b. ***Find*** an error message in a failing program. (Analyze)
        i. ***Find*** where the log files are when running a program. (Apply)
        ii. ***Find*** where a program outputs a build error. (Apply)
        iii. ***Describe*** the various levels of severity used in logging. (Understand)
    c. ***Identify*** what different context clues can look like. (Understand)
        i. ***Define*** what a stack trace is. (Remember)

ii. ***Demonstrate*** how to identify files and line numbers in a stack trace. (Analyze)

d. ***Explain*** how context clues relate to the code. (Understand)

4. **Ask** what has already been done to try and find the error.
    a. ***Explain*** how different strategies can be used to find errors. (Understand)
    b. ***Compare*** debugging strategies depending on the context of the problem. (Evaluate)
    c. ***Summarize*** how documentation can be used to explain strategies and tools. (Understand)
5. **Narrow down** where in the program the problem is occurring.
    a. ***Explain*** the value of narrowing down where a problem occurs. (Understand)
    b. ***Describe*** how expected and unexpected intermediate values can be used to narrow down where a problem occurs. (Analyze)
    c. ***Define*** what a diff is and ***demonstrate*** how it can be used to narrow down a problem. (Remember & Apply)
    d. ***List*** what common debugging tools exist. (Remember)
6. **Run** at least one test.
    a. ***Demonstrate*** how to run a unit test. (Apply)
7. **Move** through the program line by line and check variables.
    a. ***Demonstrate*** how to use a debugger and set breakpoints. (Apply)
    b. ***Trace*** the order in which lines in a program are executed. (Analyze)
    c. ***Describe*** what variables are and how their values are set. (Understand)

   d. ***Explain*** how loops, conditionals, and methods impact what lines are executed. (Analyze)

8. **Verify** the data.
   a. *[Intentionally omitted as the skills needed for this action step will be the focus of their own lesson.]*
9. **Design** a fix.
   a. *[Intentionally omitted as the skills needed for this action step will be the focus of their own lesson.]*
10. **Test** the fix.
   a. ***Explain*** how tests can be used to catch errors. (Understand)
   b. ***Describe*** the benefit of using tests to prevent an error from occurring again. (Understand)

Learning Objectives for One Lesson

The overall learning goal for this course is to prepare students to apply computer science principles. In order to support this learning goal, there are terminal and enabling learning objectives for each lesson that must also be met. The terminal learning objective for this chosen lesson is to be able to find and identify the cause of a bug in a program. If learners cannot accurately identify and fix bugs in their programs, they will be limited in their ability to apply computer science principles to solve problems. The enabling learning objectives are additional learning objectives that support the completion of this terminal learning objective. The terminal and enabling learning objectives are listed below.

Terminal Learning Objectives

At the conclusion of this lesson, learners will be able to find and determine the cause of a bug in a program.

Objective 1.

Given a program with an error that needs debugging, the learner can ask various questions to define the scope and context of the program until they feel they have sufficient context to debug.

Objective 2.

Given sufficient context on the program scope, the learner can ask questions about the problem until they are able to describe what is expected to happen and what is actually happening.

Objective 3.

Given an error message from the failing program, the learner can identify context clues that correctly point them to where the error is occurring.

Objective 4.

Given they already tried to find the error, the learner will be able to ask questions to determine what strategies have already been tried and suggest new strategies that are relevant to the type of problem that is occurring.

Objective 5.

Given a multistep program with an error, the learner will correctly identify what piece of the program contains the error.

Objective 6.

Given a program that can be easily tested, the learner will be able to run at least one test to completion.

Objective 7.

Given an idea of what part of the program the error occurs in, the learner will be able to move through the part of the program line by line and check variables until they find one that does not match the expected value.

Assessment of Learning Objectives for One Lesson: Debugging

Overview of Approach to Assessment

It is essential that the knowledge of students enrolled in this course be assessed before, during, and after the instruction. Assessment allows instructors and instructional designers to measure the students' performance and evaluate whether the instruction was effective (Smith & Ragan, 2005). For this course, the entry level skills and prior knowledge will primarily be assessed through post assessments on the previous lessons. Any prior knowledge that cannot be assessed through previous assessments will be assessed using a brief, ungraded multiple choice and short answer quiz at the start of the relevant lesson. Post assessments for each lesson will be used to measure the completion of the learning objectives. The post assessments will take two main forms, a knowledge test and a problem solving task. The knowledge test will be used to measure declarative knowledge and the ability to recognize and recall key terms or concepts. The problem solving task will be designed to assess their ability to apply the concepts and principles

covered in the lesson. The following sections are an example of what a single lesson assessment plan will look like using the debugging lesson covered in the previous sections.

Entry Level Skills

Entry level skills are the skills learners need to already have in order to begin the instruction (Smith & Ragan, 2005). For this lesson, the majority of entry level skills needed are skills that will be covered in previous units of this course. As a result, they will not be assessed separately prior to this lesson as the post assessments from those units will be considered sufficient. This includes skills such as understanding the problem scope, understanding what an algorithm is, and running tests on a program. Some example questions from previous lesson post assessments include the following:

1. *Describe what an algorithm is and when it is used.*
2. *List 3 different types of tests that can be run on a program.*

One area that may require assessment beyond what is covered in previous lessons is students' understanding of how their changes can affect a program. To address this the following question will be asked in conjunction with the pre-assessment:

1. *Consider the following scenario. You have a program that correctly adds two numbers and you want to modify it to multiply the numbers. After making changes, the program is not outputting the correct answers. Where is the problem likely occurring?*
    a. The input
    b. The existing code
    c. The modified code
    d. The output

Pre Assessments

A pretest is an important tool for understanding what learners already know about the topic of instruction (Smith & Ragan, 2005). Given this is a beginner level course, it is assumed that learners will not already know most of the prerequisite knowledge and skills identified in the task analysis and they will need to be taught during the lesson. The pre-assessment will not cover all of these prerequisites, and instead will be a broad measure of the content related to the topic of debugging computer programs. The pre-assessment will contain a series of multiple choice and short answer questions. This quiz will be designed to be completed in just a few minutes by the students prior to the lesson and is ungraded. A full version of the pre-assessment can be found in Appendix B and a few example questions follow:

1. *Which of the following is the process of trying to find an error in a program?*
    a. *Coding*
    b. *Programming*
    c. *Debugging*
    d. *Testing*
2. *Code that tries to get the 3rd element in a list that contains only 2 elements is an example of what type of error?*
    a. *Build error*
    b. *Runtime error*
    c. *Data error*
    d. *Design error*

3. *Which of the following is a line by line comparison of what has changed in a program's code?*
    a. *A log file*
    b. *A breakpoint*
    c. *A stack trace*
    d. *A diff*
4. *List 2 questions you could ask to better understand the error that is occuring in a program.*

Retrieval Practice and Self-Regulation

Students need to be able to recall, not just recognize, key concepts from this course in order to apply them in other contexts. One way to help achieve this is by using retrieval practice to help students practice recalling the information throughout the lesson (Karpicke & Grimaldi, 2012). Throughout the lesson, students will be asked to recall different errors they have encountered earlier in the course and asked to make connections between these errors and the concepts that are being taught. Students do not reliably use retrieval strategies when studying on their own (Karpicke & Grimaldi, 2012), but the hope is that having instructors facilitate these practices will encourage students to self-regulate and engage in them on their own. Additionally, the use of retrieval practice during the instruction allows for informal assessment of the objectives that cannot be formally assessed due to time limitations.

Post Assessments

Post assessments are used to determine whether the terminal learning objective and the enabling objectives have been achieved (Smith & Ragan, 2005). It is important to include

enabling objectives in the post assessment in order to understand where students who are unable to achieve the terminal learning objective may need additional support. The post assessment for this lesson will have two parts, a knowledge test containing multiple choice and short answer questions and a problem solving task to verify that students can apply what they have learned. The knowledge test will contain items focused on the declarative knowledge required for the terminal learning objective. A complete version of the post-assessment knowledge test can be found in Appendix B, what follows are a few example items:

1. *What is a stack trace?*
2. *(T or F) A build error is an error that is found before the program runs.*
3. *Which of the following is NOT a debugging strategy?*
    a. *Stepping through a program line by line*
    b. *Using breakpoints to check intermediate values*
    c. *Rewriting the program*
    d. *Running subcomponents of the program*

For the problem solving task for this lesson, students will be given a program with an error and asked to debug the error. They will be expected to describe their understanding of the problem and error, solve the error, and design a test to prevent the error in the future. Students will need to turn in a list of questions they asked and attributes of the error they used to narrow down the problem. The problem to be solved, the assessment rubric, and an example solution are provided in Appendix B.

Anderson and Krathwohl Table

Anderson and Krathwohl created a revised version of Bloom's Taxonomy that can be used to classify learning objectives based on a knowledge dimension and a cognitive process dimension (Anderson et al., 2001). Table 2 contains each of the enabling learning objectives for this lesson classified according to this revised taxonomy. Ideally, each objective would be paired with an assessment, but time and resource limitations only allow for the most critical objectives to be explicitly assessed. The remaining objectives will be informally assessed during the instruction. The assessments for the most critical objectives are also listed in Table 2.

Table 2

Anderson & Krathwohl Taxonomy

Knowledge Dimension	Cognitive Process Dimension					
	Remember	Understand	Apply	Analyze	Evaluate	Create
Factual		O2 A2				
Conceptual				O1 O8		O9 A9
Procedural			O5 O6	O3 A3	O7	O4
Metacognitive						O10 A10

Note: O = Objective; A = Assessment

Learning Activities for Unit 10: Debugging

So far, this design blueprint has presented a thorough analysis of the learner, the learning environment, and the learning task that will be the focus of this course. The learning activities

will all be designed to build off of this analysis. What follows in this section is an overview of the general instructional approach this course will take and an in depth description of the learning activities for the example lesson on debugging. The general approach is drawn from an understanding of the learner and cognitive load theory. The specific approach will detail Gagné's major events of instruction (Smith & Ragan, 2005) and the instructional strategies that will be used during each event.

General Approach

The purpose of this course is to teach low-SES middle school students the basics of computer science. This section describes the instructional strategies that form the foundation of the learning activities that make up this course. The general approach to instruction is based in cognitive load theory. To minimize extraneous load and prevent learners from experiencing cognitive overload, an understanding of the learners' characteristics and prior knowledge is required before designing the learning activities. Many instructional strategies have increased benefits for learners with low prior knowledge compared to those with high prior knowledge. This section will provide a brief refresher of the characteristics of the learners and discuss the instructional strategies used to manage cognitive load and direct instruction.

Learner Characteristics and Prior Knowledge

Given the learners for this course are middle school students, their developmental level will be an important factor in the design of the learning activities. Whether they are in Piaget's formal operational or concrete operational stage will impact their ability to understand the abstract concepts involved in computer science and dictate how much scaffolding these lessons

will require. Additionally, this course is designed as an introductory computer science course and the learners are expected to have low prior knowledge in the domain of computer science.

The learners' lack of experience in computer science will also require significant use of motivational strategies. Learners are likely to have low self-efficacy for computer science related tasks as they do not have experience to build on and may lack realistic role models in the tech industry. Without role models and a limited knowledge of the domain, it is unlikely that learners will already have an intrinsic interest in computer science and understanding of the value of learning computer science.

Cognitive Load Theory

A variety of strategies will be used to manage intrinsic load, reduce extraneous load, and maximize the cognitive load available for germane processes. The lessons and instructional activities will be chunked to create manageable size tasks for the students. Instruction will include pre-training when introducing new concepts and terms prior to the activities where students will apply them. Lessons will begin with a brief review of previous lessons to activate prior knowledge. When teaching procedures, lessons will start with a simplified version of the procedure and use spaced practice to reinforce each step before moving on to the next. Additionally, group work will be used for complex tasks where learners with low prior knowledge can benefit from the knowledge of their peers.

Instructional Strategies

The instruction for this course will use a mix of both supplantive and generative strategies for instruction. It will lean more towards supplantive strategies than generative strategies as the learners are expected to have low prior knowledge and therefore will benefit

more from supplantive strategies than generative strategies. Supplantive strategies will be used to introduce new concepts and principles with more generative strategies used for reinforcement and practice of the application of these concepts and principles in creating algorithms and solving problems. Supplantive strategies will also be used to teach the more complex skills including procedures and problem solving strategies. Generative strategies will be used to build learners' interest in the tasks and value they assign to the tasks. Learners will be encouraged to make connections to real world problems and examples through generative strategies. This will increase their perception of value for computer science and encourage transfer to new problems.

Specific Approach

While the previous section provided an overview of the general instructional strategies and approach for the learning activities, this section will revisit the debugging unit described above. The learning activities for this unit will be primarily supplantive in nature since the learning task is a problem solving task and the learners are beginner learners. Much of the prior knowledge required for this unit (eg. concepts, principles, and cognitive strategies) will be taught in earlier units during this course. The learning activities for this unit will focus on selecting, applying, and sequencing principles to find an error in a program. A lesson protocol focusing on key instructional events is used to organize the learning activities for the lesson. Elements of the lesson are presented based on their main instructional event, but may be interspersed chronologically during the lesson as the instructional events will at times blend together.

Stimulate Motivation

The purpose of this instructional event is to get students engaged and focus their attention on the task that will be covered during the lesson. At this point in the course, learners have likely

encountered errors in their own programs that have stumped them. The instructor will ask learners to describe some of these issues that they have encountered. Having learners make a personal connection here helps stimulate their motivation to learn about debugging similar problems.

Learning Objectives

After stimulating the students' motivation, the instructor will present the purpose of the unit. The instructor will not state all of the enabling learning objectives, but rather focus on the terminal learning objective that students will be able to find and determine the cause of a bug in a program. This is an important instructional event because it helps set a clear goal for the students. Having a clear goal can help students focus their attention throughout the lesson and self-regulate their learning.

Reasons for Learning

Benefits. Building from the example errors students have already seen in the course, the instructor will guide the students in creating a list of benefits of learning debugging strategies. This list should include items such as saving time, writing better code, having a stronger understanding of the code, and being able to work on existing code. Listing out these benefits can further motivate students by increasing the utility value of the skills covered in the lesson.

Risks Avoided. Similar to the benefits, the instructor will guide the students in creating a list of risks that can be avoided by using debugging skills. This list should include items such as having recurring errors, incorrectly believing an error is fixed, breaking code that is already working, and over complicating code to fix a non-existent error. The risks avoided and benefits

lists should be worked on together so students can add or modify items on the benefits list that are related to the risks avoided.

Overview

The instructor will present an overview of the unit which will be broken down into several sections. The sections include understanding the program purpose, types of errors, narrowing down a problem, error messages, and debugging tools or strategies. For each of these sections, the instructor will stimulate the learners' recall of prior knowledge and help them make connections to organize the new knowledge.

Stimulate Recall of Prior Knowledge. To stimulate the learners' recall of prior knowledge for understanding the program purpose, the instructor will ask students to reflect back on the design unit earlier in the course. They will be prompted to recall what questions they asked about the problem when designing and how they might modify them to ask about an error. The instructor will compile a list of these questions that students can refer back to later.

Present Analogies and Metaphors for New Knowledge. When presenting the types of error messages, the instructor will make use of analogies and metaphors to help learners with new knowledge. For example, the instructor could make connections to a car by explaining that a build error is like a car not starting, maybe there is a broken piece in the ignition, or that a runtime error is like making a wrong turn because something was incorrect in the directions.

Suggest Learning Strategies. The instructor will also present students with various learning strategies they can use throughout the lesson. One strategy is to have learners write down what they know and what they do not know. The instructor will highlight that this strategy can be applied to the problem as a whole or to a sub piece of a problem. Another strategy the

instructor will present is asking clarifying questions. These questions can be generated using the list of things learners do not know from the previous strategy or created on their own. The final strategy the instructor will present is breaking a problem down into smaller pieces. The instructor will provide an example of each of these strategies as they are presented and will further demonstrate them throughout the lesson. The instructor will also provide feedback to students identifying successful use of strategies or opportunities to use strategies throughout the lesson to help students recognize the contexts in which they apply.

Present Information and Examples for Prerequisite Knowledge

During this instructional event, learners will be first introduced to the knowledge they will learn during the lesson. After activating the learners' prior knowledge, the instructor will use supplantive strategies to introduce new terms and types of errors needed to debug problems. They will introduce the names for different types of errors as well as the key features of each type of error. Learners will then be placed in groups and each group will be provided with an example problem. They will use the questions developed during the recall of prior knowledge to build an understanding of their example problem and classify it into a particular type of error. Students will then share back to the class about the error they had, the classification they decided on, and how they made the decision.

Learners will also be presented with examples of error messages. The instructor will identify context clues in several example error messages (eg. a stack trace or build error) and explain each type of context clue (eg. line number, variable type, or filename). The instructor will explain that logs and stack traces have a specific ordering and that errors should be addressed in order. The instructor will also make connections between the error message and the

program they correspond to. For example, if an error message has a file name and line number the instructor will direct students to that file and line in the program. During the practice portion of the lesson, students will be given example error messages and asked to identify context clues on their own.

In addition to the types of errors and the context clues of error messages, information will also be presented about the procedure of stepping through a program line by line. This will be presented by the instructor as a manual activity on a small program, but the instructor will also mention that there are tools that can be used to facilitate the process with larger programs. The instructor will first present that the purpose of the procedure is to identify what line the error is on. The instructor will then demonstrate the steps of writing down variable names and their values as they move through a couple steps of the program. After this brief demonstration, the instructor will guide students through the same process for a separate example during the practice instructional event.

Provide Demonstration

The purpose of this instructional event is to provide learners with a chance to see how the knowledge is applied and the contexts in which it should be applied. The instructor will use slides to present a larger program (eg. multiple parts or functions) with an error and engage the students in a discussion about the structure of the program. Learners will be encouraged to think back to the learning strategies that were presented to them during the overview and asked how they could be used to narrow where a problem is occurring. As a group, the students will work through breaking the problem down into sub pieces, asking questions to clarify the purpose of

each piece, and writing down what they know about each piece. They will then identify which piece the error is coming from.

Provide and Guide Practice

This instructional event is designed to provide students a chance to practice what they have learned and make sure they are able to actually apply the knowledge. This unit is chunked to prevent overwhelming students and as a result, much of the practice will occur interspersed with the previous instructional events. Learners will be given an opportunity to practice asking questions about an error immediately after the key terminology has been presented. They will then have another opportunity to apply this knowledge combined with their knowledge of error messages after the instructor has presented the material on error messages. Finally, they will be given opportunities to apply all these elements and the process of stepping through a program to find an error from start to finish. During the first small group activities, each group of students will be given a description of an error. For the second small group activity, each group will be given a description of an error and an error message. For the third small group activity, groups will be given a description of the error, the program code, and an error message if there is one. Using these materials, they will be asked to practice stepping through the program using their personal whiteboards to keep track of variable values. The students will be placed in groups to practice so that they can ask each other questions and receive peer support in areas where they may need extra support.

Provide Feedback

Providing feedback is an important instructional event because it allows students to adjust their actions to improve their learning. Students who may be doing something wrong can identify

it and stop before it becomes a habit and students who are doing something right can make sure they continue to do it. While students are practicing debugging different errors, the instructor will move from group to group observing and providing feedback on the decisions students are making, potential clues they have missed, and additional strategies they should try to find the error more efficiently. After the groups have practiced, the instructor will bring the students back together and share feedback from the small groups that could be relevant to the entire class.

Conduct Authentic Assessment

To assess students' learning, the students will be given novel problems with similar contexts to solve and assessed on their ability to find the error. The students will be given problems with each common type of error discussed during the lesson. It is important to provide the students with new problems in an environment that resembles the environment where the knowledge will be used. Assessing the students' learning using more traditional methods like a multiple choice test could mean a student has learned the declarative knowledge but not the problem solving skills needed to apply the knowledge.

Enhance Transfer

This is a critical instructional event because students will need to be able to transfer learning to new problems as they solve new computer science problems. If they are only able to apply the knowledge to the problems covered in this lesson they will struggle in future computer science classes and in jobs that require computer science. To enhance transfer from this lesson, the instructor will conduct a review of different debugging strategies and in which situations they can be applied. By reviewing these situational cues, the hope is to make it easier for students to recognize transfer situations.

Big Ideas

After reviewing the different debugging strategies, students will be given an index card or piece of paper and asked to write down big ideas they want to remember the next time they encounter an error. In addition to writing down their own items, students will be asked to share their ideas to create a classroom list and to give their classmates ideas to include on their cards. These cards will be saved and referenced by the students in later lessons. The instructor should encourage students to check these cards in future classes to reinforce potential transfer situations.

Advance Organizer for the Next Unit

Finally, the instructor will end the class by providing an advance organizer for the next unit. They will begin by explaining that the next three units will shift focus back to data structures covering more complex data structures including linked lists, maps, and stacks and queues. For the first of these units, linked lists, the instructor will go into more depth explaining that the unit will cover the following topics:

1. Definition of a linked list
2. Comparison to arrays (covered in unit 7)
3. Types of linked lists (e.g. single linked list, doubly linked list, circular linked list)
4. Advantages and disadvantages of linked lists
5. When to use linked lists

Learning Activities Table for Unit 10: Debugging

The learning activities described in the previous sections can be found in Table 3 broken down based on the actions of the instructor and the actions of the learner. Ideally the entire unit would be taught in a single lesson but many middle school classes are under an hour to prevent

students from disengaging in long class periods. To accommodate shorter class periods the unit can be taught in a series of class periods over the course of a week. For example, Table 3 presents the activity timing for a 40 minute class occuring on Monday (M), Wednesday (W), and Friday (F). Most of the instructional events need to occur in each class period so it is not sufficient to simply pick up where things left off during the previous class session. Instead, the instructional events of teaching prerequisite knowledge, demonstrating procedures, and providing practice are shortened with a different topic (e.g. error types) or strategy (e.g. using context clues or breaking down a program) covered during each class session.

Table 3

Learning Activities for Unit 10: Debugging

Instructional Sequence	Minutes	Specific Learning Activity Description	Instructor Action/Decision (Supplantive)	Learner Action/Decision (Generative)
Stimulate Motivation	2	**M:** Class discussion of errors and bugs the students have encountered earlier in the course. **W, F:** Review the list created on Monday and add any items as needed.	**M, W, F:** Ask learners what errors they have faced.	**M, W, F:** Recall errors that they encountered in previous lessons.
Learning Objectives	1	**M, W, F:** Present the terminal learning objective for the unit and enabling learning objectives.	**M, W, F:** Share the terminal learning objective that students will be	**M, W, F:** Ask clarifying questions about the terminal learning

			able to find and determine the cause of a bug in a program.	objective if needed.
Reasons for Learning Benefits. Risks Avoided.	2	**M:** Create a list of benefits of learning debugging strategies and a list of risks that are avoided by using debugging strategies effectively. **W, F:** Review the list created on Monday	**M:** Ask the students about benefits and risks. Write the lists out on a whiteboard. If needed prompt students with some examples like saving time. **W, F:** Present the list created on Monday	**M:** Share ideas of benefits and risks. Suggest revisions to the benefits list if ideas come up while creating the risks avoided list. **W, F:** Actively listen and suggest any potential additions
Overview: a. Review/ Recall prior knowledge	2	**M, W, F:** Discussion of what was covered in previous lessons that this one will build off of.	**M, W, F:** Present the sections that will be covered. **M:** Ask students to think back to the design units and the questions they asked to understand the problem they were trying to solve. Ask how these questions might be modified to ask about bugs or errors.	**M, W, F:** Ask clarifying questions as needed. **M:** Recall questions that can be asked when designing a program and share them with the class. Suggest ways the questions could be modified to ask about bugs or errors.
b. Describe what is new	1	**M, W, F:**	**M, W, F:**	**M, W, F:**

(to be learned)		Discussion of what new material will be covered including the different types of errors that exist and strategies that can be used generally and for specific types of errors. **M:** - Program & error scope - Types of errors **W:** - Error messages & context clues **F:** - Narrowing down the problem - Debugging tools & strategies	Present what new material will be covered during the class. **M:** Present an analogy that a program is like a car and there are different types of things that can go wrong when driving. For example, a build error is like your car not starting while a runtime error is like making a wrong turn because the directions are wrong.	Actively listen and ask clarifying questions as needed.
c. Describe/ Employ learning strategies	2	**M, W, F:** Discussion of strategies that can be used throughout the lesson, including asking clarifying questions, writing down what they know, and breaking things down into a smaller problem.	**M, W, F:** Present different learning strategies and provide examples.	**M, W, F:** Listen and think about what learning strategies they already use and which ones they could try.
Teach/process prerequisite knowledge (the "what") (Declarative knowledge, concepts, processes, principles)	8	**M:** Discussion of types of errors and the key attributes to identify them. **W:** Instructor presents example error messages (e.g. stack traces and	**M:** Present definitions and key attributes for each type of error. Present a simple example for each type of error. Answer any questions	**M:** Reference learning aid on the different types or errors, take notes on additional information, and ask clarifying

		build errors) and highlights context clues. **F:** Discussion of larger programs and how it can be helpful to break problems down into smaller problems. Explanation of how testing can be used to narrow down where in a large program an error is occuring. Discussion of how this concept can also be applied at the micro level by stepping through a program line by line.	students might have. **W:** Present example error messages and highlight context clues including file names, line numbers, and variable types. Describe how the different context clues relate to the program code. **F:** Present an example of a large program and discuss the strategy of breaking a problem down to stepping through a subpiece.	questions as needed. **W:** Listen to the instructor presenting examples of context clues and ask clarifying questions as needed. **F:** Listen to how the strategies for narrowing down where the error is occuring relate to different size problems. Ask clarifying questions as needed.
Demonstrate procedures (“how to”) (Procedural knowledge)	5	**M:** Instructor presents example descriptions of errors and works with the group to classify each example. **W:** Instructor presents a practice problem for students to identify context clues as a class. **F:** Demonstration of how to step through a	**M:** Read each example and make connections to key features of error types to classify examples. **W:** Present a sample error message without context clues highlighted and ask students	**M:** Follow along with the instructor using the types of errors handout. **W:** Find and share context clues during the whole class example. **F:** Observe how the instructor steps

		program and track intermediate values to determine where the error is.	to name context clues. **F:** Demonstrate how to step through a program and track intermediate values to determine where the error is.	through the sample program.
Provide practice and feedback/ Practice and evaluate feedback	10	**M:** After definitions and key attributes are presented students will break up into small groups and be given example problems to classify. **W:** Students break up into small groups and work to identify context clues of sample problems. **F:** In small groups, students are given a large program and asked to break the problem into sub pieces and use the strategies previously discussed to narrow down which piece has the error.	**M, W, F:** Provide sample problems for small group work. Observe students working to solve small group activities and provide feedback on what learning strategies they are using well and which ones to try if the learners get stuck.	**M:** During small group work, discuss what type of error they think the example problem is and why. Recall questions from previous lessons to help understand and classify the error. Report back to the class what problem they had, what type of error they classified it as, and how they made the decision. **W:** Identify context clues and discuss with their small group based on the sample problem

				they were provided. **F:** Use strategies covered during the previous discussion to narrow down where the error is occuring and make notes about the different pieces of the program and what they know about each. Use personal whiteboards to write down the values of variables as they step through the example program.
Authentic assessment/ Assess learning	Outside of class	Students will complete a knowledge test at the end of the unit covering the declarative knowledge. They will also be given a problem solving task where they have a novel problem with an error and need to use the skills covered during the lesson to identify the error and propose a fix with a test plan.	Informally assess students during instruction by observing the small group work. Provide students with a novel problem to be used for assessment.	Recall skills covered in the lesson to find the error in the program. Recall skills from the testing unit to design a test plan for the fix. Document what strategies and context clues they used to find the error.
Retention and Transfer	3	**M, W, F:** After each small group activity, students will	**M, W, F:** Ask students to share what cues	**M, W, F:** Recall what strategies they

		share what strategies they used and the cues that helped them decide to use these strategies. This will help identify features of novel problems the strategies can transfer to.	lead to their use of different strategies and provide feedback on additional cues or strategies.	used during small group work and the cues that lead to the use of the strategies. Listen to the cues and strategies used by their classmates and consider which ones they could use in the future.
Big Ideas	3	**M, W, F:** Students create reference guides of big ideas to remember when debugging problems in the future.	**M, W, F:** Provide students with index cards or pieces of paper to create reference guides. Ask learners to think about what big ideas they would want to remember next time they have an error with one of their programs and need to debug it.	**M, W, F:** Recall key concepts they want to remember and write them on their reference guide. Share ideas with the class and incorporate new ideas they like into their reference guide.
Advance Organizer for the Next Unit	1	**F:** Presentation of advance organizer explaining that the next several units will return to discussing different types of data structures with the next unit covering linked lists.	**F:** Presents a quick overview of the units to come and answers any questions students have about what is coming up next.	**F:** Learners listen to what is coming up and ask any questions they might have.
Total Time	40			

Media Selection

Media selection is an important step in instructional design. It is the translation of learning outcomes described in the design into the instruction that learners will engage in. There are many different ways to categorize media and different categories support different cognitive activities. As Mayer (2020) explains, the ability to successfully achieve meaningful learning outcomes, those with strong retention and strong transfer, depends on the learner's cognitive activity during instruction. To ensure the instruction for this course can meet the needs of the learning outcomes, careful evaluation of the delivery media, presentation mode, and sensory modalities of the media will take place prior to selection. The following sections will discuss the media that has been selected for this course and the research behind the decisions that were made.

Media Versus Instructional Methods

According to Clark (1994), media is simply a delivery method for instruction and no single medium can increase learning. Instead, it is the instructional methods that impact the effectiveness of the media that is chosen for instruction. When presented with two or more media that are capable of meeting the instructional strategy needs, Clark suggests choosing the delivery method based on cost and access. Much of the research on instructional media fails to accurately compare media because it compares one media with another using different instructional methods (Clark et al., 2010). There are however several ways the selection of media can impact the success of instruction.

Digital media can limit the use of specific instructional strategies, but emerging technologies can also increase access to certain types of instruction. Digital media can struggle in

situations where there is a need for sensory modes like smell, a high level of context authenticity, or synchronous feedback which can be challenging or impossible to recreate through digital media (Clark et al., 2010). It is a common assumption that the use of digital media is more exciting for students and will increase their motivation to learn, however there is no evidence to support this assertion (Clark et al., 2010). While there are no motivational increases related to digital media use, Salomon (1984) found that the way students perceive the difficulty of different media can impact the amount of effort they invest in learning from it. Students may perceive certain types of digital media as easier and invest less effort in their learning. Emerging technology has created new opportunities to provide access to instruction to more people. Those in remote areas can learn through distance learning and AR/VR capabilities make it easier to create highly contextual instruction. However, focusing too much on emerging technologies can be dangerous. Designing based on the capabilities of the technology (i.e. technology-centered instruction) can result in the creation of instruction that does not use strong instructional methods (i.e. learner-centered instruction) and does not result in meaningful learning (Mayer, 2020). The use of strong, evidence-based instructional methods is critical for activating cognitive processes in the learner.

The Cognitive Theory of Multimedia Learning developed by Mayer (2020) states that individuals learn better when they learn from both pictures and words rather than pictures or words alone. The three main assumptions of the theory are that humans have dual channels for processing information, working memory capacity is limited, and active processing improves learning. There are three main cognitive processes that derive from these assumptions and occur during learning, selecting relevant words and pictures, organizing the selected words and

pictures, and integrating the organized words and pictures with existing knowledge (Mayer, 2020). Careful selection of instructional methods and media will focus on helping learners with these processes to improve learning.

General Instructional Platform Selection in Terms of Affordances

While media selection does not have a direct impact on the cognitive processes like instructional methods, there are several ways that the selection of media can provide additional opportunities for learning. According to Clark et al. (2010), media selection is directly linked to cost, access, and efficiency of learning. Each of these key considerations has been factored into the media selection for this course and will be described in the following sections. This course will take place in a physical classroom. The course is designed to take place in a normal classroom and not in a computer laboratory. Instead of relying on computers, the course will use primarily print based media.

Access

Access to enough computers for all students in the class may be limited as the target schools are low-socioeconomic schools that may not have enough computers or require teachers to reserve computers. The equipment (eg. laptops or high-speed internet) needed to teach this course using digital media, especially in an online or blended format, would likely be prohibitive to many of these schools. The use of print based media also allows for increased flexibility in the design of group activities that would otherwise have to dictate group size based on available devices.

Consistency

When thinking about designing algorithms and debugging programs, learners will likely have a variety of questions. Some questions will likely have answers that are relevant to all students even those who may not have known how to formulate the question. Teaching this course in a physical classroom ensures that all students have access to the answers to these questions. Additionally, a physical classroom can make it easier for students to ask questions since they do not need to use a special platform for communicating but can just ask verbally. Teachers will also be able to customize the instruction to the specific group of students more easily than updating a learning management system (LMS).

Cost

The cost of offering online or blended instruction would be too expensive as the target students are not likely to have access to high-speed internet and computers at home. Buying a set of classroom computers and getting high-speed internet would seem to be one-time costs but there are additional maintenance costs in the form of repairs, software or hardware upgrades, and teacher training for the latest upgrades. Schools that share a computer laboratory or a laptop cart would not be able to use these devices for other purposes while this course was taking place. Instead, offering the course in a physical classroom without computers limits the cost to more traditional classroom materials than laptops or computers which are cheaper to purchase.

Specific Instructional Platform Selection in Terms of Restrictions

Most media support most types of instructional methods but there are several key elements of instructional methods that can limit what media are suitable for the instruction. The three considerations described by Clark et al. (2010) are the desired level of conceptual

authenticity, immediate feedback, and sensory modalities. This course will take place in a physical classroom without computers. A brief description of how each consideration relates to this course follows and full details can be found in Table 4.

Table 4

Key Considerations for Media Selection

Key Consideration	Media Considerations
Conceptual Authenticity	● *No use of computers reduces the coding authenticity as most transfer tasks will use computers.* ● *No authenticity loss for design and problem solving tasks which experts often perform through written documents or verbal discussions.* ● *No authenticity loss for key concepts that are abstract and common across languages.*
Immediate Feedback	● *Immediate feedback needed when debugging problems.* ● *A physical classroom is easier for students to show an instructor what they are working on.* ● *Easier to ask questions of instructors and classmates in a physical classroom.*
Special Sensory Requirements	● *No sensory requirements beyond visual and aural.*

Conceptual Authenticity

By not using computers in the classroom, some conceptual authenticity is sacrificed as most transfer situations will use computers. However, this course is designed to be a foundational course with a focus on key concepts and problem solving skills which do not require a computer and are consistent across programming languages. Even experts will use written documents and verbal conversations to design and discuss how they will solve a problem before writing code.

Immediate Feedback

One of the most time consuming pieces of solving computer science problems is debugging a program. While this can be completed in remote learning it is much easier to debug when you can ask direct questions and get immediate feedback from an instructor. The high demand for immediate feedback when solving problems in this course is a major consideration in teaching this course in a physical classroom.

Special Sensory Requirements

There are no special sensory requirements for this course beyond visual and aural and so no special media considerations are made as a result.

Client Preferences or Specific Conditions of the Learning Environment

The client and other stakeholders in this course have a significant impact on the media that will be used in this course. While the instructional methods are the most important piece of the media decision and should not be sacrificed, client and stakeholders preferences should be balanced with cost, access, and efficiency considerations when deciding between different media that are capable of supporting the instructional methods. The client for this course, low-SES

schools, requires more weight to be placed on cost considerations when making the decision. The stakeholders for this course, including students, parents, teachers, and principals, have expressed a desire for computer science to be taught as part of the school curriculum (Gallup, 2016). These stakeholders are likely to have preconceived notions about what a computer science class should look like. They may have expectations that computers should be used extensively with class sessions taking place with each student on a computer to promote contextual authenticity. While the course will primarily not be taught on computers, it will be supplemented with occasional computer-based activities to accommodate the client preferences.

Specific Media Choices

With the decision to teach this course in person with students working primarily without computers in mind, this section will cover the specific mediums to be used during instruction. The primary medium for instruction will be paper-based resources used in handouts and to create objects for instruction as needed. Paper was chosen because it is cost effective, easy to create handouts, and supports key instructional strategies. Paper can serve as a scratchpad during instruction and provide an easy way for students to take notes and engage in organizational strategies. To supplement the paper-based instruction, students will also make use of personal whiteboards for more complex learning tasks like designing an algorithm and debugging an error. These whiteboards will allow students to organize their ideas or track variables throughout the program. While the instruction could be completed directly on paper, whiteboards allow students to erase and update the content to stay more focused on the information they care about and reduce extraneous load associated with managing information from all steps at once. Additionally, these media allow students to move objects around in physical space to aid in

organizing and keeping students engaged. For example, students might have different lines of a program on each whiteboard and move them around to create the correct ordering of the code. Finally, the instructor of the course will use a classroom computer to demonstrate examples and show off software that helps illustrate the topics being discussed. These computers are used to provide conceptual authenticity as most of the coding the students will encounter outside of the course will take place on a computer. More details about the specific media and the benefits they provide can be found in Table 5. The paper resources and student whiteboards allow each student to be directly involved during instruction, but the use of computers by the instructor in this course does not offer the same affordance. This will require the instructor to take special care in their use of computers to keep students engaged during the computer use and limit the computer-based instruction to prevent students from disengaging.

Table 5

Media Choices in Middle School Computer Science for Low-SES Schools

Medium	Purpose	Benefits / Key Factors Influencing Choice
Paper	*The primary medium for instruction. Will be used for handouts, objects for instructional activities, and assessments. Will be heavily used for design tasks.*	• Cost effective and easy to create handouts. • Can be moved in physical space. • Easy for students to store and reference as needed. • Students can easily add notes to paper materials.
Computers	*Will be used by the instructor to supplement the paper-based instruction for increased conceptual authenticity.*	• High conceptual authenticity. • Lots of coding resources and examples available online.

Student Whiteboards	*Students will use the mini whiteboards to solve and debug problems.*	• Erasability allows students to make changes easily. • Can be used to update variable values when manually debugging. • Can be moved around in physical space when working in groups to reorder parts of an algorithm or program.

Implementation of the Curriculum

Throughout the design and implementation of the course it will be important to engage key stakeholders and gather feedback. The implementation will occur in several stages with revisions being made as needed. To start, individual units will be piloted with teachers of existing computer science courses at non-target schools. These first adopters will help identify parts of the unit that are challenging for either the instructor to teach or students to understand. This stage will also provide an opportunity for instructors with existing computer science experience to provide feedback on what they think might be missing. After each unit has been tested and revised, the next stage of implementation will focus on innovator schools in the target demographic (i.e. low-SES with no existing computer science). During this stage, all units of the course will be taught together to identify potential gaps in the overall curriculum and prerequisite knowledge that may not be adequately covered in previous units. Throughout these stages feedback will be collected through surveys and interviews with students, instructors, and administrators. In addition to this formative feedback, student performance data will be collected to assess the ability for the course to meet the course outcomes. More details about the data that will be collected can be found in the Assessment and Evaluation section.

Assessment and Evaluation Plan

Curriculum Purpose, Need, and Outcomes

There is a growing number of jobs that require computer science experience (U.S. Bureau of Labor Statistics, 2021) and a large demand for schools to teach computer science (Gallup, 2015). However, many schools still do not teach any computer science (Gallup, 2016). The demand is even higher for parents who make under $54,000 (Gallup, 2015). The purpose of this course is to teach the basic principles of computer science and help low-SES students recognize that computer science is a field where they can succeed.

Evaluation Framework

To ensure these course outcomes are met and the needs are addressed, it is important to create an evaluation plan prior to the implementation of the course. The plan will address each of the four levels of the New World Kirkpatrick Model (Kirkpatrick & Kirkpatrick, 2019). The New World Kirkpatrick Model builds off Kirkpatrick's original model but reverses the order in which the levels are presented to focus on Level 4, the most important, first since many individuals were getting stuck on Level 1 and never reaching Level 4. Each level is described below with details about the elements of the course that will be assessed at that level and how they will be measured.

Level 4: Results and Leading Indicators

Level 4 of Kirkpatrick's model focuses on the larger organizational problem and whether the course results in the desired outcomes (Kirkpatrick & Kirkpatrick, 2019). Assessment at this level is completed by measuring leading indicators, short term observations that signal the behaviors necessary to reach the outcomes are occurring (Kirkpatrick & Kirkpatrick, 2016).

There are both external and internal outcomes for this course that look to measure student engagement and stakeholder interest in the program. A detailed description of the outcomes and how they will be measured can be found in Table 6.

Table 6

Indicators, Metrics, and Methods for External and Internal Outcomes

Outcome	Metric(s) (Unit of measure)	Method(s) (How measured)
External Outcomes		
Continued student engagement with the field of computer science after the course	Percentage of students majoring in computer science or related fields in higher education	After completion of the course, students will be given an opportunity to join a learning community as mentors for future students. Students involved in the community will be surveyed about their experiences.
	Percentage of students engaging in computer science related programs outside of school (e.g. after school programs, summer camps, etc.)	After completion of the course, students will be given an opportunity to join a learning community as mentors for future students. Students involved in the community will be surveyed about their experiences.
More low-SES schools are able to offer a computer science course for the first time	Number of schools adopting or adapting the course materials	Course resources are openly available, but schools offering the course or using resources are asked to provide contact information.

Positive interest from parents, administrators, and local communities about the program	Number of local media stories discussing the schools that are implementing the course	Analysis of local news stories
Internal Outcomes		
Continued engagement with the field of computer science after the course	Percentage of students enrolling in additional computer science classes prior to graduation	Student transcripts and enrollment metrics
Students apply the concepts and strategies in their other courses	Student performance in other courses	Student grade reports

Level 3: Behavior

Level 3 of Kirkpatrick's model focuses on whether the learning can be applied to additional tasks beyond the learning activity (Kirkpatrick & Kirkpatrick, 2019). To achieve the results described in Level 4, the learners must not just complete the instruction but actually make changes to their behavior (Kirkpatrick & Kirkpatrick, 2016). In addition to being a key component of the Kirkpatrick Model, this is also a key component of many learning theories including Mayer's Theory of Multimedia Learning (Mayer, 2020).

Critical Behaviors Required to Perform the Course Outcomes

Kirkpatrick and Kirkpatrick (2016) suggest that the first step in performing a Level 3 evaluation is to define the critical behaviors that are expected from the learners. These are the behaviors that contribute the most to reaching the course outcomes when performed sufficiently (Kirkpatrick & Kirkpatrick, 2016). Table 7 contains the critical behaviors for this course, how they will be measured, and the frequency of measurement.

Table 7

Critical Behaviors, Metrics, Methods, and Timing for Evaluation

Critical Behavior for Course Outcomes	Metric(s) (Unit of measure)	Method(s) (How measured)	Timing (How often)
1. Apply computer science principles to solve problems	Connections made between computer science concepts and new problems	Observation of group work for mentions of computer science terms (e.g. loops, conditionals, lists, etc.) in other classes	Observed during major group projects
2. Take a systematic approach when solving unfamiliar problems or encountering unexpected results	Number of times students break down a large problem into smaller ones	Observation of class work in future classes	Observed during major group projects
	Number of times students engage with classmates to co-design solutions to problems	Observation of class work in future classes	Observed during major group projects
3. Sign up for future computer science opportunities	Enrollment in future computer science courses or programs	Student course enrollment and surveys of students about external programs	At the start of each semester

Required Drivers

Required drivers are the processes or systems that provide the support and accountability for critical behaviors (Kirkpatrick & Kirkpatrick, 2016). They do this in four major ways, reinforcing, monitoring, encouraging, and rewarding the performance of the critical behaviors that have been identified (Kirkpatrick & Kirkpatrick, 2016). The required drivers to support the critical behaviors for this course are listed in Table 8.

Table 8

Required Drivers to Support Critical Behaviors

Method(s)	Timing	Critical Behaviors Supported
Reinforcing		
Demonstration of problem solving strategies	During the demonstrate procedures event of the instructional activities	2
Constructive feedback on algorithms and problem solving strategies	During group work	1, 2
Encouraging		
Student created advanced organizer for key procedures and topics	During the course	1, 2
Instructor highlighting of potential computer science opportunities (e.g. future courses, external clubs, etc.) students can sign up for	As opportunities arise, with a minimum of 3 presented a semester	3
Coaching and mentoring from previous students involved in the learning community	During the course and after the course as long as the student is involved in the community	1, 2, 3
Rewarding		
Highlight student end of year projects for the entire school	At the end of the course	1,2
Highlight student achievements in external programs	During the course and after the course for students mentors involved in the learning community	3
Monitoring		
Instructor observation of group work and feedback on strategies	During group work	1, 2

Student reflection on collaborative group work and strategy use	During the course after assignments and group work	1, 2
Student interviews to understand how students are using what they have learned in other courses	During the course and each semester after	1, 2, 3

Organizational Support

It is challenging to implement required drivers without proper support so each organization should make a plan to support their implementation (Kirkpatrick & Kirkpatrick, 2016). For this course, work should be done with key stakeholders in each organization to create a custom support plan. However, a suggested support plan follows as a starting point for each organization. Students are expected to implement the critical behaviors in future classes so the instructors of these courses will need to be aware of the behaviors that are expected. It is recommended that the organizations offer a professional development session to introduce the instructors to the behaviors and how to recognize and support them. Schools should also work to create a plan for coaching and mentoring to take place. This is necessary to build a strong learning community between current students and past students. The plan should include information on when and where the mentoring can take place as well as any resources that should be available in these spaces.

Level 2: Learning

Level 2 of Kirkpatrick's model focuses on whether the learner has acquired the knowledge and skills that the learning activity was designed to teach (Kirkpatrick & Kirkpatrick, 2019). While Kirkpatrick's levels are designed with corporate training in mind, we see this

concept show up in instructional design in the form of learning goals and objectives. For this course, Level 2 reflects the knowledge of computer science principles, problem solving strategies, and the connections to real world problems this course is designed to teach.

Terminal Learning Objectives

To ensure evaluation at this level covers all the necessary knowledge the terminal learning objectives from each unit can be used as a starting point. The terminal learning objectives for each unit of this course are:

Unit 1: Given an unfamiliar problem, learners will be able to ask questions about the problem until they are able to describe the problem to someone else.

Unit 2: Given sufficient understanding of a problem, learners will be able to create an algorithm to correctly solve the problem.

Unit 3: Given a problem and a solution, learners will be able to create a set of tests to verify the solution works correctly.

Unit 4: Given a program that uses variables, learners will be able to describe the data stored in the variables and the type of the variables.

Unit 5: Given an algorithm that requires different actions in different cases, learners will be able to create a correct solution that uses conditionals.

Unit 6: Given an algorithm that requires a repeated action, learners will be able to create a correct solution that uses loops.

Unit 7: Given data stored in an array, learners will be able to describe the basic operations of an array, how each operation affects the data, and when they might use an array.

Unit 8: Given a complex multi-step problem, learners will be able to use a function to make the solution easier to understand when read by someone else.

Unit 9: Given a complex problem that uses a lot of data, learners will be able to use a class to simplify the solution and make the program easier to understand when read by someone else.

Unit 10: Given a program with an error, learners will be able to find and determine the cause of the error in the program.

Unit 11: Given data stored in a linked list, learners will be able to describe the basic operations of a linked list, how each operation affects the data, and when they might use a linked list.

Unit 12: Given data stored in a map, learners will be able to describe the basic operations of a map, how each operation affects the data, and when they might use a map.

Unit 13: Given data stored in a stack or queue, learners will be able to describe the basic operations of a stack or queue, how each operation affects the data, and when they might use a stack or queue.

Unit 14: Given a real world problem, learners will be able to propose an algorithm to solve the problem and describe the computer science principles needed to implement the solution.

Components of Learning Evaluation

The primary method for evaluating learning will be through traditional assessments throughout the course. Students will complete knowledge tests and problem solving tasks for each unit. In addition to these traditional assessments, instructors will also be observing small group collaborative work to informally assess learning. Finally, attitude, confidence, and commitment components will be assessed through two summative tasks at the end of the semester and student interviews and surveys taking place one month after the course. Details about how each component of learning will be assessed can be found in Table 9.

Table 9

Evaluation of the Components of Learning for the Program

Method(s) or Activity(ies)	Timing
Declarative Knowledge "I know it."	

End of unit knowledge tests	End of each unit
Observation of small group collaborative work	Throughout the course
End of semester final project	End of the course
Procedural Skills “I can do it right now.”	
End of unit problem solving task	End of each unit
Observation of small group collaborative work	Throughout the course
Attitude “I believe this is worthwhile.”	
End of semester journal reflection	End of course
End of semester real world problem essay	End of course
Confidence “I think I can do it in the future.”	
End of semester journal reflection	End of course
Observation of small group collaborative work	Throughout the course
Pre- and post-course motivational surveys	Before the course and one month after the course
Student interviews after the course	One month after the course
Commitment “I will do it in the future.”	
End of semester journal reflection	End of course
Student interviews after the course	One month after the course

Level 1: Reaction

Level 1 of Kirkpatrick’s model focuses on the motivation of the learner and their reaction to the learning activity (Kirkpatrick & Kirkpatrick, 2019). There are three components of reaction that Kirkpatrick and Kirkpatrick (2016) suggest evaluating - engagement, relevance, and customer satisfaction. Table 10 describes how each of these components will be evaluated for this course.

Table 10

Components to Measure Reactions to the Program

Method(s) or Tool(s)	Timing
Engagement	

Instructor observation during the learning activities	During the course
Student interviews after the course	One month after the course
Relevance	
Discussion with students about how they will use the knowledge	Throughout the course during the stimulate motivation, reasons for learning, and big ideas instructional events
Student interviews after the course	One month after the course
Interviews with past students involved in the learning community to understand how they are using the material in their courses	Each semester after the course
Customer Satisfaction	
Student interviews after the course	One month after the course

Evaluation Tools

Immediately Following the Program Implementation

Immediately following the course, students will be asked to complete a survey to evaluate Level 1 and Level 2. The survey will focus on their reaction to the course and their confidence in using what they have learned. In addition to the student survey, students will complete a final project, a journal reflection, and a real world problem essay to further evaluate Level 2. All of these evaluation tools can be found in Appendix C.

Delayed For A Period After The Program Implementation

After the course, interviews with former students will be regularly conducted to evaluate Kirkpatrick Levels 1, 2, 3, and 4. These interviews contain a mix of knowledge, motivation, and exploratory items. The complete interview protocol can be found in Appendix D.

Data Analysis and Reporting

There are a lot of different stakeholders for this course, and their interests will vary drastically as well. The stakeholders for this course include students in the course, their parents, and instructors teaching the course as well as administrators at the school and district level who

may be interested in expanding the use of the course curriculum. This section describes how the data for the external outcomes for Kirkpatrick's Level 4 will be presented to stakeholders. However, if stakeholders are interested in additional data they can reach out directly to the schools implementing the program or follow up directly to understand the feasibility of getting the desired data. The figures in this section are meant to illustrate how the data will be visually presented and are based on dummy data.

The first outcome is focused on student engagement and will be presented using pie charts like in Figure 6 and Figure 7. The second outcome is focused on the growth of the program and is presented using a line graph as shown in Figure 8. Finally, the third external outcome focuses on the community perception of the program and will be presented as key quotes from the news stories. For example, a news story might have said, "An innovative new computer science course helps set low-SES students up for success. A true game changer!"

Figure 6

Percentage of Students majoring in Computer Science or a Related Field

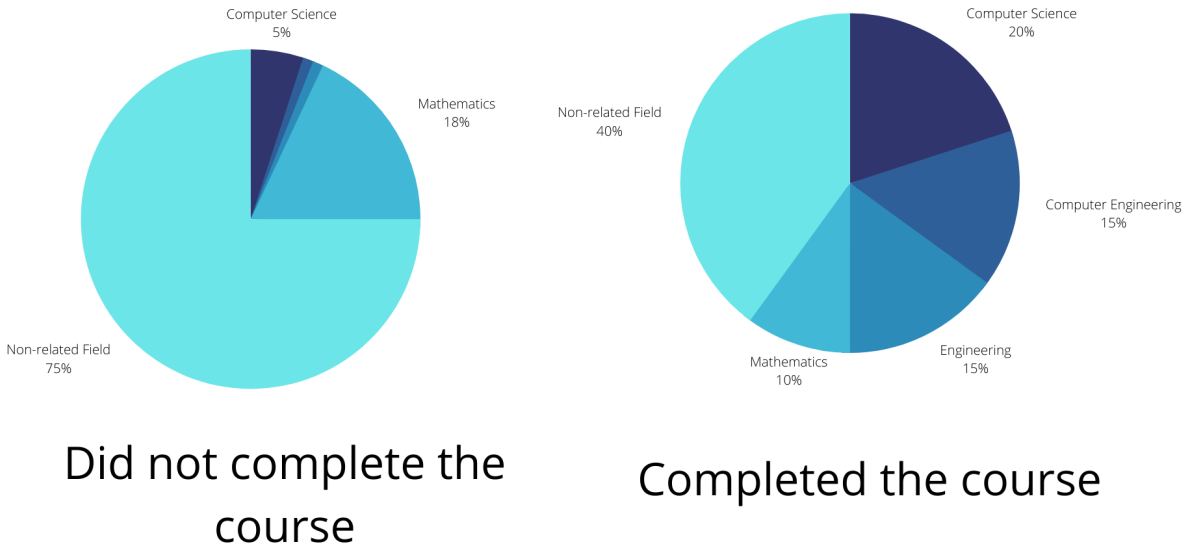


Figure 7

Percentage of Students Engaging in Computer Science Programs

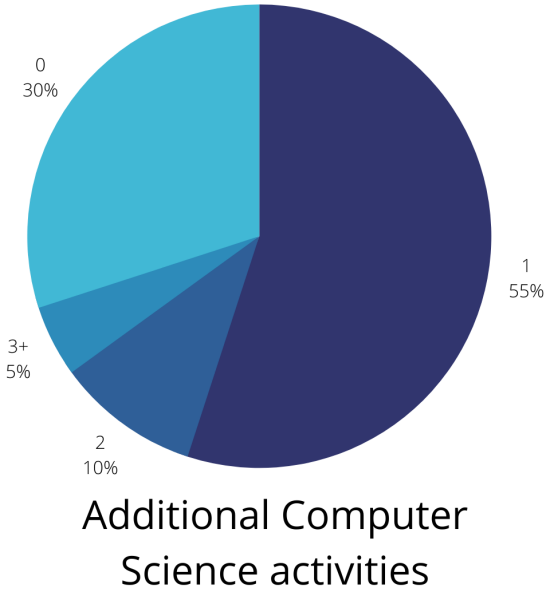
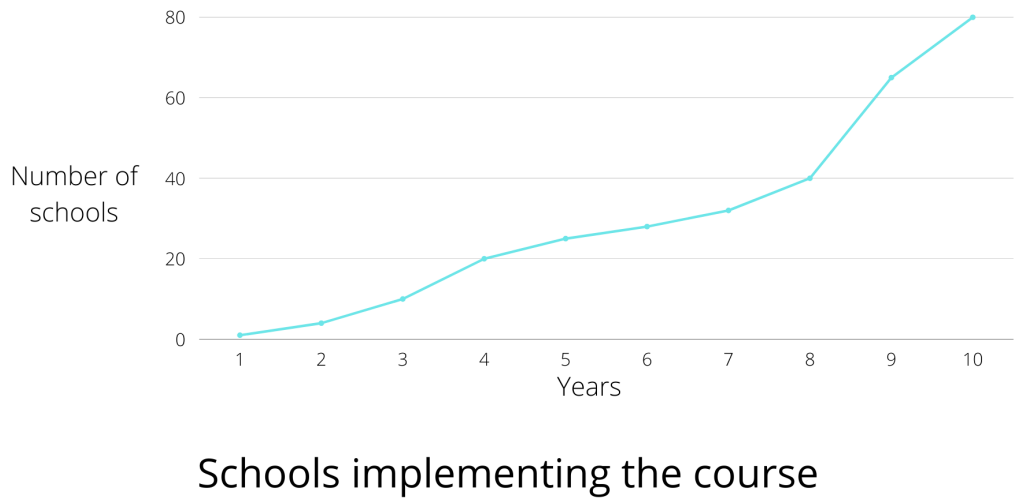


Figure 8

Number of Schools Adopting the Course



Conclusion

There is a growing demand in the job market for individuals with experience in computer science. Parents and students are also highly interested in their schools offering computer science courses (Gallup, 2015). Parents with incomes under $54,000 are also more likely to think computer science should be a required course than parents with an income between $54,000 and $105,000 or an income over $105,000 (Gallup, 2015). This course is designed to address this need for computer science education in low-SES schools. The target learners are middle school students with limited or no knowledge of computer science.

This course was designed using the ADDIE model of instructional design. Some of the key steps in this process were conducting a needs assessment, performing a cognitive task

analysis, and identifying instructional strategies to best support the goals of the instruction. The needs assessment made it possible to build a shared understanding of the instructional needs, the learners, and the learning environment. Then, building from the needs assessment a set of overarching learning outcomes were generated. During the cognitive task analysis, a subject matter expert was interviewed to ensure the instruction covered both the observable actions and cognitive decisions that experts make. Given this is an entry level course and the learners have low prior knowledge, many of the learning activities for the debugging lesson are supplantive in nature. The overall goal in designing this course is to create instruction that will enhance students' learning of computer science principles and create a strong foundation for students' continued success in the field of computer science.

References

Adams Becker, S., Cummins, M., Davis, A., Freeman, A., Hall Giesinger, C., & Ananthanarayanan, V. (2017). *NMC Horizon Report: 2017 Higher Education edition.* The New Media Consortium.

Anderson, L. W., Krathwohl, D. R., Airasian, P. W., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J., & Wittrock, M. C. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives* (Complete edition). Longman.

Bandura, A. (1997). *Self-Efficacy: The exercise of control*. Worth Publishers.

Clark, R. E. (1994). Media will never influence learning. *Educational Technology Research and Development*, *42*(2), 21–29.

Clark, R. E., Yates, K., Early, S., Moulton, K., Silber, K., & Foshay, R. (2010). An analysis of the failure of electronic media and discovery-based learning: Evidence for the performance benefits of guided training methods. *Handbook of Training and Improving Workplace Performance*, *1*, 263–297.

Eddy, S. L., Brownell, S. E., & Wenderoth, M. P. (2014). Gender gaps in achievement and participation in multiple introductory biology classrooms. *CBE—Life Sciences Education*, *13*(3), 478–492. https://doi.org/10.1187/cbe.13-10-0204

Gagné, R. M. (1984). Learning outcomes and their effects: Useful categories of human performance. *American Psychologist*, *39*(4), 377–385. https://doi.org/10.1037/0003-066X.39.4.377

Gallup. (2015). *Searching for computer science: Access and barriers in U.S. K-12 education*.

https://services.google.com/fh/files/misc/searching-for-computer-science_report.pdf

Gallup. (2016). *Trends in the state of computer science in U.S. K-12 schools*. https://services.google.com/fh/files/misc/trends-in-the-state-of-computer-science-report.pdf

Gunawardena, C., Frechette, C., & Layne, L. (2018). *Culturally inclusive instructional design: A framework and guide to building online wisdom communities*. Taylor & Francis.

Karpicke, J. D., & Grimaldi, P. J. (2012). Retrieval-based learning: A perspective for enhancing meaningful learning. *Educational Psychology Review*, *24*(3), 401–418.

Kirkpatrick, J., & Kirkpatrick, W. K. (2016). *Kirkpatrick's four levels of training evaluation*. Association For Talent Development.

Kirkpatrick, J., & Kirkpatrick, W. K. (2019). *An introduction to the New World Kirkpatrick® Model. Kirkpatrick Partners.* Kirkpatrick Partners.

Lombardozzi, C. (2015). *Learning environments by design*. American Society for Training & Development.

Mayer, R. E. (2011). *Applying the science of learning*. Pearson/Allyn & Bacon.

Mayer, R. E. (2020). *Multimedia learning*. Cambridge University Press.

Micari, M., Pazos, P., & Hartmann, M. J. Z. (2007). A matter of confidence: Gender differences in attitudes toward engaging in lab and course work in undergraduate engineering. *Journal of Women and Minorities in Science and Engineering*, *13*(3). https://doi.org/10.1615/JWomenMinorScienEng.v13.i3.50

Midgley, C., Maehr, M. L., Hruda, L. Z., Anderman, E., Anderman, L., Freeman, K. E., & Urdan, T. (2000). *Manual for the patterns of adaptive learning scales.* University of

Michigan.

Pajares, F. (2009). Self-efficacy theory. *Education.com*, 6.

Pintrich, P. R. (1991). *A Manual for the use of the motivated strategies for learning questionnaire (MSLQ)*. University of Michigan.

Reigeluth, C. M. (1992). Commentary: Elaborating the Elaboration Theory. *Educational Technology Research and Development*, *40*(3), 80–86.

Rooney, K., & Khorram, Y. (2020, June 12). *Tech companies say they value diversity, but reports show little change in the last six years*. CNBC. https://www.cnbc.com/2020/06/12/six-years-into-diversity-reports-big-tech-has-made-little-progress.html

Salomon, G. (1984). Television is "easy" and print is "tough": The differential investment of mental effort in learning as a function of perceptions and attributions. *Journal of Educational Psychology*, *76*(4), 647–658. https://doi.org/10.1037/0022-0663.76.4.647

Santrock, J. W. (2017). *Essentials of life-span development*. McGraw-Hill Education.

Smith, P. L., & Ragan, T. J. (2005). *Instructional design* (3rd ed.). John Wiley & Sons.

Thomas, M., Mitchell, M., & Joseph, R. (2002). The third dimension of ADDIE: A cultural embrace. *TechTrends*, *46*(2), 40.

U.S. Bureau of Labor Statistics. (2021, September 8). *Computer and information research scientists: Occupational outlook handbook:* https://www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm

Wigfield, A. (2002). *Development of achievement motivation* (J. S. Eccles, Ed.). Elsevier

Science.

Wood, L., & Howley, A. (2012). Dividing at an early age: The hidden digital divide in Ohio elementary schools. *Learning, Media and Technology*, *37*(1), 20–39. https://doi.org/10.1080/17439884.2011.567991

Appendix A

Pre-assessment Tools Before the Course

There are three components that make up the pre-assessment for this course, demographic items, knowledge items, and motivation items. All students will receive the following demographic items:

1. What is your age? _____
2. What is your gender? _____
3. What is your race/ethnicity? (Select all that apply)
    - ☐ White
    - ☐ Black or African American
    - ☐ American Indian or Alaska Native
    - ☐ Asian
    - ☐ Native Hawaiian or Pacific Islander
    - ☐ Other
4. Do you have access to a computer at home? (Select one)
    - ☐ Always
    - ☐ Most of the time
    - ☐ About half the time
    - ☐ Sometimes
    - ☐ Never
5. Do you have access to the internet at home? (Select one)
    - ☐ Always

- ☐ Most of the time
- ☐ About half the time
- ☐ Sometimes
- ☐ Never

All students will receive the same knowledge items. These items represent a small subset of the topics covered in the course and are designed to assess whether students have any prior experience with computer science. These items and their scoring are described in Table A1.

Table A1

Knowledge Items for Course Pre-assessment

Knowledge Questions	How is the response scored?
Which common construct can be used to repeat actions multiple times? **a.** an if statement **b.** an algorithm **c.** a list **d.** a loop	**Answer: d** 1 point for a correct answer, 0 points for an incorrect answer.
When creating a computer program you should include? **a.** all steps **b.** only the most important steps **c.** only unique steps **d.** only the decision steps	**Answer: a** 1 point for a correct answer, 0 points for an incorrect answer.
Which data structure is best for representing items in a specific order? **a.** a dictionary **b.** a list **c.** a map **d.** a string	**Answer: b** 1 point for a correct answer, 0 points for an incorrect answer.
Which of the following is the process of	**Answer: c**

trying to find an error in a program? **a.** coding **b.** programming **c.** debugging **d.** testing	1 point for a correct answer, 0 points for an incorrect answer.
(T or F). Computer science problems can be used to solve different types of problems.	**Answer: T** 1 point for a correct answer, 0 points for an incorrect answer.
(T or F). An algorithm is a series of steps used to solve a problem.	**Answer: T** 1 point for a correct answer, 0 points for an incorrect answer.
(T or F). Functions must have input parameters.	**Answer: F** 1 point for a correct answer, 0 points for an incorrect answer.
(T or F). Variables must have a value set.	**Answer: F** 1 point for a correct answer, 0 points for an incorrect answer.
In the line of code “size = 13”, the word size is called a(n) _____?	**Answer: Variable** 1 point for a correct answer, 0 points for an incorrect answer.
An element’s position in an array is called its _____?	**Answer: Index** 1 point for a correct answer, 0 points for an incorrect answer.
What common process in computer science requires using a series of strategies to find an error in a program?	**Answer: Debugging** 1 point for a correct answer, 0.5 points for a related answer (eg. testing, bug), and 0 points for incorrect and unrelated answers.

Students will be randomly assigned to three groups for the motivation items. Each group will receive a different motivation scale. The potential scales are self-efficacy, value, and skepticism about the relevance for future success. All three potential scales are presented below and measured using the following Likert scale:

Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
1	2	3	4	5

a. Self-efficacy scale (Adapted from Pintrich, 1991, p. 13)

    1. I believe I will receive an excellent grade in this course.
    2. I'm certain I can understand the more difficult computer science concepts presented in this course.
    3. I'm confident I can understand the basic concepts taught in this course.
    4. I'm confident I can understand the most complex concepts presented in this course.
    5. I'm confident I can do an excellent job solving the problems presented in this course.
    6. I expect to do well in this course.
    7. I'm certain I can master the skills being taught in this course.
    8. Considering the difficulty of the concepts, the problems, the teacher, and my skills, I think I will do well in this course.

b. Value scale (Adapted from Pintrich, 1991, p. 11)

    1. I think I will be able to use what I learn about computer science in other courses.
    2. It is important for me to learn computer science concepts.
    3. I am very interested in computer science.
    4. I think computer science concepts are useful for me to learn.
    5. I like the subject matter of computer science.

6. Understanding computer science concepts is very important to me.

c. Skepticism about the relevance for future success scale (Adapted from Midgley et al., 2000, p. 28)

    1. Even if I understand computer science, it will not help me have the kind of life I want when I grow up.
    2. My chances of succeeding later in life don't depend on learning computer science principles.
    3. Doing well in computer science doesn't improve my chances of having a good life when I grow up.
    4. Getting good grades in this course won't guarantee that I will get a good job when I grow up.
    5. Even if I am successful in computer science, it won't help me fulfill my dreams.
    6. Doing well in computer science won't help me have a satisfying career when I grow up.

Appendix B

Assessment Tools for Unit 10: Debugging

Pre-assessment for Unit 10: Debugging

1. *Which of the following is the process of trying to find an error in a program?*
    a. *Coding*
    b. *Programming*
    c. *Debugging*
    d. *Testing*
2. *Code that tries to get the 3rd element in a list that contains only 2 elements is an example of what type of error?*
    a. *Build error*
    b. *Runtime error*
    c. *Data error*
    d. *Design error*
3. *Which of the following is a line by line comparison of what has changed in a program's code?*
    a. *A log file*
    b. *A breakpoint*
    c. *A stack trace*
    d. *A diff*

4. *Consider the following scenario. You have a program that correctly adds two numbers and you want to modify it to multiply the numbers. After making changes, the program is not outputting the correct answers. Where is the problem likely occurring?*
    a. *The input*
    b. *The existing code*
    c. ***The modified code***
    d. *The output*
5. *List 2 questions you could ask to better understand the error that is occuring in a program.*

Post-assessment for Unit 10: Debugging

The post-assessment for this unit is made up of two components, a knowledge test and a problem solving task.

Knowledge Test

1. *What is debugging?*

   The process of finding the cause of a bug or error in a program.

2. *Why is it important to know how to debug?*

   Knowing how to debug can save you time and provide a structured approach when a program is not working properly. Nobody writes perfect code so debugging is a process you'll have to complete often.

3. *What is a stack trace?*

   A stack trace is a type of error message that describes the error and where in the program it occurs. It includes the list of method or function calls that led to the error.

4. *(T or F) A build error is an error that is found before the program runs.*

   True

5. *Which of the following is NOT a debugging strategy?*

   a. *Stepping through a program line by line*

   b. *Using breakpoints to check intermediate values*

   c. ***Rewriting the program***

   d. *Running subcomponents of the program*

6. *Identify 3 context clues in the following error message.*

```
./mergesort/MergeSorter.java:32: error: variable combinedSorted might not have been initialized
            combinedSorted.add(leftSorted.get(leftIndex));
            ^
./mergesort/MergeSorter.java:35: error: variable combinedSorted might not have been initialized
            combinedSorted.add(rightSorted.get(rightIndex));
            ^
./mergesort/MergeSorter.java:42: error: variable combinedSorted might not have been initialized
        combinedSorted.addAll(rightSorted.subList(rightIndex, rightSorted.size()));
        ^
./mergesort/MergeSorter.java:44: error: variable combinedSorted might not have been initialized
        combinedSorted.addAll(leftSorted.subList(leftIndex, leftSorted.size()));
        ^
4 errors
```

7. *What debugging strategy can be used to mimic breakpoints when debugger software is not available?*

   a. ***Print statements***

   b. *Stepping through the program*

   c. *Looking at a diff*

   d. *Running a test*

Problem Solving Task

The problem solving task consists of the following prompt and associated resources. The learners' solutions will be graded using the rubric in Table B1.

Prompt:

Given the following description of the problem and program code, identify where in the program the error is occurring. Be sure to include 1) a classification of the type of error, 2) a description of where the error is occurring and why, and 3) what debugging strategies you used to find the error and the reason for choosing each strategy.

Description of the Problem:

When you go to run the LetterSort program on the input list ["c", "b", "e", "A", "M", "i"], it runs to completion but outputs the list [“c”, “c”, “c”, “c”].

Figure B1

Problem Solving Task Program Code for Unit 10: Debugging

LetterSorter.java

```
1 package lettersort;
2
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6
7 public class LetterSorter {
8     public final List<String> CHAR_LIST = Arrays.asList(
9         "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m",
10         "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
11         "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
12         "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z");
13
14     public List<Integer> convertLettersToNumbers(List<String> letterList) {
15       List<Integer> numberList = new ArrayList<Integer>();
16       // Loop through the letters and convert it to a number.
```

```
17        for (int index = 0; index < letterList.size(); index++) {
18          for (int charIndex = 0; charIndex < CHAR_LIST.size(); charIndex++) {
19            // If the character in the CHAR_LIST matches add the index to the numberList.
20            if (CHAR_LIST.get(charIndex) == letterList.get(index)) {
21                numberList.add(charIndex);
22                continue;
23            }
24          }
25
26          // The character was not found so throw an error.
27          if (numberList.size() != index + 1) {
28              throw new RuntimeException(
29                  String.format("Invalid character: %s cannot be converted to a number.",
30                                letterList.get(index)));
31          }
32        }
33
34        return numberList;
35      }
36
37      public List<String> convertNumbersToLetters(List<Integer> numberList) {
38                List<String> letterList = new ArrayList<String>();
39        for (int index = 0; index < numberList.size(); index++) {
40            // The number does not match a character so throw an error.
41            if (numberList.get(index) > CHAR_LIST.size() || numberList.get(index) < 0) {
42                throw new RuntimeException(
43                    String.format("Invalid number: %d cannot be converted to a letter.",
44                                  numberList.get(index)));
45            }
46            // Look up the character that matches the number in the CHAR_LIST.
47            letterList.add(CHAR_LIST.get(numberList.get(index)));
48        }
49        return letterList;
50      }
51  }
```

MergeSorter.java

```
1  package mergesort;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.List;
6
7  public class MergeSorter {
8
```

```
9   public List<Integer> mergeSort(List<Integer> inputArray) {
10      // There's only 1 element so return the array.
11      if (inputArray.size() < 2){
12          return inputArray;
13      }
14
15      // Find the midpoint and split into two lists.
16      int midpoint = inputArray.size() / 2;
17      List<Integer> left = inputArray.subList(0, midpoint);
18      List<Integer> right = inputArray.subList(midpoint, inputArray.size());
19
20      // Sort both lists.
21      List<Integer> leftSorted = mergeSort(left);
22      List<Integer> rightSorted = mergeSort(left);
23
24      // Merge the sorted lists by keeping a pointer to the first unused number in each list.
25      // Compare the first unused item in each list and add the lowest to the combined list.
26      // Stop when the end of either list has been reached.
27      ArrayList<Integer> combinedSorted = new ArrayList<Integer>();
28      int leftIndex = 0;
29      int rightIndex = 0;
30      while (leftIndex < leftSorted.size() && rightIndex < rightSorted.size()) {
31          if (leftSorted.get(leftIndex) <= rightSorted.get(rightIndex)){
32              combinedSorted.add(leftSorted.get(leftIndex));
33              leftIndex++;
34          } else {
35              combinedSorted.add(rightSorted.get(rightIndex));
36              rightIndex++;
37          }
38      }
39
40      // Add the remaining items in the list where the end was not reached.
41      if (leftIndex == leftSorted.size()) {
42          combinedSorted.addAll(rightSorted.subList(rightIndex, rightSorted.size()));
43      } else {
44          combinedSorted.addAll(leftSorted.subList(leftIndex, leftSorted.size()));
45      }
46
47      return combinedSorted;
48    }
49 }
```

Main.java

```
1 import java.util.ArrayList;
2 import java.util.Arrays;
```

```
3 import java.util.List;
4 import lettersort.LetterSorter;
5 import mergesort.MergeSorter;
6
7 class Main {
8    public static void main(String args[]) {
9
10     List<String> input = Arrays.asList("c", "b", "e", "A", "M", "i");
11
12     LetterSorter letterSorter = new LetterSorter();
13     MergeSorter mergeSorter = new MergeSorter();
14
15     List<Integer> numberList = letterSorter.convertLettersToNumbers(input);
16     List<Integer> sortedList = mergeSorter.mergeSort(numberList);
17     List<String> sortedLetterList = letterSorter.convertNumbersToLetters(sortedList);
18
19     System.out.println("Sorted Array:");
20     System.out.println(sortedLetterList.toString());
21   }
22 }
```

Table B1

Problem Solving Task Rubric for Unit 10: Debugging

	Level of Performance		
Component	Excellent (3pts)	Good (2pts)	Needs improvement (1pt)
Error Classification	All correct classifications of the error are listed.	At least one correct classification is listed. However, the list may also include incorrect classifications or be missing correct classifications.	The classifications listed are incorrect.
Error Identification	The solution correctly identifies where in the program the error is (e.g. line number, variable, etc.) and why the error is occurring (e.g.	The solution may narrow down where the error is (e.g. line number, variable, function, etc.) but does not correctly identify why the error is occuring (e.g. incorrect	The solution does not narrow down where the error is occurring or may identify the wrong part of the program.

	incorrect syntax, not properly setting the variable, incorrect logic, etc.).	syntax, not properly setting the variable, incorrect logic, etc.).	
Strategy Use	Multiple debugging strategies were used to identify where the error is in the program and the learner can explain the reason for choosing those strategies.	At least one debugging strategy was used to identify where the error is in the program but the learner may not be able to explain the reason for choosing that strategy.	The learner was not able to identify a debugging strategy that could be used to identify where the error is in the program.

Example Solution:

This error is a logic error. On line number 22, the code passes the left half of the array into the mergesort function instead of the right half. I could tell it's a logic error because the program runs to completion but the output is not what I would expect. Additionally, there is no error message like we would find with a build error or a runtime error. When looking at the output from the error, there are a couple of things I noticed right away. First, the output is all the same letter. Second, the number of elements returned are not the same as there were in the original list. This suggests that something is going wrong around where the code splits the list and sorts the two halves. Since this is a multi-file program, I decided to break the problem down into three parts based on the different files, `Main.java` which starts the program and passes the arrays between the different functions, `LetterSorter.java` which converts back and forth between letters and numbers, and `MergeSorter.java` which splits the list and sorts it. Therefore, I knew the error was occuring in `MergeSorter.java`. Once I knew which file the error was

occurring in, I wanted to narrow it down to a specific line. Since there was no error message that gave a line number, I chose to step through the program with the input array from the description of the problem. When I got to line number 21, I had `left=["c", "b", "e"]` and `right=["A", "M", "i"]` written down as intermediate values. This meant that when I got to lines 21 and 22 the code would need to sort each of these lists. However, I noticed that line number 22 calls the method on `left` instead of `right`.

Appendix C

Evaluation Tools Immediately Following the Program Implementation

There are several different evaluation tools that will be used immediately following the course. The first is a student survey measuring Kirkpatrick Levels 1 and 2. The items for the survey are presented in Table C1.

Table C1

Student Survey for Kirkpatrick Levels 1 and 2

How effective was the class environment in helping you to learn computer science? (Level 1) Not at all effective Slightly effective Moderately effective Very effective Extremely effective
How confident are you that you'll be able to use what you've learned in future classes? (Level 1) Not at all confident Slightly confident Moderately confident Very confident Extremely confident
How likely are you to recommend this course to a friend? (Level 1) Extremely unlikely Somewhat unlikely Neither likely or unlikely Somewhat likely Extremely likely
How likely are you to continue learning computer science? (Level 2) Extremely unlikely Somewhat unlikely

Neither likely or unlikely
Somewhat likely
Extremely likely

Motivation items (one of the following based on random assignment; Level 2)

All scales measured using the following Likert scale:

Strongly disagree	Somewhat disagree	Neither agree nor disagree	Somewhat agree	Strongly agree
1	2	3	4	5

Self-efficacy scale (Adapted from Pintrich, 1991, p. 13)

1. I believe I will receive an excellent grade in this course.
2. I'm certain I can understand the more difficult computer science concepts presented in this course.
3. I'm confident I can understand the basic concepts taught in this course.
4. I'm confident I can understand the most complex concepts presented in this course.
5. I'm confident I can do an excellent job solving the problems presented in this course.
6. I expect to do well in this course.
7. I'm certain I can master the skills being taught in this course.
8. Considering the difficulty of the concepts, the problems, the teacher, and my skills, I think I will do well in this course.

Value scale (Adapted from Pintrich, 1991, p. 11)

1. I think I will be able to use what I learn about computer science in other courses.
2. It is important for me to learn computer science concepts.
3. I am very interested in computer science.
4. I think computer science concepts are useful for me to learn.

5. I like the subject matter of computer science.
6. Understanding computer science concepts is very important to me.

Skepticism about the relevance for future success scale (Adapted from Midgley et al., 2000, p. 28)

1. Even if I understand computer science, it will not help me have the kind of life I want when I grow up.
2. My chances of succeeding later in life don't depend on learning computer science principles.
3. Doing well in computer science doesn't improve my chances of having a good life when I grow up.
4. Getting good grades in this course won't guarantee that I will get a good job when I grow up.
5. Even if I am successful in computer science, it won't help me fulfill my dreams.
6. Doing well in computer science won't help me have a satisfying career when I grow up.

To further evaluate Kirkpatrick Level 2, students will be provided with a new problem they have not seen in the example problems during the course and asked to write code to solve the problem. The solutions the students produce will be evaluated with the rubric in Table C2. The rubric uses four levels of performance to ensure there are more levels of gradation as it is easy to make mistakes or overlook small details while still having an overall strong solution.

Coding Prompt:

> A group of friends is interested in sending messages to each other that cannot be read by others if intercepted. They settle on a system in which messages are sent as a list of characters sorted alphabetically. Each character in the list also has a number associated with it. If the character is the first in the alphabetically sorted list the number represents

its position in the message. For each character after that, the number represents its position in the message relative to the previous character. If a character shows up more than once in the message, they should be listed in the same order they appear in the message. For example, the message “Hello World” would be represented as [(“ ”, 5), (“H”, -5), (“W”, 6), (“d”, 4), (“e”, -9), (“l”, 1), (“l”, 1), (“l”, 6), (“o”, -5), (“o”, 3), (“r”, 1)]. Design the algorithm and write the code to encode and decode the messages. Be sure to include the test cases you would use to verify your algorithm.

Table C2

Coding Rubric

	Level of Performance			
Component	Excellent (4pts)	Good (3pts)	Satisfactory (2pts)	Needs improvement (1pt)
Algorithm	The algorithm for solving the problem is easy to understand, would result in a correct solution, and shows clear consideration for potential boundary conditions.	The algorithm would result in a correct solution for most cases, but may not be easy to understand or show clear consideration for potential boundary conditions.	The algorithm shows an understanding of the problem but may be missing a piece or two resulting in an incorrect solution for some cases.	The algorithm is hard to follow and missing pieces resulting in an incorrect solution for most cases.
Data structures	The solution to the problem uses the most time and space efficient data structures.	The solution sacrifices space or time efficiency for ease of understanding or	The solution uses one or two data structures that make the steps of the solution hard to follow.	The solution attempts to use data structures that cannot properly represent the data.

		improvements in the other.		
Coding	The code written to solve the problem is clear, uses concepts like loops and conditionals to be concise, aligns with the described algorithm, and aligns with standard style practices.	The code written to solve the problem aligns with the described algorithm but may contain a few unnecessary lines or lines that do not align with standard style practices.	The code written to solve the problem aligns with the described solution but does not make use of concepts like loops and conditionals to create concise code making it hard to follow.	The code written to solve the problem does not match the described algorithm and lacks loops and conditionals needed to be easy to follow.
Functionality	The solution runs properly and results in a correct solution for all test cases.	The solution runs properly and results in correct solutions for all but the boundary condition test cases.	The solution runs but does not result in a correct solution for basic test cases.	The solution does not run for any test cases.
Testing	The provided solution has test examples that cover all cases.	The provided solution has test cases to cover all basic cases but may be missing examples for boundary conditions.	The provided solution addresses most basic test cases but some are missing.	No potential test cases were provided.

Example Solution:

The algorithm to encode a message will split the message up into each character, then create a pair that contains each character and its position in the message. Those pairs will then be sorted alphabetically using the default unicode sorting. Once the list of characters

is sorted, we'll loop through the characters and update the number with each character to reflect the difference between its position and the position of the previous character. The algorithm to decode the message will create a string builder of the same length of the message. It will then loop through the encoded message adding the first character in its specified position and calculating the index for each subsequent character by adding its number to that index. That index will be used to add the character to the string. Once all the characters are added the decoded message will be returned.

MessageCoder.java

```java
package messagecoder;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import org.apache.commons.lang3.tuple.Pair;

public class MessageCoder {

    public List<Pair<String, Integer>> encodeMessage(String message) {

        // Save position information.
        ArrayList<Pair<String, Integer>> positionList = new ArrayList<Pair<String, Integer>>();
        for (int i = 0; i < message.length(); i++) {
            positionList.add(Pair.of(message.substring(i, i + 1), i));
        }

        // Sort the letters.
        Collections.sort(positionList, new SortPositionList());

        // Update the positions.
        ArrayList<Pair<String, Integer>> codedMessageList = new ArrayList<Pair<String, Integer>>();
        for (int i = 0; i < positionList.size(); i++) {
            Pair<String, Integer> pair = positionList.get(i);
            if (i == 0) {
                codedMessageList.add(pair);
            } else {
            Pair<String, Integer> previousPair = positionList.get(i - 1);
                codedMessageList.add(Pair.of(pair.getKey(), pair.getValue() - previousPair.getValue()));
            }
        }
```

```
34
35          return codedMessageList;
36      }
37
38      public String decodeMessage(List<Pair<String, Integer>> codedMessage) {
39          StringBuilder messageBuilder = new StringBuilder();
40          messageBuilder.setLength(codedMessage.size());
41
42          int index = 0;
43          for (int i = 0; i < codedMessage.size(); i++) {
44              Pair<String, Integer> pair = codedMessage.get(i);
45              if (i == 0) {
46                  // Use the number from the first character as the initial index.
47                  messageBuilder.setCharAt(pair.getValue(), pair.getKey().charAt(0));
48                  index = pair.getValue();
49              } else {
50                  // Use the displacement of the next character to get the updated index.
51                  index = index + pair.getValue();
52                  messageBuilder.setCharAt(index, pair.getKey().charAt(0));
53              }
54          }
55          return messageBuilder.toString();
56      }
57
58      // A comparison function for (letter, index) pairs.
59      public class SortPositionList implements Comparator<Pair<String, Integer>> {
60          public int compare(Pair<String, Integer> a, Pair<String, Integer> b) {
61              // If the letter is the same sort based on which shows up first in the message.
62              if (a.getKey().equals(b.getKey())) {
63                  return a.getValue().compareTo(b.getValue());
64              }
65              return a.getKey().compareTo(b.getKey());
66          }
67      }
68  }
```

MessageCoderTest.java

```
1  import messagecoder.MessageCoder;
2  import static org.junit.Assert.assertEquals;
3
4  import java.util.ArrayList;
5  import java.util.Arrays;
6  import java.util.List;
7  import org.apache.commons.lang3.tuple.Pair;
8  import org.junit.Assert;
9  import org.junit.Before;
10 import org.junit.Test;
11 import org.junit.runner.JUnitCore;
12 import org.junit.runner.Result;
13 import org.junit.runner.RunWith;
14 import org.junit.runner.notification.Failure;
15 import org.junit.runners.Suite;
16
```

```
17 public class MessageCoderTest {
18
19     private MessageCoder messageCoder;
20
21     @Before
22     public void setUp() {
23         messageCoder = new MessageCoder();
24     }
25
26     // Encode tests
27     @Test
28     public void encodeMessage_emptyList_returnsEmptyString() {
29         List<Pair<String, Integer>> encodedMessage = Arrays.asList();
30         String message = "";
31         assertEquals(messageCoder.encodeMessage(message), encodedMessage);
32     }
33
34     @Test
35     public void encodeMessage_singleLetter() {
36         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("b", 0));
37         String message = "b";
38         assertEquals(messageCoder.encodeMessage(message), encodedMessage);
39     }
40
41     @Test
42     public void encodeMessage_punctuation() {
43         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("!", 2), Pair.of("H", -2), Pair.of("i", 1));
44         String message = "Hi!";
45         assertEquals(messageCoder.encodeMessage(message), encodedMessage);
46     }
47
48     @Test
49     public void encodeMessage_lowercase() {
50         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("a", 1), Pair.of("c", -1), Pair.of("t", 2));
51         String message = "cat";
52         assertEquals(messageCoder.encodeMessage(message), encodedMessage);
53     }
54
55     @Test
56     public void encodeMessage_uppercase() {
57         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("E", 1), Pair.of("H", -1), Pair.of("L", 2),
58                                                                    Pair.of("L", 1), Pair.of("O", 1));
59         String message = "HELLO";
60         assertEquals(messageCoder.encodeMessage(message), encodedMessage);
61     }
62
63     @Test
64     public void encodeMessage_mixedcase() {
65         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("H", 0), Pair.of("i", 1));
66         String message = "Hi";
67         assertEquals(messageCoder.encodeMessage(message), encodedMessage);
68     }
69
70     @Test
71     public void encodeMessage_space() {
72         List<Pair<String, Integer>> encodedMessage =
```

```
73              Arrays.asList(Pair.of(" ", 1), Pair.of("a", -1), Pair.of("a", 4), Pair.of("c", 1), Pair.of("e", 1),
74                      Pair.of("p", -3), Pair.of("s", -1));
75          String message = "a space";
76          assertEquals(messageCoder.encodeMessage(message), encodedMessage);
77      }
78
79      @Test
80      public void encodeMessage_repeatedLetter() {
81          List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("a", 0), Pair.of("a", 1), Pair.of("a", 1));
82          String message = "aaa";
83          assertEquals(messageCoder.encodeMessage(message), encodedMessage);
84      }
85
86      @Test
87      public void encodeMessage_firstLetterFirstPosition() {
88          List<Pair<String, Integer>> encodedMessage =
89              Arrays.asList(Pair.of("a", 0), Pair.of("b", 1), Pair.of("c", 2), Pair.of("d", -1));
90          String message = "abdc";
91          assertEquals(messageCoder.encodeMessage(message), encodedMessage);
92      }
93
94      @Test
95      public void encodeMessage_helloWorld() {
96          List<Pair<String, Integer>> encodedMessage =
97              Arrays.asList(Pair.of(" ", 5), Pair.of("H", -5), Pair.of("W", 6), Pair.of("d", 4), Pair.of("e", -9),
98                      Pair.of("l", 1), Pair.of("l", 1), Pair.of("l", 6), Pair.of("o", -5), Pair.of("o", 3),
99                      Pair.of("r", 1));
100         String message = "Hello World";
101         assertEquals(messageCoder.encodeMessage(message), encodedMessage);
102     }
103
104
105     // Decode tests
106     @Test
107     public void decodeMessage_emptyList_returnsEmptyString() {
108         List<Pair<String, Integer>> encodedMessage = Arrays.asList();
109         String message = "";
110         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
111     }
112
113     @Test
114     public void decodeMessage_singleLetter() {
115         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("b", 0));
116         String message = "b";
117         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
118     }
119
120     @Test
121     public void decodeMessage_punctuation() {
122         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("!", 2), Pair.of("H", -2), Pair.of("i", 1));
123         String message = "Hi!";
124         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
125     }
126
127     @Test
128     public void decodeMessage_lowercase() {
```

```
129         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("a", 1), Pair.of("c", -1), Pair.of("t", 2));
130         String message = "cat";
131         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
132     }
133
134     @Test
135     public void decodeMessage_uppercase() {
136         List<Pair<String, Integer>> encodedMessage =
137             Arrays.asList(Pair.of("E", 1), Pair.of("H", -1), Pair.of("L", 2), Pair.of("L", 1), Pair.of("O", 1));
138         String message = "HELLO";
139         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
140     }
141
142     @Test
143     public void decodeMessage_mixedcase() {
144         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("H", 0), Pair.of("i", 1));
145         String message = "Hi";
146         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
147     }
148
149     @Test
150     public void decodeMessage_space() {
151         List<Pair<String, Integer>> encodedMessage =
152             Arrays.asList(Pair.of(" ", 1), Pair.of("a", -1), Pair.of("a", 4), Pair.of("c", 1), Pair.of("e", 1),
153                           Pair.of("p", -3), Pair.of("s", -1));
154         String message = "a space";
155         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
156     }
157
158     @Test
159     public void decodeMessage_repeatedLetter() {
160         List<Pair<String, Integer>> encodedMessage = Arrays.asList(Pair.of("a", 0), Pair.of("a", 1), Pair.of("a", 1));
161         String message = "aaa";
162         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
163     }
164
165     @Test
166     public void decodeMessage_firstLetterFirstPosition() {
167         List<Pair<String, Integer>> encodedMessage =
168             Arrays.asList(Pair.of("a", 0), Pair.of("b", 1), Pair.of("c", 2), Pair.of("d", -1));
169         String message = "abdc";
170         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
171     }
172
173     @Test
174     public void decodeMessage_helloWorld() {
175         List<Pair<String, Integer>> encodedMessage =
176             Arrays.asList(Pair.of(" ", 5), Pair.of("H", -5), Pair.of("W", 6), Pair.of("d", 4), Pair.of("e", -9),
177                           Pair.of("l", 1), Pair.of("l", 1), Pair.of("l", 6), Pair.of("o", -5), Pair.of("o", 3),
178                           Pair.of("r", 1));
179         String message = "Hello World";
180         assertEquals(messageCoder.decodeMessage(encodedMessage), message);
181     }
182
183 }
```

In addition to the coding performance task, students will also be asked to complete an essay describing a real world problem they think could be solved using computer science. The prompt for the essay follows and the rubric can be found in Table C3.

Essay Prompt:

> Describe a real world problem you care about that could benefit from computer science and propose a solution (no need to write code). Be sure to touch on what properties of the problem lend themselves well to a computer science based solution and what concepts they align well with.

Table C3

Essay Rubric

	Level of Performance			
Component	Excellent (4pts)	Good (3pts)	Satisfactory (2pts)	Needs improvement (1pt)
Problem	The description of the problem is clear, identifies key properties of the problem, and explains why it is an important problem to solve.	The description of the problem is hard to follow but contains the key properties of the problem and why it is important to solve.	The description of the problem is clear but fails to identify either the key properties or why it is an important problem to solve.	The description of the problem is hard to follow and lacks the key properties of the problem and/or why it is important to solve.
Algorithm	The algorithm for solving the problem is easy to understand, would result in a correct	The algorithm would result in a correct solution for most cases, but may not be	The algorithm shows an understanding of the problem but may be missing a	The algorithm is hard to follow and missing pieces resulting in an incorrect

	solution, and shows clear consideration for potential boundary conditions.	easy to understand or show clear consideration for potential boundary conditions.	piece or two resulting in an incorrect solution for some cases.	solution for most cases.
Concepts	At least 3 major computer science concepts are mentioned and accurate connections are made between the concepts and the selected problem.	At least 3 major computer science concepts are mentioned but not all of the connections that are made are accurate.	Only one or two major computer science concepts are mentioned and/or the connections that are made are inaccurate.	No major computer science concepts are mentioned or no connections are made between the concepts and the selected problem.

Finally, students will be asked to complete a journal reflection about their experience in the course. This reflection is not graded and designed as an open-ended tool for understanding students' experiences. A coding process will be conducted on the responses to identify key themes. The prompt for the journal reflection follows:

> In about 500 words, describe your experience in this course. How has your approach to computer science and problem solving changed? Do you see computer science playing a different role in your life after having taken this course?

Appendix D

Evaluation Tools Delayed For A Period After The Program Implementation

The interview protocol that will be used follows:

1. **Date/time/location**
2. **Background Information on Interviewee**
    a. What is your name?
    b. What grade are you in?
    c. Tell me about your experience with computer science.
3. **Knowledge items**
    a. Tell me what you think is the most important piece of computer science after completing this course. (Level 2)
    b. Can you give examples of problems that could be solved using computer programs? (Level 2)
    c. What are some jobs you know of that use computer science? (Level 2)
4. **Motivation items**
    a. How important is it to you to continue learning computer science? (Level 2)
    b. What are some of the reasons you chose to learn computer science? (Level 2)
    c. How confident are you that you'll be able to use what you've learned in this course? What makes you say that? (Level 2)
5. **Additional exploratory items**
    a. Do you think computer science should be a required course for all middle school students? Why or why not? (Level 2)

b. Do you think computer science is a field where everyone can succeed regardless of their race or ethnicity? Why or why not? (Level 2)
c. Do you think computer science is a field where everyone can succeed regardless of their gender? Why or why not? (Level 2)
d. Tell me about how you approach solving problems. (Level 3)
e. How do you use computer science in your life? (Level 3)
f. How relevant is what you learned in this course to your current experiences? (Level 1)
g. What computer science opportunities are you currently involved with? Please list all opportunities. (Level 3 & Level 4)