Barriers to Computer Programming Student Success: A Quantitative Study of Community College Students in Southwest Missouri

by

Tiffany D. Ford

August, 2015

A Dissertation submitted to the Education Faculty of Lindenwood University in partial fulfillment of the requirements for the degree of

Doctor of Education

School of Education

Barriers to Computer Programming Student Success:

A Quantitative Study of Community College Students

in Southwest Missouri

by

Tiffany D. Ford

This Dissertation has been approved as partial fulfillment

of the requirements for the degree of

Doctor of Education

Lindenwood University, School of Education

Sherry R. DeVore	8/20/15
Dr. Sherry DeVore, Dissertation Chair	Date
[illegible]	8/20/15
Dr. Rhonda Bishop, Committee Member	Date
[illegible]	8/31/15
Dr. Mark Snyder, Committee Member	Date

Declaration of Originality

I do hereby declare and attest to the fact that this is an original study based solely upon my own scholarly work at Lindenwood University and that I have not submitted it for any other college or university course or degree.

Tiffany Dawn Ford

Signature: ____ Date: _____________

Acknowledgements

Thanks are owed to my committee members, Dr. Sherry DeVore for your guidance and expertise in helping me navigate my way through this process and Dr. Mark Snyder for your insight on my work and consistent support throughout the years. Special thanks to my committee chair, Dr. Rhonda Bishop, for her unfailing motivation and willingness to support my goals and ideas. I am truly honored to have found such mentors for this process.

I would like to thank all of the participants who took the time to help in this research study for your input was valuable and necessary for this to be possible. Finally, thanks are owed to my family for their love and support. My husband Michael deserves my deepest thanks for his confidence in me and for his support in all of my endeavors. I also want to thank my son, Desmond, who is my constant source of strength. This milestone in my life would not have been possible without your love and support.

Abstract

Student success in computer programming courses has been a long-studied problem and computer science major retention has historically been substantially lower than other majors. The issue of retention for computer science majors has become more pronounced in two-year, open-enrollment institutions. This quantitative study, grounded in Gardner's theory of multiple intelligences, attempted to address some of the causes of poor retention for entry-level computer science majors at two-year colleges by looking for predictors of student success in their first computer programming course. Two of the intelligences from Gardner's (1993) theory, Logical-Mathematical and Visual-Spatial, were used along with two factors: student success in previous mathematics courses and the student's own perception of his or her programming skill. Three research questions guided the study and a survey instrument was developed to evaluate student success factors such as the number of credit hours enrolled and the amount of time spent on homework. Secondary data were obtained from the institution, which contained grades from math and programming courses. After analysis of the data, results indicated there was not a statistically significant difference in student success in entry-level computer programming classes after having taken at least a college-level math course. There was a significant different found for students who had completed an intermediate-level math class before taking their programming course. The findings of this study may be used to help two-year community college administrators determine the benefits of higher prerequisites for beginning programming courses to increase retention and student success.

Table of Contents

List of Tables

List of Figures

Chapter One: Introduction

There is an underlying problem which has existed since the introduction of computer science as a degree field at Cambridge in 1953: Can anyone who wants to learn computer programming be capable of learning it? (Jones, 2001). A study conducted by Ambrósio, Costa, Almeida, Franco, and Macedo (2011) found some distinctive cognitive abilities in students who have had success in learning computer programming, which abilities seem to point to an affirmative answer. The research determined:

> Such aptitudes are represented in three groups of students: those who learn easily, those who never seem to fully grasp what programming requires despite true effort, and those who experience a sudden insight, making them leap from a point where they had difficulties to another where they overcome them. (Ambrósio et al., 2011, p. 1)

Instructors in studies by Groff (2013), Kay (2011), and Vihavainen, Paksula, and Luukkainen (2011) stated the opinion that students are capable of learning to program if given the right instruction.

During a study by Vihavainen et al. (2011), a new method of teaching introductory computer science courses, Extreme Apprenticeship, was developed where students were given support, feedback, and a specific method in which goals were set and students moved through the material. The focus of Extreme Apprenticeship was on the mastery of topics instead of objective learning and the team of Vihavainen et al (2010) garnered positive feedback from the study. When learning programming, "Extreme Apprenticeship method's idea [*sic*] of taking continuous feedback and scaffolding to an extreme level provides enough support to also help some of the inefficient novices, who usually drop programming courses to learn programming" (Vihavainen et al., 2011, p. 97).

Students do not always find the complex problem solving techniques to be learned as stimulating as their instructors would hope, and often find "Computer Science to be difficult, tedious, boring, irrelevant and asocial" (Kay, 2011, p. 177). The field of computer science keeps changing, which makes it challenging from a teaching perspective and even more challenging for those who want to stay current in the field (Hoda & Andreae, 2014; Oram & Greg, 2011). Since 1953, the common theme among educators has been that programming is difficult to learn (Oram & Greg, 2011). Many studies have focused on this topic as well as the failure and drop rate for computer science majors (Porter, Bailey-Lee, & Simon, 2013a; Porter, Guzdial, McDowell, & Simon, 2013b; Reese, Jankun-Kelly, Henderson, & Lee, 2013; Stewart-Gardiner, 2011; Watson & Li, 2014).

Compounding the problem of retention is the added issue of students who complete computer programming courses but do not actually understand how to program (Boesch & Steppe, 2011; Hoskey & Maurino, 2011; Piteira & Costa, 2012). Instructors continue to struggle to teach the principles needed for success in the computer programming field to students who do not seem to be able to learn them (Hoda & Andreae, 2014; Hu, 2011; Scott & Ghinea, 2013). The concerns of retention and student success have led many educators to begin to think some students do not have the ability to learn the necessary skills (Hoda & Andreae, 2014; Hu, 2011; Scott & Ghinea, 2013). In response to this concern, educators have started to assess and evaluate students for aptitude using various tests and simulations (Boesch & Steppe, 2011; Leal, 2013; Scott &

Ghinea, 2013). Over the years, several approaches to teaching computer programming courses, such as Problem-Based Learning, the use of manipulatives, the creation of personalized programs of study, the use of video game projects to engage students, and the creation of new teaching methods have emerged in response to make computer programming more accessible for students (Ambrósio et al., 2011; Kay, 2011; Vihavainen et al., 2011).

In Chapter One, an overview of the purpose for this study on student aptitude for computer programming is given, and the theoretical framework which guided the study is discussed. A statement of the problem is given and each of the research questions are reviewed. Chapter One also contains information about definitions of the terms; limitations, which are inherent in the design of this study; and assumptions.

Background of the Study

Computer programming is a distinct craft which demands ongoing practice to master and requires a unique mixture of creativity, attention to detail, problem solving skills, and foundational logic that many students have a hard time grasping (Oram & Greg, 2011; Scott & Ghinea, 2013). According to Oram and Greg (2011), researchers believe human brains are naturally good at using languages, but programming languages are artificial and contrived. Computer programming is not considered an innate activity and can be considered equal to learning to read and writing a foreign language one will never get to hear (Hoskey & Maurino, 2011).

Going back to the beginning of computer science as a field, educators have worked to find a way to make their subject matter appealing and accessible for students, and there are studies as far back as the 1960s, when the industry was first taking off, which explored the idea of programming aptitude (Hu, 2011). Many studies focused on "correlating programming aptitude with factors that are unrelated with programming, either the student academic record or psychological features" (Leal, 2013, p. 2). Other studies evaluated the use of aptitude tests students could take that would allow schools to determine if a student had the ability to learn the computer science curriculum (Avancena, Nishihara, Vergara, & City, 2012; Boesch & Steppe, 2011). When testing students for aptitude before entering a computer science program, some schools developed tests and others created simulations which would evaluate the student's ability to solve problems or use reasoning (Boesch & Steppe, 2011; Leal, 2013; Scott & Ghinea, 2013).

Industry partners have created further aptitude tests, such as the International Business Machines Corporation Programmer Aptitude Test (Mazlak, 1980). While many of these tests evaluate whether a student has the mental models or logical thinking patterns to match typical computer programming operations, the tests do not always have a statistically significant predictive power concerning whether or not a student will be capable of passing computer science courses (Boesch & Steppe, 2011). As the failure rate problem for entry-level computer science students continues to exist, schools around the world have sought ways to determine the aptitude of incoming students for computer programming work (Boesch & Steppe, 2011; Leal, 2013; Scott & Ghinea**,** 2013). Testing students appears to make sense when "programming has a significant place in the curriculum and which [computer science departments] have the luxury of selecting students from many applicants, a method to identify early those who are likely to succeed is desirable" (Boesch & Steppe, 2011, p. 1).

The issue of retention in computer science is beginning to have a negative impact on the job field (Tabarrok, 2012). Since schools are unable to produce a large number of computer science graduates, the job market is beginning to see a decline in available applicants to meet the needs of entry-level employment in computer science jobs (Watson & Li, 2014).The focus on science, technology, engineering, and mathematics education has only grown as the demand for technically-oriented jobs has increased (Tabarrok, 2012; Watson & Li, 2014). For example, in 2009 there were 37,994 bachelor's degrees in computer fields given in the United States, which is 36% less than there were 25 years ago (Tabarrok, 2012).

A study by Oram and Greg (2011) revealed that while student interest in the field of computer science is not declining, many students drop out of the computer science major after failing in their programming courses. The cause for the drop in degree-earning students is the lack of success students are having in computer programming courses (Oram & Greg, 2011). This failure trend is not just a national problem, and there are "indications that roughly one out of every three students who start a CS1 (Computer Science 1) course, around the world in all kinds of institutions, fails or gives up" (Oram & Greg, 2011, p. 112).

Educators feel like they are in a position where they need to increase enrollment to meet industry demand but are setting students up for failure. (Avancena et al., 2012; Boesch & Steppe, 2011; National Science Foundation, 2012). Due to the increases in industry demand shown by Tabarrok (2012) and Watson and Li (2014), more students, who may not have the ability to be successful in these courses, are being pushed into computer programming classes (Watson & Li, 2014). The issue of student completion is compounded by college administrators who want low drop and fail rates, which seems difficult in a computer science program (Blaho, Foltin, Fodrek, & Murgaš, 2012; Hoskey & Maurino, 2011; McClenney, Marti, & Adkins, 2012; Watson & Li, 2014).

Theoretical Framework

This study's purpose was to determine if there were indicators of programming success for students in two-year colleges who were majoring in computer science by looking at their success in other academic areas, such as mathematics. This study was conducted using a quantitative approach based on the framework of Gardner's (1985) theory on multiple intelligences (MI). This theory was first introduced by Howard Gardner in 1985, and today, through further research, identifies nine distinct intelligences (Gardner, 2013). Gardner's (2013) current theory of MI has emerged from recent cognitive research and allows researchers to document the extent to which students possess different frames of mind and therefore learn, remember, perform, and understand in different ways.

In Gardner's (2013) most recent book, he stated the problem in the educational system is the assumption everyone can learn the same material in the same way. The theory of MI indicates no two people will exhibit the same intelligences in the same proportions, since it is not only educational background which is the foundation for these intelligences, but also a person's childhood and personal experiences (Gardner, 2013). The theory of MI, unlike other theories about intelligence, encompasses a wider, more universal set of competencies than the idea of a single general intelligence.

Only people with a select set of intelligences perform and excel in some roles, areas, and professions (Ahanbor & Sadighi, 2014; Ghazi, Shahzada, Gilani, Shabbir, &

Rashid, 2011). In computer programming, there are two intelligences from the theory of MI that would give students the best set of skills for the profession (Korkmaz, 2012). The first intelligence that would help computer programmers is Logical-Mathematical, which is defined as the ability to reason, calculate, think conceptually and abstractly, and explore patterns and relationships (Gardner, 2013). People who are strong in the intelligence of Logical-Mathematical like to experiment, solve puzzles, and ask questions (Gardner, 2013). It is best for people who are strong in the Logical-Mathematical intelligence to learn and form concepts before they have to deal with details (Gardner, 2013). The second intelligence that would help computer programmers is the area of Visual-Spatial, which is defined as the ability to think in terms of physical space, as architects and sailors (Gardner, 2013). People strong in the area of Visual-Spatial are very aware of their environments and like to draw, work jigsaw puzzles, read maps, and daydream (Gardner, 2013).

In the past, schools used aptitude and intelligence testing to determine if a student should be accepted into a computer science program or if a student had particular traits or aptitudes that would make the student suitable in a particular career field (Boesch & Steppe, 2011; Leal, 2013; Scott & Ghinea**,** 2013). However, many educators, career counselors, and psychologists have questioned the validity of these tests since aptitude tests are not capable of measuring qualitative traits, such as social perceptiveness (Ahanbor & Sadighi, 2014; Ghazi et al., 2011). Counselors working to help students achieve career goals believe, “the single most important contribution education can make to a child’s development is to help him toward a field where his talents best suit him” (Goleman, 1986, p. 1). The theory of MI aligns with the idea of differentiated instruction and may assist in determining whether a student can be successful in a particular field (Ahanbor & Sadighi, 2014; Ghazi et al., 2011; Korkmaz, 2012).

While the theory of MI has been a foundational theory in curriculum development and instructional design in the field of education, Gardner (1985, 1993, 2013) has often spoken about the misinterpretation of MI and its misuse in the educational setting. Many educators seem to feel the theory of MI shows the need to teach subjects using all identified intelligences or to identify student skills in an intelligence and then gear all lessons in that direction (Ahanbor & Sadighi, 2014; Ghazi et al., 2011; Korkmaz, 2012). Gardner (1985, 1993, 2013) gave examples of these misapplications, such as singing the multiplication tables as a way to reach students with strong musical intelligence. Groff (2013) cited a lesson titled, T*he Dance of the Fractions,* in which students dance to a song about fractions. Misunderstandings on how to apply the theory of MI have led to labeling students as visual learners or bodily-kinesthetic learners, which has mixed and confused the theory of MI with learning styles (Groff, 2013).

This study used the theory of MI as Gardner intended (Gardner, 2013), by focusing on the theory that everyone has innate abilities that can be utilized to learn needed skills. The theory of MI embraces the concept of finding out where a student's strengths lie and where the best fit will be for those strengths (Gardner, 2013). Through this framework, this study explored the idea only students with the correct set of intelligences are capable of learning the concepts of computer programming successfully.

Statement of the Problem

The relationship between math and programming ability has been a long-held belief in the industry, where several studies have shown having a strong mathematical background was a predictor for programming success (Ambrósio et al., 2011; Erdogan, Aydin, & Kabaca, 2008; Hu, 2011; Scott & Ghinea, 2013). This is not unexpected since math can give students a foundation in logical thinking and problem solving, both of which are necessary to begin to solve the problems found in computer programming (Ambrósio et al., 2011; Erdogan, Aydin, & Kabaca, 2008; Hu, 2011; Scott & Ghinea, 2013). However, throughout many of the studies on the relationship between programming and mathematics, the issue of two-year students has not been addressed. Since the majority of four-year students have taken a minimum of college algebra, if not calculus 1, before they take their first programming course (Hoskey & Maurino, 2011), these studies cannot be applied to two-year students who may be at remedial math levels when they enter their computer science program (Jenkins & Cho, 2012).

Many studies found there is a difference in the admissions process between the two types of schools, which caused a problem (Ambrósio et al., 2011; Erdogan et al., 2008; Hoskey & Maurino, 2011; Hu, 2011; Scott & Ghinea, 2013). Most four-year schools can selectively admit students to their program and require rigorous entrance exam scores, whereas two-year schools are generally open admissions, and any student with any educational background can apply and attend (Clark, Waller, Lumadue, & Hendricks, 2012). Given the incoming two-year student has a very different educational foundation (Clark et al., 2012), results of studies completed on university students should not be applied.

Purpose of the Study

Each year, many students attempt their first programming course, and across the country, colleges continue to see drop rates around 30% (Hoskey & Maurino, 2011;

National Science Foundation, 2012). There have been several studies on this subject (Blaho et al., 2012; Hoskey & Maurino, 2011; McClenney et al., 2012; Watson & Li, 2014); however, continued research is purposeful as there have been rapid changes in technology in the industry, as well as the technology resources in the classroom. The purpose of this study was to determine if there were indicators for programming aptitude for students enrolled in two-year schools. This study looked for a student's ability to learn computer programming based on his or her math aptitude and indicators for student success.

Research questions and hypothesis. The following research questions guided the study:

1. Is there a statistically significant difference between a student's previous success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?

   $H1_0$: There is no statistically significant difference between the student's success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

   $H1_a$: There is a statistically significant difference between the student's success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

2. Is there a statistically significant difference between a student's previous success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?

H2₀: There is no statistically significant difference between the student's success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

H2ₐ: There is a statistically significant difference between the student's success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

3. What indicators do students in their first computer programming course report as supporting their aptitude for the field of computer science?

Definitions of Key Terms

For the purposes of this study, the following terms were defined:

Active learning based teaching model. A teaching methodology for computer science courses that consist of four stages: trigger, activity, discussion, and summary (Hazzan, Lapidot, & Ragonis, 2011).

Alice. A free educational tool to teach computer programming through a drag-and-drop environment to create computer animations using three-dimensional models and to engage programming students through video game development (Alice Educational Software, 2015).

Aptitude. In this study, aptitude was defined as "a state or condition of the person that makes it possible to engage profitably in a given learning activity" (Snow, 1992, p. 8).

Computational thinking. A form of thinking which allows a person to deal with systems that generate large amount of data and better understand the constraints to a problem, computational limits, and complexity (Hu, 2011).

Computer programming. Computer programming is defined as “a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result” (U.S. Copyright Office, 2014).

Dynamic typing. A computer programming concept in which the compiler waits until runtime to see if an operation requested by the code is possible. This method is less strict about what the complier allows at compile time (Griffiths, 2013).

Extreme apprenticeship. A teaching methodology where students complete work in groups or pairs with continuous feedback (Vihavainen et al., 2011).

Flipped classroom. An instructional strategy that reverses the traditional classroom environment by delivering instructional content online and moving activities that may have traditionally been considered homework into the classroom (Mok, 2014).

Intelligence. Intelligence is defined as “the skill in achieving whatever it is you want to attain in your life within your sociocultural context by capitalizing on your strengths and compensating for, or correcting, your weaknesses” (Sternberg, 2004, p. 1).

Pair programming. A teaching methodology for computer programming courses where two programmers work together at the same keyboard (Braught, Wahls, & Eby, 2011).

Peer instruction (PI). A computer programming teaching methodology focusing on class-wide discussion and a student-to-student learning model (Porter et al., 2013a).

Pex4Fun. A web-based computer programming simulation tool created by the Research in Software Engineering (RiSE) group at Microsoft Research (Tillmann, De Halleux, Xie, & Bishop, 2012a).

Problem-based learning. An instructional method characterized by the use of real world problems as the context within which students learn critical thinking, problem solving skills, and acquire knowledge of the essential concepts of the course (Ambrósio et al., 2011).

Scratch. A programming language created by the Lifelong Kindergarten Group at the Massachusetts Institute of Technology Media Laboratory (Meerbaum-Salant, Armoni, & Ben-Ari, 2013). This language was designed to be an entry-level into computer programming and allows for dragging and dropping blocks to represent program components (Meerbaum-Salant et al., 2013).

Success. In this study, success was defined as a state of educational attainment or academic achievement where "entering students persist to completion and attainment of their degree, program, or educational goal" (Cuseo, 2013, p. 1).

Limitations and Assumptions

The following limitations were identified in this study:

Sample demographics. The sample used in this study was limited to currently-enrolled students at a two-year, open-enrollment school in southwest Missouri. Therefore, the results only reflect the experience of students in the southwest Missouri area and may not be generalizable to other populations (Creswell, 2014). This study was also limited to students who voluntarily agreed to participate and were enrolled in their first computer programming course during the fall semester of 2014 and spring semester of 2015. Therefore, the results are limited in scope and represent conditions existing during the timespan in which the data were collected.

Academic success. This study focused on data that showed a student's academic success in a particular course. Academic success may have been influenced by several factors outside the scope of this study (Felder & Brent, 2005; Kendall & Schussler, 2012). These areas could be: class size, the student's comfort level with the course material, previous training from other institutions, and the course instructor's teaching style (Felder & Brent, 2005; Kendall & Schussler, 2012).

Instrument. This study contained an original survey, which must be considered a limitation (Fraenkel, Wallen, & Hyun, 2014). In Chapter Three, the steps taken to validate the survey and create questions designed to avoid confusion are outlined. The researcher was not able to check for understanding of the survey questions by the participants while they were completing the survey. Since participation in the survey was voluntary, the level of participation was unpredictable and affected the sample size. The time allotted for the survey was two weeks during the midterm period, which may have affected response time due to midterm exam preparation. Students who had dropped the class prior to the point the survey was made available did not get the opportunity to take the survey.

Attendance policy. At the institution where this research was performed, a mandatory attendance policy is in place which requires students to be present for a specific number of seated course periods or to regularly participate in online courses (Institution Course Catalog, 2015). Students who are not in compliance with the attendance policy are withdrawn from the course by their instructor (Institution Course Catalog, 2015).

The following assumptions were accepted:

1. The responses of the participants were offered honestly and without bias.
2. The introductory programming course from which these data were collected was taught using the same course content, objectives, and delivery method for all sections regardless of instructor or campus location.
3. The course syllabi and abstracts were reviewed by the researcher prior to the beginning of the study to ensure each course contained an equal level of academic rigor.
4. The qualifications of each instructor who taught the introductory programming course included in this study were assumed to be equal. Each instructor's resume and teaching history was reviewed by the researcher to ensure an equal level of competency in the required course skills and teaching experience at the college level.

Summary

Computer programming aptitude has been heavily studied over the years but not in the context of the two-year college student (Ambrósio et al., 2011; Clark et al., 2012; Erdogan et al., 2008; Hoskey & Maurino, 2011; Hu, 2011; Scott & Ghinea, 2013). Concepts of retention, academic placement, and proficiency in related fields, such as mathematics, heavily tie into the study of student aptitude in the field of computer science (Hoda & Andreae, 2014; Veilleux et al., 2013). Research has shown two-year colleges may not be preparing students in foundational academic areas as well as needed, which could be causing a continually high failure rate in computer science courses (Jenkins & Cho, 2012; Scott-Clayton, 2011).

One of the biggest problems for schools is "community college developmental instruction is generally narrowly focused on helping student [*sic*] take and pass college-level math and English courses rather than preparing them for success in college-level programs of study generally" (Jenkins & Cho, 2012, p. 1). This study was designed to determine if there was a statistically significant difference between student pass rates in their first computer programming course and their math ability and whether there were indicators for student aptitude and success from two-year college students.

In Chapter Two, a review of the literature relevant to this topic is provided. Previous studies involving computer programming aptitude are reviewed. Chapter Two also provides a review of new teaching methodologies being used by instructors to help engage students in computer programming courses. The information in these sections provides an overview of the previous work completed in this field and shows how this study fills the gap in current research about two-year college students in the field of computer science.

Chapter Two: Review of Literature

Professors in the field of computer science have known for decades there are problems in several areas such as retention, content delivery, and the expectations of rigor in their departments (Ambrósio et al., 2011; Kay, 2011; Nilsen & Larsen, 2011; Vihavainen et al., 2011). These topics have been widely discussed and studied over the years, as researchers have worked to answer questions about why some drop computer science classes while other students fail to grasp knowledge which comes easily to other students (Boesch & Steppe, 2011; Oram & Greg, 2011). The large majority of this research was conducted in a four-year university environment where admission requirements, rigor, and student expectations are very different from requirements in a community college environment of open-enrollment (McClenney et al., 2012). It has become apparent community colleges have the same problems reported by their four-year counterparts with a much larger set of variables when trying to help students succeed (Ambrósio et al., 2011; Erdogan et al., 2008; Hoskey & Maurino, 2011; Hu, 2011; Scott & Ghinea, 2013).

The purpose of this study was to provide research to instructors of computer science in community colleges to help them identify students who have the ability to be successful in computer science courses. This research study provided statistical results which may be used by colleges to identify students who require remediation in their mathematic skills before declaring a major in computer science. Ultimately, the goal was to provide more information that can be used to improve student success and retention in the program of computer science.

This chapter includes a review of the relevant literature on the topic, as well as a more in-depth look at the theoretical framework used as the foundation for this research. The literature reviewed in this section focuses on theories on programming aptitude, teaching methodologies in computer science classrooms, issues related to retention in computer science programs, and ways higher education institutions are working to retain more students in the computer science major. The research outlined in Chapter Two form a foundation for this research study and clarify the underlying problems found in computer science classrooms. While many of the challenges discussed in this chapter have been documented for many years, it has become evident there is a gap in the existing research for students enrolled in two-year schools and the community college environment (Clark et al., 2012).

Theoretical Framework

As outlined in Chapter One, the theoretical theory used in this study was grounded in Gardner's theory of multiple intelligences (MI). For decades, instructors have struggled with the concept of student learning styles, which can be used to describe a person's individual habits and the way he or she processes information. This concept, first presented in 1985 by Gardner, allows teachers to identify a student's particular strengths and aptitudes so teachers may present the information in their lessons in a way that will be most natural for the student to learn.

There have been many studies conducted using Gardner's theory as a means of improving the delivery of instruction in the classroom (Akkuzu & Akçay, 2011; Furnham, 2014; Gentry & Lackey, 2011; Gregory & Chapman, 2012; Latham, 1997). These studies focused on identifying which intelligences tie into academic achievement, and the use of the theory of MI has been used in computer science to help improve the academic achievement of students along with student satisfaction levels in computer science courses since the 1980s (Akkuzu & Akçay, 2011; Gentry & Lackey, 2011; Korkmaz, 2012). According to Korkmaz (2012), the original incorporation of MI into the classroom was used to help students who had trouble comprehending the entry-level concepts of algorithm courses and the logical steps needed in the programming process. Of the nine intelligences in Gardner's (1993) theory, two stand out as the most important for computer programming students: Logical-Mathematical and Visual-Spatial (Gardner, 1985; Ghazi et al., 2011; Korkmaz, 2012; Wulf, 2005). A study on the relationship between a student's perceived intelligences and his or her academic achievement by Ghazi et al. (2011) found a significant positive correlation between a student's perception of his or her intelligences in the areas of Logical-Mathematical and Visual-Spatial and his or her ability to perform well academically. However, it was also shown some intelligences do not have a strong correlation to academic achievement (Ghazi et al, 2011). The fact MI theory may not have a relationship to a student's ability to do well may be the indicator needed to show why some students strive for success in computer science but are unable to achieve a passing grade (Ghazi et al, 2011; Wulf. 2005).

Retention in computer science has become a large, nationwide problem to the extent it has become naturally accepted 30% of new majors will not stay (Hoskey & Maurino, 2011). To address this, instructors have started searching for ways to apply MI theory to courses to either tailor the learning environment toward a student's learning aptitudes or to help identify students who do not have the intelligences that match the topics of computer science (Adorjan & Friss de Kereki, 2013). A study on computer science students' critical thinking skills, it was found students who tested at high aptitudes for the Logical-Mathematical intelligence were able to learn foundational skills needed in computer programming at a high level, and "as their logical-mathematical [*sic*] intelligence levels increase, there is a parallel increase in scores related to their algorithm design skills" (Korkmaz, 2012, p. 183).

Many other instructors have also found MI theory pointed toward an answer for student success (Adorjan & Friss de Kereki, 2013; Akkuzu & Akçay, 2011; Doyle, 2011; Furnham, 2014; Ghazi et al, 2011; Latham, 1997). Instructors who changed their classroom models or teaching methods to accommodate this concept of different types of intelligences saw improvement in students' test scores and *note*d "students were more positively motivated in the science classroom through multiple intelligence teaching method" (Akkuzu & Akçay, 2011, p. 5). Akkuzu and Akçay (2011) suggested once an instructor began to teach toward a student's strengths, the student responded more appropriately and was capable of learning concepts with which he or she previously struggled.

Although the theory of MI has been used successfully, it has also created a large amount of debate around how the theory should be interpreted and applied in an educational setting (Doyle, 2011; Gardner, 2013). Gardner's theory has been criticized as being too broad to work in an educational environment or for allowing too much interpretation (Latham, 1997). The common misconception educators hold is every lesson they teach must contain all of the intelligences (Gardner, 2013; Gentry & Lackey, 2011; Latham, 1997). As Gardner himself pointed out, human intelligence is a combination of capacities, and the theory of MI helps express that understanding better than traditional numerical scales which quantify intelligence (Gentry & Lackey, 2011). The theory of multiple intelligences allows educators to see the whole person, not just his or her exam scores or test-taking abilities, “for example, some may literally see what others do not see —spatial-visual, while others may have great ability to influence people –interpersonal” (Gentry & Lackey, 2011, p. 9). It appears Gardner’s theory may help educators determine if a student has the correct abilities to learn the presented material, and instructors can use MI information about a student to help create lessons which focus on his or her strengths, but MI should not be used as a catch-all theory for improving student learning (Adorjan & Friss de Kereki, 2013; Akkuzu & Akçay, 2011; Doyle, 2011; Furnham, 2014; Gardner, 2013; Ghazi et al, 2011; Latham, 1997).

Theories of Programming Aptitude

Hoskey and Maurino’s (2011) study on success factors in advanced programming showed some professors found students entering advanced programming courses did not master the basic skills from introductory courses at a level to allow them to be successful. Oram and Greg (2011) posed the question: “Are we teaching badly everywhere?” (p. 114) in response to the world-wide problem of computer science drop-outs and poor student performance. Teachers in computer science continue to see a “double hump” (Dehnadi & Bornat, 2006, p. 17) in their statistical data for computer programming course performance (Dehnadi & Bornat, 2006).

Some students seem to be able to learn programming very quickly while the larger majority suffer and struggle (Dehnadi & Bornat, 2006). When the focus is turned on two-year college students, the problem is even more apparent (Jenkins & Cho, 2012). Since these students are most likely at a remedial math level when they enter their program in a two-year school, they will be missing key analytical skills taught in those courses, which puts them even further behind their peers (Jenkins & Cho, 2012). Simply loosening the requirements of the program will not help students succeed. Allowing students to pass through computer programming courses without a strong foundation sets them up for failure as "non-programmers can't comprehend what computers are, and crucially are not, capable of and therefore design is beyond them; and they can't comprehend what the actual difficulty of programming is, so they can't be managers either" (Bornat, 2011, p. 1).

Given that low retention and high failure rates in computer science courses have become an ongoing trend, some faculty are beginning to believe in the idea of programming aptitude (Boesch & Steppe, 2011; Bornat, 2011; Leal, 2013; Scott & Ghinea, 2013, 2014). Aptitude, as defined by Snow (1992), is "a state or condition of the person that makes it possible to engage profitably in a given learning activity" (p. 8). Through the years, as more research has been performed in an attempt to identify the issue, many educators have started to wonder if there is an issue of aptitude causing students to not be successful in computer science programs (Boesch & Steppe, 2011; Bornat, 2011; Leal, 2013; Scott & Ghinea, 2013, 2014). This issue of aptitude forces educators to view students from a different perspective and ask the question: Is this student going to be capable of learning these concepts? It also appears an answer to this question is hard to find (Boesch & Steppe, 2011; Bornat, 2011; Leal, 2013; Scott & Ghinea, 2013, 2014). As more educators work to increase program retention, instructors have started examining ways to screen students for the right kind of aptitude to determine if they have a high probability for success before the students begin the program (Scott & Ghinea, 2013).

The question of aptitude requires the skills, traits, and concepts necessary for a student to be successful in computer programming are first identified. Some predictions of aptitude are taken from high school grades and standardized test scores in areas such as math since math scores in high school have been previously determined to be good indicators of success in college computer science classes (Coyle, Purcell, Snyder, & Richmond, 2014; Leal, 2013). Research by Coyle et al. (2014) and Walker, Hirakawa, and Steinbach (2011) explored the ability of Scholastic Aptitude Test (SAT) and American College Testing (ACT) scores to help determine if a student would excel in a particular college major. Based on results from Coyle et al. (2014) and Walker et al. (2011), students who could excel in computer science could be identified through math and verbal scores.

Further research has also shown students who excel at mathematical thinking excel at computational thinking (Hu, 2011). In a 2008 study on psychological predictors, Erdogan et al. found five indicators of success for students in computer programming courses: problem solving ability, motivation, learning style, previous experience, and gender. With this information, colleges are beginning to experiment with identifying students who may not fit the right profile for the computer science field to help guide students in their choice of major as well as a mitigation step to decrease the student attrition rates in these programs (Adadi et al., 2014; Avancena et al., 2012; Boesch & Steppe, 2011; Leal, 2013; Scott & Ghinea, 2014; Teague & Lister, 2014).

Educators have worked to create aptitude tests or other screening methods to predict student success in computer science courses (Adadi et al., 2014; Avancena et al., 2012; Boesch & Steppe, 2011; Leal, 2013; Scott & Ghinea, 2014). As researchers continue to study preparedness of students for entry level computer science courses, they have discovered students share "rational misconceptions" (Bornat, 2011, p. 3), make the same mistakes as their peers, and have the same problems with key concepts. This pattern that students exhibit of logical fallacy helps guide entry-level computer science instructors towards an idea for an aptitude test that checks for these misconceptions in understanding and logic (Bornat, 2011). The problem educators seem to encounter when creating these tests is not only the need for clear indicators of what a student knows, they needed to know how they can tie that information back to the concepts a student needs to understand or be capable of learning (Bornat, 2011; Scott & Ghinea, 2014). These aptitude tests are developed in a way which evaluates a student's cognitive abilities for pattern recognition, numerical relationships, spatial reasoning, and pattern identification (Scott & Ghinea, 2014). Aptitude exams are now in the form of traditional pencil and paper assessments, the ACT, exams made of instructor-written questions, hands-on exams in which students must decipher or work with code, and simulations that take on the form of games or interactive programs (Adadi et al., 2014; Avancena et al., 2012; Boesch, 2011; Leal, 2013; Scott & Ghinea, 2014).

The main problem with aptitude tests seems to be one of reliability (Adadi et al., 2014; Bornat, 2011; Leal, 2013; Walker et al., 2011). Many researchers have found mixed results when using aptitude tests (Adadi et al., 2014; Bornat, 2011; Leal, 2013) and have not always accurately determined if a student would succeed or fail in learning computer programming. It has been found "predicting programming aptitude is still a challenging task, although this topic is being studied for more than 40 years, and its relevance has been well establish for almost a quarter of a century" (Leal, 2013, p. 2). Placement systems that rely on exams, such as the ACT, have to take into account students who are not good test takers and other factors, and "they cannot ascertain intangible elements, such as student interest, motivation, priorities, and competing time commitments" (Walker et al., 2011, p. 26). There does not appear to be a lack of research in the area of aptitude testing, but there is a lack of clear results on whether or not these aptitude tests actually work in predicting which students will be successful in their programming courses.

Another method available to determine if a student is prepared for computer science courses is faculty evaluation (Walker et al., 2011). In this method, students are tested using a previously-designed aptitude test that includes an interview component (Walker et al., 2011). Students are then allowed into a major based on the recommendation of the faculty member who conducted their review (Walker et al., 2011). This interview process has almost as many flaws as the aptitude exams themselves which "requires significant faculty time and also is subject to variation according to the individual faculty members involved" (Walker et al., 2011, p. 26).

Student Preparedness for Community College

Many studies have been conducted on the level of preparedness of community college students (Barbatis, 2010; Barnett, 2011; Goldrick-Rab, 2010; Stigler, Givvin, & Thompson, 2010; Yates, 2010). Since community colleges were originally designed to be open-enrollment institutions, the students who enter into degree programs often start at a disadvantage in comparison to their four-year university counterparts since they have typically been out of high school for several years before attempting college (Barbatis, 2010; Barnett, 2011; Goldrick-Rab, 2010; Stigler et al., 2010; Yates, 2010). These students often have to start in developmental level classes to learn or recover skills in math, reading, and writing, which will put them at the college level of work expected in the courses required for their degree program (Barbatis, 2010; Jenkins & Cho, 2012; Stigler et al., 2010). Community colleges are having to take on the burden of providing comprehensive developmental remediation to a significant number of students (Yates, 2010) due to the number of students who enter at a developmental level as a result of the open-enrollment nature of community colleges.

Stigler et al. (2010) noted the problem is at its most severe in mathematics. Where students enter the college far behind where they need to be to complete their degree in two years; moreover:

> Most colleges have a sequence of developmental mathematics courses that starts with basic arithmetic, then goes on to pre-algebra, elementary algebra, and finally intermediate algebra, all of which must be passed before a student can enroll in a transfer-level college mathematics course. (Stigler et al., 2010, p. 2)

Developmental placement may mean a student must take several years of development courses before beginning the courses in the student's chosen major, which puts the student on a four-year plan at a two-year school (Stigler et al., 2010; Yates, 2010).

A longer degree path puts community college students in a higher category of risk to drop-out since it has been found if it takes a student three years to complete a two-year degree, only 16% will be successful (Goldrick-Rab, 2010). There is also the added strain of finances and personal responsibilities since community college students are often non-traditional students with families, outside work, and financial responsibilities. The longer an adult student is in school, the more strain is placed on meeting all of these personal responsibilities and the higher likelihood something will occur that will prompt the student to drop out (Goldrick-Rab, 2010; Jenkins & Cho, 2012).

With so many students attending community college unprepared for the mathematics coursework required in a computer science major, it follows that many of the students who intend to major in computer science may never reach an entry-level computer programming course since they are unable to make it through the developmental math courses set as prerequisites (Goldrick-Rab, 2010; Jenkins & Cho, 2012). Since many community college students may not yet have clear goals for college or a career, more time could be spent taking classes in many areas before the student settles on a particular career path (Jenkins & Cho, 2012). Since research shows the longer a student is enrolled the less likely he or she will complete a degree, it becomes a race between a student's incoming level of skill in the key developmental areas of reading, writing, and math and the prerequisites needed to enter a desired major once the student has chosen one (Goldrick-Rab, 2010; Jenkins & Cho, 2012; Yates, 2010).

Community colleges have been working to address the problem of student preparedness with reform efforts such as Achieving the Dream (ATD), which works to improve developmental education outcomes (ATD, 2015). However, reforms implemented through the ATD program have not yet resulted in an improvement in community college completion rates (ATD, 2015; Jenkins & Cho, 2012). To compound the problem, students who enter community colleges unprepared are not just non-traditional students who have been out of school for years, but are recent high school graduates found to be lacking in key areas of reading and mathematics, which are needed to be successful (Goldrick-Rab, 2010; Melguizo, Bos, & Praather, 2011; Stigler et al., 2010). Research into this problem found students graduating from high school who were "not able to perform basic arithmetic, pre-algebra and algebra, [which] shows the real cost of our failure to teach mathematics in a deep and meaningful way in our elementary, middle and high schools" (Stigler et al., 2010, p. 3). Ultimately, if the methods used to teach mathematics to middle school and high school students fails, the students coming into community college math classes will be pre-disposed to years of failure and discouragement towards the subject (Melguizo et al., 2011).

Teaching Methodologies in Computer Science

For many years, educators in computer science have tried to find ways to make programming more appealing and engaging (Ambrósio et al., 2011; Hoskey & Maurino, 2011; Kay, 2011; Nilsen & Larsen, 2011; Porter et al., 2013a, Porter et al., 2013b). Studies such as, *Identifying Cognitive Abilities to Improve CS1 \Outcome,* by Ambrósio et al. (2011), looked at failure rates in computer science classes and tried to identify teaching methodologies that helped students at risk of failing. Since computer programming requires the knowledge of many tasks in different knowledge domains, such as critical reasoning, problem identification, procedural thinking, and reading comprehension, many instructors have sought to determine a student's skills in these areas to see where he or she needs improvement (Hoskey & Maurino, 2011; Kay, 2011; Porter et al., 2013a, Porter et al., 2013b). As research into ways to make computer programming classes more appealing and engaging has progressed, new methodologies for teaching computer programming have been developed (Ambrósio et al., 2011; Hoskey & Maurino, 2011; Kay, 2011; Porter et al., 2013a, Porter et al., 2013b). These methodologies have been categorized as: Group-focused, Programming Language, Activity-based, and Game-focused.

Group-focused methodologies. Group-focused methodologies teach strategies which require students to work in pairs or teams (Braught et al., 2011; Maguire, Maguire, Hyland, & Marshall, 2014; Porter et al., 2013a, Porter et al., 2013b; Radermacher, Walia, & Rummelt, 2012). Pair Programming or Peer Programming has become a key element of programming classrooms focusing on group-work; it is designed to give students a greater understanding of programming skills and increase course completion rates (Braught et al., 2011; Maguire et al., 2014; Porter et al., 2013a, Porter et al., 2013b; Radermacher et al., 2012). Through Pair Programming, students are able to meet more people in their computer science classrooms, which creates a greater social tie and support structure for students who feel isolated and alone (Maguire et al., 2014). Another benefit of the Peer Programming methodology is practice in collaborative environments that mimic real-world computer programming practices since most problem-solving tasks are not completed by a sole programmer while on the job (Maguire et al., 2014). Some disadvantages to the Pair Programming model have been identified (Radermacher et al., 2012). Creating effective pairs can become a difficult and time-consuming job for the instructor since not all students approach or solve a problem the same way (Radermacher et al., 2012). A student has to overcome issues of communication and be able to transfer knowledge to his or her pair partner, information the student may not yet fully grasp, which creates a situation where a pair will not be able to solve the task at hand (Radermacher et al., 2012).

Another Group-Focused Methodology is Extreme Apprenticeship, which is similar to the Pair Programming method since students work with others, but differs in that students may work with a larger group, and feedback is provided by the instructor continuously throughout the programming process (Vihavainen et al., 2011). This method is more like a traditional apprenticeship model, where a student learns from a master as the work is completed (Vihavainen et al., 2011). During this learning process, there is continuous communication and feedback between the student and the instructor so the instructor can gauge a level of mastery (Vihavainen et al., 2011).

A third type of Group-focused methodology is Peer Instruction (PI), where students lead discussions and solve three to five questions in pairs or groups with the instructor in the role of guide (Porter et al., 2013a). The intent of PI is to create a collaborative environment that moves away from the lecture format and encourages students to be more engaged in the learning process (Porter et al., 2013a). Although some students are grade-competitive and sometimes have trouble learning to work with others on graded work, students are guided into thinking about their future software development teams and working on gaining the skills needed in a collaborative work environment (Porter et al., 2013a, Porter et al., 2013b).

Research on Group-focused methodologies has shown these methods have increased retention and the number of students with passing grades in computer science courses (Braught et al., 2011; Maguire et al., 2014; Porter et al., 2013a, Porter et al., 2013b; Radermacher, 2012). The use of Pair Programming and PI demonstrated a strong increase of student learning outcomes (Braught et al., 2011; Porter et al., 2013a, Porter et al., 2013b). Group-focused methodologies also helped show students computer science was not the asocial career choice portrayed by the media and popular culture (Porter et al., 2013a, Porter et al., 2013b).

Programming language methodologies. One of the biggest questions educators often face in their entry-level computer programming courses is which programming language to teach (Lack et al., 2013). Many of the programming languages used in the work-place, such as C++ or Java, are complex and not always easily comprehensible to a new learner (Hoskey & Maurino, 2011; Lack et al., 2013; Meerbaum-Salant et al., 2013). Educators in entry-level computer science courses have found using programming languages that are more English-like in their syntax are easier to teach to new students (Lack et al., 2013). When teaching new students in computer programming "the most difficult and time-consuming part of learning to program is learning the syntax and semantics of a programming language" (Lack et al., 2013, p. 2). However, educators do not agree on a single programming language that works well for teaching new computer science students. (Hoskey & Maurino, 2011; Lack et al., 2013; Meerbaum-Salant et al., 2013; Tillmann et al., 2012a).

Educationally-focused programming languages are another option for computer instructors. Educators have worked to create a more engaging learning environment in computer science through the use and development of special programming languages designed for teaching (Lack et al., 2013; Meerbaum-Salant et al., 2013; Tillmann et al., 2012a). Educationally-focused programming languages have been developed under the assumption that some students find computer programming languages used in the industry too difficult to learn (Lack et al., 2013; Meerbaum-Salant et al., 2013; Tillmann et al., 2012a). The programming languages designed for education give the learner a simple introduction into programming concepts so they can work up to a more sophisticated language (Lack et al., 2013; Meerbaum-Salant et al., 2013; Tillmann et al., 2012a). Two popular educationally-based programming languages are Scratch and Grace (Meerbaum-Salant et al., 2013; Tillmann et al., 2012a).

Along with these education-based programming languages have come visual programming environments such as Alice, which gives the students a drag-and-drop interface to work with while learning basic concepts of computer programming theory, such as looping and decision making (Alice Educational Software, 2015; Meerbaum-Salant et al., 2013). The use of visual programs, such as Alice, which allow students to tell stories with interesting characters, has been shown to increase a student's motivation in computer science classes (Alice Educational Software, 2015; Meerbaum-Salant et al., 2013). Due to students' positive responses, many universities have started using Alice in entry-level computer science courses (Davies, Polack-Whal, & Anewalk, 2011; Meerbaum-Salant et al., 2013).

One educationally-focused language that has become popular in introductory computer programming courses is Scratch. Scratch was developed to give new programmers a way to learn and practice computer programming tasks with immediate visual feedback (Davies et al., 2011; Meerbaum-Salant et al., 2013). Scratch has been used in computer science departments to help teach students introductory concepts of computer programming before being formally introduced to a computer programming language (Meerbaum-Salant et al., 2013).

Other programming languages and visually-based development environments have been created to help educators provide a more accessible learning environment for students in introductory computer programming courses (Lack et al., 2013). Grace is a computer programming language specifically designed by a group of computer science educators to address the need for an entry-level computer programming language that teaches specific computer programming concepts without overwhelming the student (Lack et al., 2013). Some of these environments are more like simulations, rather than actual coding, and continue to raise the debate about which one should be used (Hoskey & Maurino, 2011).

As educators continue to study the problem of entry-level programming language and student comprehension, conflicting studies on whether educationally-designed programming languages help students learn computer programming for the workforce continue to emerge (Hoskey & Maurino, 2011; Lack et al., 2013; Oram & Greg, 2011). The use of educationally-design programming languages and simulations has been widely debated, with some educators thinking they should not be teaching a computer programming language a student will never be able to use in a real-world setting (Oram & Greg, 2011; Kay, 2011). Other educators feel by using an educationally-designed language, instructors are able to work students up to the level of a more complex language and allow students more success, which helps keep retention within computer science programs high (Lack et al., 2013; Meerbaum-Salant et al., 2013; Tillmann et al., 2012a).

Activity-based methodologies. Activity-based teaching methodologies focus on the student working on a particular activity, or a set of activities, which help guide learning of course concepts (Hazzan et al., 2011). The concepts the instructor wishes the student to master are presented as a set of activities, case studies, or challenges, often using the Active Learning Based Teaching Model, which is a four-stage teaching methodology composed of: trigger, activity, discussion, and summary (Hazzan et al., 2011). Activity-based methodologies are based on the idea of active versus passive learners where a student's disposition as either an active or passive learner will help the student learn more efficiently in some situations rather than others (Felder & Silverman, 1988).

Instruction on how to complete tasks or formulate the concepts needed for the instructional activities are either presented through traditional lecture instruction or can be presented using video clips (Amresh, Carberry, & Femiani, 2013; Manley & Urness, 2014). Many times, Activity-based teaching methodologies are used in the format of inverted, or flipped-classrooms, which involves presenting materials to students in a digital format so in-person class time can focus on lab activities (Manley & Urness, 2014). In the inverted format, students are required to take the initiative in their learning by watching videos, taking quizzes, or completing reading assignments outside of class on their own schedule (Manley & Urness, 2014).

By using videos in a flipped classroom environment, studies showed higher average scores and better student engagement in entry level computer programming classes (Amresh et al., 2013; Manley & Urness, 2014; Mok, 2014). Moreover:

> The videos also appeared to have a bigger impact on changing student attitudes about whether programming is fun. Student comments and instructor experiences indicated that the video-based instruction led to more meaningful interaction

> between the instructor and students while working on lab assignments. (Manley & Urness, 2014, p. 6)

Students in these studies also reported programming using activity-based assignments and lessons in a flipped classroom gave them more time to prepare, and they liked the fact they could use their class time for more engaging activities (Mok, 2014). The Activity-based format has also been found to help students who study slower or need more time with the coursework and "enabled weaker but diligent students to study at their own pace and come to class as prepared as their stronger contemporaries" (Mok, 2014, p. 10).

A further method for Activity-based methodologies, combined with the Group-focused methodology, is the use of computer simulation environments and three-dimensional worlds, such as Second Life (SL) (Esteves, Fonseca, Morgado, & Martins, 2011; Linden Labs, 2015). Second Life provides a persistent three-dimensional world where students can connect, collaborate, and get immediate feedback on the work they are producing (Esteves et al., 2011). Students interact with the SL program world via their avatar, which is a digital projection of their character (Linden Labs, 2015). Groups of avatars can work together on activities since multiple avatars can edit the same object simultaneously (Esteves et al., 2011). The SL environment also allows for collaboration asynchronously since the SL world is persistent, allowing students to log in and work when their schedules allow (Esteves et al., 2011; Linden Labs, 2015).

New technology is being developed which will allow instructors to bring virtual technology into classrooms with products like Oculus Rift and Microsoft Hololens (de Castell, Jenson, & Larios, 2015; Kossey, Berger, & Brown, 2012). These products allow the instructor to create a virtual environment with which the student can interact (de Castell et al., 2015; Kossey et al., 2012). As this technology develops, the potential for interaction between students who are not physically present in the classroom will become more of a reality. Virtual reality technology and the use of computer simulation environments allow for the implementation of labs which allow for tactile contact and manipulation in areas where this was not possible or feasible in the past (de Castell et al., 2015). Virtual reality products like the Oculus Rift and Microsoft Hololens help remove some of the limitations found in virtual environments run on a computer (de Castell et al., 2015; Kossey et al., 2012). These virtual reality tools like the Oculus Rift and Microsoft Hololens are also being used to help students learn problem-solving skills and discover new ways to assess and train spatial abilities (de Castell et al., 2015; Kossey et al., 2012).

Research into activity-based methodologies has shown students preferred hands-on activities and the structure of lab work to lecture-based teaching (Felder & Silverman, 1988; Hazzan et al., 2011; Manley & Urness, 2014), and Activity-based methodologies have also shown many advantages to the traditional style of teaching when working with the concepts found in computer science (Hazzan et al., 2011). Some of these advantages were found to be higher-level skills in analysis, synthesis, and evaluation of tasks in a field which required large amounts of each of these skills (Hazzan et al., 2011; Manley & Urness, 2014).

Game-focused methodologies. Researchers have been looking at ways to teach the concepts of computer programming to under-represented groups, such as women and children (Carbonaro, Szafraon, Cutumisu, & Schaeffer, 2010; Kazimoglu, Kiernan, Bacon, & Mackinnon, 2012; Tillmann et al., 2012a). New teaching methodologies have arisen from the efforts of these researchers to make computer science more appealing to younger students and females as well as to help retain students in the computer science major (Kazimoglu et al., 2012). Due to the success of game-focused methodologies, many are now being used in post-secondary classrooms (Carbonaro et al., 2010). Game-focused methodologies use the same concepts found in computer programming classes to create video games, and virtual environments leverage the mass appeal of video games (Basawapatna, Koh, & Repenning, 2010; Carbonaro et al., 2010).

Research into using Game-focused teaching methodologies has found students from middle-school to the graduate level found more interest in programming when it was presented as a video game focused projects than when students were solving more business-related problems (Basawapatna et al, 2010). Software programs, such as Alice, developed by Carnegie Mellon University, have been designed specifically for teaching students programming concepts through video games by providing a visual environment that helps students learn game design principles, as well as exercise the logical skills of computer programming (Alice Educational Software, 2015; Carbonaro et al., 2010; Kazimoglu et al., 2012). The Alice programming environment allows for a simplified look at three-dimensional animated video games through a drag-and-drop interface where students construct scenes and tell a story (Alice Educational Software, 2015; Carbonaro et al., 2010). Once students gain an understanding of the logical concepts needed in video game programming, they can move on to more sophisticated environments, such as AgentSheets, originally designed at the University of Colorado in Boulder, which can be used with traditional programming languages to create games (Basawapatna et al, 2010). AgentSheets moves students into learning a programming language they would later use in their career field, and not a constructed language only used in education (Basawapatna et al., 2010).

Microsoft has been helping teachers engage computer science students through the idea of teaching computer programming through games and through the paradigm of mobile application development (Tillmann et al., 2012b). Since students are more likely to own a smartphone or tablet device rather than a laptop, finding ways for students to learn software development using the devices they already own has become an area of focus (Tillmann et al., 2012b). TouchDevelop is a program designed by the Microsoft Research team designed to be used to write code in a mobile environment without the need for a traditional computer (Tillmann et al., 2012b). TouchDevelop uses its own programming language and not a standard programming language students would use in a work environment (Tillmann et al., 2012b).

Working in conjunction with the mass appeal of video games is the appeal of social sharing between students (Basawapatna et al, 2010; Tillmann et al., 2012b). Programs like AgentSheets and TouchDevelop are designed with social aspects, so students can share what they have created with other students and developers (Basawapatna et al, 2010; Tillmann et al., 2012b). Game sharing takes place on websites provided by the program developers and allows students to download other people's shared games and write comments (Basawapatna et al, 2010; Tillmann et al., 2012b). The sharing and social built into AgentSheets and TouchDevelop creates another layer of engagement for students who are working in classrooms utilizing these teaching programs (Basawapatna et al, 2010; Tillmann et al., 2012b).

The success of using Game-focused teaching methodologies has been validated through research (Basawapatna et al, 2010; Carbonaro et al., 2010; Kazimoglu et al., 2012; Tillmann et al., 2012). Studies on Game-focused methodologies have found students learning the concepts of computer programming while making games were more engaged and learned the higher-order thinking skills needed to be successful in more rigorous courses (Basawapatna et al, 2010; Carbonaro et al., 2010; Kazimoglu et al., 2012; Tillmann et al., 2012). By allowing students to use video game creation as an entry-point into computer programming, students are finding more success in their achievements of building a game than they were in building traditional business-focused applications (Basawapatna et al, 2010). Students are finding computer programming can be fun, which can lead to greater student retention and more students who will ultimately take another programming course with confidence in their growing skills (Basawapatna et al, 2010; Carbonaro et al., 2010). In spite of these innovations in teaching computer science students, retention remains a problem (Basawapatna et al, 2010; Carbonaro et al., 2010).

Retention in Computer Science Programs

Throughout the review of research, the over-arching topic of retention remained a constant. Typically, community colleges have to rely on data from four-year institutions for retention since the amount of research conducted in a two-year environment is very limited (McClenney et al., 2012). For community colleges, there is a more diverse mix of students with varying needs and skill levels, which makes retention problems more challenging (McClenney et al., 2012).

One of the key problems community colleges have is the issue of remediation. Students who enter community colleges typically have a long list of remedial courses they must complete before they are ready to join their intended major (Jenkins & Cho, 2012; Scott-Clayton, 2011). The Community College Survey of Student Engagement (CCSSE) provides tools for administrators and educators in two-year learning environments and helps track, monitor, and identify issues of student success (McClenney et al., 2012). The number of students who start at a community college and never finish continues to alarm administrators (Jenkins & Cho, 2012; McClenney et al., 2012). Fewer than 36% of first-time college students at a two-year institution earned a postsecondary credential within six years of enrollment (Jenkins & Cho, 2012). This issue of retention in community colleges only seems to become more pronounced in programs that are traditionally academically rigorous, such as computer science (Jenkins & Cho, 2012; McClenney et al., 2012). Not only do computer science programs require a lot of analytical and critical thinking skills, but the majority of incoming computer science students lack the foundational math skills shown by researchers to help make students more successful in learning the concepts of computer programming (Hoda & Andreae, 2014; Watson & Li, 2014).

For many schools, retention rates for first-year programming courses are the lowest on campus when compared to other programs (Haungs, Clark, Clements, & Janzen, 2012). It is not uncommon to find the pass rate for the first computer science course at a school to be around 67% (Watson & Li, 2014). However, this is not only an issue of retention at the beginning of the computer science degree program but throughout the student's time at the school and into the job market (College Board, 2013). As students continue to find computer programming courses difficult, and instructor efforts to teach the subject fail, the field of computer science is seeing a large decrease in students who complete degrees and enter the field (College Board, 2013). According to College Board, the organization that administers and tracks AP exams, 31,117 students took the Computer Science AP exam in 2013. The slightness of this number is clear when it is compared to the 476,277 students who took the English Literature exam; 442,890 students who took the U.S. history exam; or 108,219 students who took the exam for Macroeconomics (College Board, 2013).

The topic of retention has been discussed and studied for many years while instructors worked on ideas to help engage and keep students in their classes (DeClue, Kimball, Lu, & Cain 2011; Haungs et al., 2012; Reese et al., 2013; Rizvi, Humphries, Major, Jones, & Lauzun, 2011; Stewart-Gardiner, 2011). Many instructors feel there are several areas to be explored in the problem of computer science retention, with the top three being: students whose interest or ability do not match with the curriculum, students who do not understand the field of computer science, and students who begin their majors and transfer out to other departments as they find additional topics that spark their interest (DeClue et al., 2011; Haungs et al., 2012; Reese et al., 2013; Rizvi et al., 2011; Stewart-Gardiner, 2011).

Many strategies have been implemented in order to increase the retention of computer science students, and "over the past 25 years, the total number of students in college has increased by about 50 percent. But the number of students graduating with degrees in STEM subjects has remained more or less constant" (Tabarrok, 2012, p. 1). Use of collaborative learning strategies in the classroom and the implementation of a programming teaching concept called Pair Programming are showing the best results in helping students who are having difficulties adjusting to the rigor of the computer science courses (Braught et al., 2011; DeClue et al., 2011; Hakimzadeh, Adaikkalavan, & Batzinger, 2011; Porter et al., 2013a; Porter et al., 2013b).

Through the use of methodologies such as PI, instructors at the University of San Diego were able to halve the failure rates of students in their introductory computer science courses and help students build a support network of peers they could use when they had difficulties with the material (Porter et al., 2013a). Peer Instruction also removed some of the negative feelings students had about grade competition and helped move them into a mindset of collaboration and working together as a team (Porter et al., 2013b). During Porter et al.'s (2013a) study on the use of PI in introductory computer science courses, it was found the use of this method of instruction reduced failure rates by 15%. The success of PI was supported by the CCSSE report, which showed support for learners had the greatest impact on a student's persistence in their courses (McClenney et al., 2012). Pair Programming is a similar concept being used in the classroom. Pair Programming got its start in Agile, a computer software development methodology, and "students using pair programming in upper-division computer science courses produced higher-quality programs and learned the material faster" (Porter et al., 2013b, p. 1). Pair programming also helps students who are more social than the stereotypical asocial programmer (Porter et al., 2013b). Computer programming is seen as a singular task of "sitting in the corner and hacking for hours on end" (Porter et al., 2013b, p. 2), and it helps students who are truly feeling lonely make a connection in their courses.

Some computer science instructors have looked to students for guidance on what they could be doing better (Blaho et al., 2012; Piterira, 2012). In a study on novice programmers in Portugal, the instructors conducted a survey of the students to help identify their difficulties and perceptions of their computer programming courses (Piterira, 2012). Through the use of these surveys, the researchers were able to identify topics students did not comprehend or felt they did not have a clear understanding of and were able to direct their teaching efforts in a more focused way (Piterira, 2012). The survey was typically given at the beginning of the semester and then periodically throughout the course so the instructor could use the information to manage the delivery of the curriculum and see where students were really struggling (Blaho et al., 2012). This survey should be different from the course evaluation used at the end of the semester to rate the course and the instructor (Blaho et al., 2012).

Another way instructors have worked to improve retention and interest in computer science is by implementing games, activities, and programming languages designed to help improve learning in their courses (Rizvi et al., 2011). This is due to research being conducted on student retention that pointed to the difficulty of the course being one of the main factors students did not succeed in computer science courses. Some researchers suggested the "introduction of the object-oriented (OO) paradigm in CS1 courses [entry-level computer programming courses] might have been a contributing factor in low enrollment" (Rizvi et al., 2011, p. 2). Creating curriculum for entry-level computer programming courses designed for better student retention and completion has given rise to computer programming languages that have been developed for the sole purpose of helping teach students computer programming in an easier to understand environment (Rizvi et al., 2011). Scratch was developed by the Lifelong Kindergarten group at the MIT Media Lab in partnership with the UCLA Graduate School of Education and Information Studies and was designed to help students age eight and up learn programming skills that could then be transferred into real-world programming environments (Rizvi et al., 2011).

Students who do not understand the major of computer science is a common occurrence, and instructors who ask their students to define computer science have found students have very different answers (Reese et al., 2013). Students may have interest in computer science, but due to the discrepancy in student minds about the concepts needed and their abilities, they often become discouraged or disillusioned about the skills they need to acquire to be successful (Reese et al., 2013). To combat this misunderstanding, some instructors have researched the creation of interest-based introductions to computer science that allow the student to find a related topic that interests them and then incorporates the necessary concepts (Haungs et al., 2012; Piterira & Costa, 2012; Reese et al., 2013). Focusing classwork in this way allowed students to learn the fundamentals of computer science and computer programming in a way that was personally appealing (Haungs et al., 2012; Piterira & Costa, 2012).

A side-effect of interest-based curriculum is that students are not only choosing the class topic that interests them the most, they are also choosing their learning communities since these classes will all be composed of like-minded students (Haungs et al., 2012). These learning communities easily adapt into peer study groups who have a higher interest in the course topic than in the traditionally structured computer science course (Haungs et al., 2012). At California Polytechnic, researchers Haungs et al. (2012)

created a study from their four-track introductory computer science course that allowed students to choose between the topics of robotics, gaming, music, and mobile apps. By allowing students to work on projects that followed personal interests, a 12% increase in the number of students who chose to move on to the next computer science course was reported. The number of students who earned an A in their introductory computer science course increased by 10% (Haungs et al., 2012).

Attendance policies in colleges. One of the ways colleges try to increase student retention is through close monitoring of class attendance (Leufer & Cleary-Holdforth, 2010), and there have been many studies about the effect of absenteeism and student performance (Adair & Swinton, 2012; Leufer & Cleary-Holdforth, 2010; Macfarlane, 2013; Marburger, 2006; Weimer, 2012). While the policies at most colleges and universities state class attendance is compulsory, some institutions, such as Michigan State University, give professors some freedom to implement their own class attendance policies (Macfarlane, 2013). These policies can mean the door to the classroom may be locked after the first 10 minutes of class so late students cannot enter, or it may mean student are not allowed to make-up assignments from a missed class period (Macfarlane, 2013). In a study that looked at different university policy statements on compulsory student attendance, Macfarlane (2013) stressed, while absence is bad for a student's academic performance, many professors feel holding students accountable for the consequences accrued due to absences is ultimately a better lesson (Macfarlane, 2013). The opposite view of the attendance policy argument is held by the students who see themselves as fee-paying customers who should not have to attend if they do not want to attend (Macfarlane, 2013).

Due to the inverse relationship between attendance and student performance, some institutions are going a step further by implementing mandatory attendance policies with consequences if not followed (Institution Course Catalog, 2015; Weimer, 2012). Studies showed when a mandatory attendance policy was in place, more students attended, but the outcome of that attendance was conflicting (Marburger, 2006; Weimer, 2012). In a study comparing attendance between classes with different attendance policies, Marburger (2006) found, "by the final third of the semester, the daily percentage of students who were absent in the no-policy class more than doubled. In sharp contrast, daily absenteeism in the policy class remained fairly constant throughout the semester" (p. 154). However, Weimer (2012) noted although "students who get better grades are also more likely to be students who come to class, [this] does not mean that making students come to class will result in better grades for them" (p. 40).

Aside from the issue of student retention are new mandates from the Department of Education for Higher Education requiring institutions who receive financial aid to keep detailed records of student attendance (Harvey, Bright, & Wamsley, 2014). Mandatory tracking of attendance for all students is becoming the new norm for institutions in order to comply with these regulations (Harvey et al., 2014), and "institutions that are required to take attendance are expected to have a procedure in place for routinely monitoring attendance records to determine in a timely manner when a student withdraws" (Federal Student Aid, 2014, p. 5). Submission of student attendance data is the responsibility of institutions, and since the data are already being collected, many schools, such as Tarrant County College in Texas and Berkeley College in New York, have chosen to go forward with consequences to students who are not attending classes (Berkeley College Student Handbook, 2015; Tarrant County College Attendance Policy, 2015).

The consequences institutions are creating in response to new financial aid policies for 2014-2015 are strict (Federal Student Aid, 2014; Harvey et al, 2014). Not only will students be required to attend classes or be removed, but they will also have to pay back any federal aid funds they had received for the course (Federal Student Aid, 2014; Harvey et al, 2014; Jesse, 2015). Students who are not attending their courses will be required to be removed from the course by the institution, and any disbursed financial aid funds to the student will be required to be returned to the school, which will return the funds to the Department of Education (Federal Student Aid, 2014; Harvey et al, 2014). Higher education institutions will be required to repay federal funds received based on the student absences reported (Federal Student Aid, 2014; Harvey et al, 2014). Since the outcome of not keeping good attendance records could result in a very large bill from the Federal Student Aid department for funds that should not have been given to students who were not attending their classes, higher education institutions are taking their new attendance mandates very seriously (Federal Student Aid, 2014; Harvey et al, 2014; Jesse, 2015).

One of the outcomes of mandatory attendance policies is a drop in the number of students who can ultimately complete a course with a passing or failing grade. For many of the colleges that have implemented mandatory attendance policies, academic progress does not always factor into letting an absent student stay in the course (Archambault, Kennedy, & Bender, 2013; Macfarlane, 2013). For example, students may have a passing grade but could be removed from the class for missing too many class periods

(Archambault et al., 2013; Macfarlane, 2013). There are further implications of mandatory attendance policies since schools are seeing a decline in the number of students who fail a course (Archambault et al., 2013; Macfarlane, 2013). Ultimately, students who were in a situation to fail the course often stop attending and are removed via the mandatory attendance policy process (Archambault et al., 2013; Macfarlane, 2013). It is typically the institution's view that a drop or withdraw mark on the student's transcript is better than a failing grade since failing grades have a negative effect on a student's grade point average (Harvey et al., 2014; Marburger, 2006; Weimer, 2012).

Another reason schools are starting to add mandatory attendance policies is due to recognition through the ATD program (ATD, 2015; Special Assistant to the Provost, G. O'Connor, personal communication, June 12, 2015). At the institution where this research was performed, a mandatory attendance policy was in place since the school wanted to align to the guidelines developed by ATD in pursuit of the foundation's recognition (Special Assistant to the Provost, G. O'Connor, personal communication, June 12, 2015). The program was founded in 2004 by the Lumina Foundation with the goal of helping set guidelines community colleges should follow to help improve student success at their institutions (ATD, 2015). The ATD foundation, as a reform movement, outlined mandatory attendance policies as one of the initiatives community colleges should create (ATD, 2015).

In the guidelines for a mandatory attendance policy, students should not be permitted to miss more than 25% of the total time allocated for classes and should be withdrawn from the course if they do so (ATD, 2015). Community colleges are encouraged to implement an attendance policy through the ATD program to help more students complete their courses and, ultimately, their degree (ATD, 2015). By making students attend class, college administrators hope students make connections with their instructors and ask for help when they need it (ATD, 2015).

Community colleges are eligible for benefits by creating policies outlined by the ATD foundation, such as getting access to special grant opportunities and sharing data with other institutions for reporting and benchmarking (ATD, 2015). Coaching and support services are also provided by ATD to help community colleges meet accreditation and state performance standards (ATD, 2015). Since the ATD foundation recommended policies on mandatory attendance align with the new federal mandates for institutions that receive federal financial aid, it makes sense for community colleges to pursue the ATD standards for their support and the potential for return on investment through increased student success (Special Assistant to the Provost, G. O'Connor, personal communication, June 12, 2015).

Programming as a Hobby

Before computers were mainstream, they were thought of as a hobby (Levinson, 1995). Computers were something for people to tinker with in their garage and go to conventions to talk about (Levinson, 1995). It was part craft and part obsession for the teens who started in the early years of computers (Haonkonen, 2013; Levinson, 1995). Computer hobbyists would create programs in their basements and trade them to friends who were also into this new hobby (Atwood, 2007). The argument can be made that in order to use a computer back in those days, some fundamental knowledge of how computers worked was required and even more knowledge was needed in order to program one (Atwood, 2007). Computers were not as easy to use as they are now, and a basic understanding of coding and troubleshooting was necessary to get anything done (Atwood, 2007; Haonkonen, 2013), which is not the problem in today's society where technology is easier to use than ever.

Today, the idea computer programming is not just for people going into the field of computer science is becoming more prevalent (Pollack, 2014; Shein, 2014). Since society is now an extremely technology-powered environment, there has been a call for basic code literacy (Pollack, 2014; Shein, 2014). The trend for general programming skill is pushing more people, who would not have otherwise taken a computer programming course, into computer science classrooms, where they are working alongside computer science majors (Pollack, 2014; Shein, 2014).

The reasoning does not indicate everyone should be able to write a computer program; however, through the learning process of computer science courses, students learn problem-solving skills and critical thinking skills needed in many different fields (Pollack, 2014; Shein, 2014). America's leaders, such as President Barak Obama, have stepped forward with this message as well (NPR Staff, 2014). President Barak Obama encouraged educators and students with a talk on technology and education in 2014, and during his talk with educators on computer skills, the President wrote his first computer program to demonstrate his commitment to learn about our technology-heavy society (NPR Staff, 2014). House Majority Leader Eric Cantor made the statement computer coding was "the necessary tool of this century" (NPR Staff, 2014, p. 1). Legislators like Representative Tony Cárdenas from Southern California wants to pass a new bill titled, American Can Code, which encourages schools to start teaching programming as early as Kindergarten (NPR Staff, 2014).

There is an opposing view to the idea everyone should learn to program, which is held by educators and computer programmers alike (Atwood, 2007; Felker, 2013). The basics of computer programming are too hard for the mainstream population to learn, and critics further contend the idea that a fundamental level of knowledge is needed to work computers, is no longer valid (Atwood, 2007). Felker (2013) argued the lack of value hobbyists are going to be able to actually bring to the table since their skills are not as focused as a professionals. Since students not studying computer science for a major would only take a minimum number of classes, they would only have a rudimentary level of skill in programming that would not be useful (Felker, 2013).

Critics argue there is not a point to teaching everyone a skill only a few people truly need to learn and use the analogy not everyone is an auto mechanic, but they probably know how to drive a car (Atwood, 2007; Felker, 2013). Society is generally focused on dividing skills and labor so the people who need to learn specialized concepts can do so, but those who do not need that information would not have to go through the trouble of learning it (Atwood, 2007; Felker, 2013; Shein, 2014). There is also the concern that a push to teach the skill of computer programming in elementary classrooms will cause a displacement of something from the curriculum already being taught, and it is doubtful there is room in an already full elementary curriculum to add computer skills to the list (Felker, 2013).

The implication of a push to have everyone learn the basics of computer programming is the removal of corporate or institutional validation for computer programmers who consider themselves to be professionals (Fields & King, 2014). This shift in identity may have consequences on an industry that already sees a lack of qualified applicants (Felker, 2013). According to data from a 2005 Carnegie Mellon University study, more than 13 million workers described themselves as programmers, although only 3 million of these workers actually had a job as a professional software developer (NPR Staff, 2014). The concern is the market may be filled with applicants who feel their three-month boot camp, a very short-term learning experience with an overview of different programming techniques, gives them the qualifications for a professional position normally filled by a four-year graduate (Atwood, 2007; Felker, 2013; Fields & King 2014; Shein, 2014).

Attending college classes as a hobbyist. Community colleges not only offer classes towards a degree path but also welcome students who want to take a few classes to further their skills in a particular hobby (Vaughan, 2006). These students, known as hobbyist students, do not have a major declared (Vaughan, 2006). A hobbyist developer was defined by Jackson (2014) as "someone who spends 10 hours a month or more writing computer or mobile device programs, even though they are not paid primarily to be a programmer" (Jackson, 2014, p. 1).

Hobbyist students can pick and choose among courses listed in the course catalog and take a variety of courses that interest them, as long as they meet the prerequisites for the course (Vaughan, 2006). The difference between a hobbyist student and a regular student is found in the ultimate goal of attending college classes. Hobbyist students only intend to take a few courses without wanting a degree or pursuing the career (Vaughan, 2006). Hobbyist students are especially prevalent in the area of computer programming as the rise of the hobbyist programmer has become a mainstream idea (Atwood, 2007; Hars & Ou, 2001; Jackson, 2014).

While hobbyist students in computer programming typically have a higher rate of motivation than the average student population, their level of effort in their courses is not always as high since the consequences of failure are not as evident as they are for a student working to complete the course for their major (Fields & King, 2014). Further complications with hobbyist students come from the financial aid guidelines as some schools count each student who starts a degree path as a major in that field regardless of whether they have declared it or not (Chambliss & Takacs, 2014; McKinney & Novak, 2015). Since hobbyist students may be included in census information that is provided to the Department of Education for the calculation of funds, their presence may ultimately cause these numbers to be inaccurate, which could cause funding calculation issues or penalties (Chambliss & Takacs, 2014; McKinney & Novak, 2015).

Determining the correct number of students in a major is more complicated in community colleges where students can obtain a degree without ever declaring a major (Chambliss & Takacs, 2014; McKinney & Novak, 2015). Chambliss and Takacs (2014) pointed out, declaring a major is most often the postdated acknowledgement of a student’s commitment in a discipline since he or she has completed the courses without ever formally declaring his or her intent. Students attending community college classes just to get skilled for a particular hobby may not participate in student groups or other outside opportunities provided by the department and may appear to be a less-engaged student (McKinney & Novak, 2015).

If students are entering the degree path and taking entry level computer programming courses without the intention of ever completing a degree, this could be a problem for higher education institutions (Harvey et al., 2014; Monaghan & Attewell, 2015; Zeidenberg, 2015). Community colleges are often provided funding for their state based on the number of students who complete degrees at their institution (Harvey et al., 2014; Monaghan & Attewell, 2015; Zeidenberg, 2015). Student hobbyists may be counted as majors but never graduate, which can negatively impact the funding for an institution when the goals is to get students to complete a degree or transfer to a four-year university in order to secure more funding (Monaghan & Attewell, 2015; Zeidenberg, 2015). Hobbyist students may also account for the often-cited number of students who are leaving the computer science major who never had any intention of completing a degree (Hoskey & Maurino, 2011).

Summary

In this chapter, an overview of the relevant literature was presented to help provide a foundation on which this research study was guided and designed. The theory of multiple intelligences (MI) identified different areas of intelligences and helped outline professions that would benefit from an optimal set of skills (Gardner, 2013). Multiple intelligences theory has a strong foundation in educational research and can be used to help identify students who have the ability to be successful in a computer science classroom (Akkuzu & Akçay, 2011; Gentry & Lackey, 2011; Korkmaz, 2012). However, some detractors of MI theory do not feel it is capable of giving the full picture of a student's abilities, and a student who is determined to learn the skills necessary to be successful in computer science would still be able to do so (Ghazi et al, 2011). The theory of MI is key to determining whether or not there is any validity to aptitude in computer science.

Using the theory of MI, researchers can begin to work on the issue of retention in computer science courses (Akkuzu & Akçay, 2011; Gentry & Lackey, 2011; Korkmaz, 2012). As the workforce begins to have a large gap between incoming graduates and outgoing retirees, positions in computer programming are going to start to be in very high demand (College Board, 2013). However, it is key new graduates have a foundational set of skills in order to be considered employable (Akkuzu & Akçay, 2011; Gentry & Lackey, 2011; Hoskey & Maurino, 2011; Korkmaz, 2012).

Many faculty are finding students are leaving their computer science courses without a firm understanding of the concepts needed to be successful in the field (Hoskey & Maurino, 2011). In response, many new ideas on how to teach computer science have been developed in order to help make the topics more enjoyable for students (Ambrósio et al., 2011; Haungs, 2011; Radermacher et al., 2012). New teaching methods and techniques have been successful in improving student outcomes in introductory computer science classrooms (Ambrósio et al., 2011; Haungs, 2011; Radermacher et al., 2012). New technology, such as virtual reality games designed for learning, and wearable technology, such as the Microsoft Hololens, give students the ability to interact with environments in a whole new way, as well as investigate how spatial learning affects the area of computer science (de Castell et al., 2015).

This chapter contained an overview of many different types of teaching methodologies found in computer science classrooms. These new teaching methodologies were designed to impart the concepts needed for computer programming and the fundamentals of computer science (Ambrósio et al., 2011; Haungs, 2011; Radermacher et al., 2012). However, students reported they find the concepts difficult to learn and boring

(Kay, 2011; Nilsen & Larsen, 2011). In order to address the retention issue of computer science majors and the attrition of students in their first entry-level computer programming course, educators have been working to find new ways of presenting the foundational concepts of computer programming that students will find more engaging (Hoskey & Maurino, 2011; Kay, 2011; Porter et al., 2013a; Porter et al., 2013b).

Community colleges seem to have the largest problem with student retention, especially in more difficult programs such as computer science (Jenkins & Cho, 2012; Scott-Clayton, 2011). There have not been many studies focused on the issue of retention and the remediation of community college students, especially in the area of computer science (Jenkins & Cho, 2012). One way community colleges have tried to retain students is through attendance policies since there have been studies linking improved student academic performance to higher attendance (Adair & Swinton, 2012; Leufer & Cleary-Holdforth, 2010; Macfarlane, 2013; Marburger, 2006; Weimer, 2012).

Colleges are also following new mandates from the Department of Education to keep attendance on students in order to receive federal financial aid funds (Harvey et al., 2014). As these new mandates come into effect, colleges that receive federal financial aid are developing new tracking systems for attendance (Harvey et al., 2014). Also in response to these new mandates, institutions have started developing mandatory attendance policies that do not allow students to miss more than a pre-determined number of class periods before being removed from the course (Federal Student Aid, 2014; Harvey et al., 2014; Jesse, 2015). Attendance policies could affect the number of students who finish a course with either a passing or failing grade since continued attendance and participation are being required to stay enrolled (Archambault et al., 2013; Macfarlane, 2013). Since there has not been a lot of research on the subject of community colleges, the findings of this study could help identify methods for increasing student success in those schools.

Hobbyist students who are taking a few classes to gain skill in a particular area without the intent of obtaining a degree were also explored. Hobbyist students attend community colleges to take a few courses and never intend to graduate with a degree (Chambliss & Takacs, 2014; McKinney & Novak, 2015). Students who are not taking classes to obtain a degree can be a problem for schools whose funding model is based on the number of students who finish a degree path or transfer to a four-year university (Monaghan & Attewell, 2015; Zeidenberg, 2015). There are many implications student hobbyists create for schools that rely on degree-completing students for funding, for educators who want serious and motivated students in their classroom, and for the industry of computer programming (Atwood, 2007; Felker, 2013; Fields & King 2014; Hars & Ou, 2001; Jackson, 2014; NPR Staff, 2014).

In the area of computer science, students are encouraged to take entry level computer programming courses as a skill they may need in a heavily technology-based society but without the intent of becoming professional programmers (Atwood, 2007; Felker, 2013; Fields & King 2014; Hars & Ou, 2001; Jackson, 2014; NPR Staff, 2014). The debate on whether this skill is useful or relevant for non-professionals is ongoing in education and at the legislative level (Atwood, 2007; Felker, 2013; Fields & King 2014; Hars & Ou, 2001; Jackson, 2014; NPR Staff, 2014). Critics of the idea everyone should be taught to program feel it would devalue the work professionals perform, and they feel not everyone is capable of learning the needed skills (Atwood, 2007; Felker, 2013).

As well as the issue of non-degree seeking students, community colleges also deal with students who are not prepared for college classes (Braught et al., 2011; DeClue et al., 2011; Hakimzadeh et al., 2011; Porter et al., 2013a; Porter et al., 2013b). The issue of student preparedness for entry into community college was addressed in this chapter. Many students entering community college need remedial help in mathematics since they have been out of school for a while, and/or they did not properly learn the concepts in high school (Goldrick-Rab, 2010; Jenkins & Cho, 2012; Melguizo et al., 2011; Stigler et al., 2010). Students entering college deficient in mathematics often have to take several years of developmental math classes before they can meet the program requirements or prerequisites for the computer science major (Goldrick-Rab, 2010).

Since the longer a community college student is enrolled the less likely the student is to succeed, some of these students may drop out before they even get the opportunity to take computer science classes (Jenkins & Cho, 2012). If students are allowed to take entry level computer programming courses while they are working on their developmental math skills, it is most likely they will not have the foundational concepts needed for their programming classes, which will only lead to further discouragement (Hoskey & Maurino, 2011). Given a nation-wide drop rate of 30% in entry-level computer science courses, this issue of developmental math classes may be a cause for low retention (Hoskey & Maurino, 2011).

In Chapter Three, the methodology, research questions, and hypotheses used for this study are discussed. This chapter also outlines the population and sample used and gives a description of the instrumentation and data gathering techniques. An analysis of the study’s findings are presented in Chapter Four, and further discussion on these findings and recommendations are given in Chapter Five.

Chapter Three: Methodology

The theory of MI has played a key role in helping instructors develop classroom curriculum to meet student needs and improve instruction (Akkuzu & Akçay, 2011; Gardner, 2013; Gentry & Lackey, 2011; Korkmaz, 2012). As outlined in Chapter Two, the issue with failure in computer science classrooms appears to be deeper than just an issue of teaching methodology (Adorjan & Friss de Kereki, 2013; Boesch & Steppe, 2011; Bornat. 2011; Gardner, 2013; Ghazi et al., 2011; Korkmaz, 2012; Leal, 2013; Scott & Ghinea, 2013, 2014). The majority of the research on the topic of student failure rates in computer science courses has been conducted with students attending four-year universities, which have different requirements regarding admissions and prerequisites (Lail, 2009). This leaves a gap in the research conducted on students at two-year community colleges in the field of computer science (Clark et al., 2012; Jenkins & Cho, 2012). Although these schools appear to have the same issues of failure and retention in their classes, there is little research on how this problem affects community colleges given these differences.

This study investigated the potential for two predictors of student success in his or her first computer programming course: student self-perception of skill and success in a previous math course. By determining if these two are predictors of student success, there may be the potential for educators to use this information to identify students who may or may not be successful in their courses or as a computer science major. Also, this study sought to provide more information directly related to community colleges and students enrolled in two-year programs.

This chapter gives an overview of the problem addressed by this study. The research questions, hypotheses, and data collection methods are reviewed. The data analysis process used in this study is also included. The majority of the chapter is devoted to reviewing the methodology applied in this study, the population and sample studied, and an overview of the instrument used in gathering the data.

Problem and Purpose Overview

As previously noted in Chapter One, there is a well-documented problem in the retention of students taking entry level computer science courses (Hoskey & Maurino, 2011; Nikula, Gotel, & Kasurinen, 2011; Tabarrok, 2012). There have been many approaches and attempts at trying to help retain students in these courses, but none have posed a solution that can be applied to all students or at all institutions (DeClue et al., 2011; Haungs et al., 2012; Reese et al., 2013; Rizvi et al., 2011; Stewart-Gardiner, 2011). Since this problem of retention and student success has continued to persist, this study focused on the issue of student aptitude for computer programming and introductory computer science courses. This study explored the difference between a student's math aptitude and his or her ability to learn computer programming and searched for other indicators that may predict whether a student can be successful in learning the concepts of computer programming. The purpose of this study was to determine if there were indicators for programming aptitude for students enrolled in community colleges.

Research questions and hypothesis. The following research questions guided the study:

1. Is there a statistically significant difference between a student's previous success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?

   $H1_0$: There is no statistically significant difference between the student's success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

   $H1_a$: There is a statistically significant difference between the student's success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

2. Is there a statistically significant difference between a student's previous success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?

   $H2_0$: There is no statistically significant difference between the student's success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

   $H2_a$: There is a statistically significant difference between the student's success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors.

3. What indicators do students in their first computer programming course report as supporting their aptitude for the field of computer science?

Research Design

This study was based on a quantitative analysis of student data using the approach of the conceptual theorist, which is often used with research that is "impersonal, value-free, disinterested, imaginative, and problematic, involving multiple causation, purposeful ambiguity, and uncertainty" (Conrad & Serlin, 2005, p. 356). Creating a quantitative study is a novel approach, as educational research is typically qualitative in design (Doyle, 2011). This research study was non-experimental since the researcher could not manipulate the independent variable (Fraenkel et al., 2014).

Variables. According to Creswell (2013), variables are identified in a study in order to quantify or categorize important information and help break down the data in the study into smaller ideas to test. These variables are often classified into two groups, dependent and independent, which allow the researcher to review which pieces of information influence the study (Fraenkel et al., 2014). Variables are typically defined and used in quantitative research to help direct the research questions being used and to help determine the strategies of inquiry used (Creswell, 2013).

Independent variables, sometimes called experimental variables, are the pieces of data a researcher uses to observe the effect on dependent variables (Fraenkel et al., 2014). The independent variable in this study was success in the prior math course the student had passed. Since the research questions focused on the level of math a student had completed, this variable helped to show if there was in an indication a student would be successful in a computer programming course based on the level of math completed.

Dependent variables, sometimes called outcome variables, are pieces of data effected by the independent variables (Fraenkel et al., 2014). In this study, the dependent variable was the number of students who were able to pass the introductory computer programming course. The researcher examined factors that may indicate a student's success in an introductory computer programming course, the number of students who were able to pass the class was identified and connected with other data collected from the survey in order to look for success indicators.

Instrument reliability and validity. Validity and reliability are important considerations when designing an instrument since the researcher needs to be able to use the instrument to support inferences made in the study (Conrad & Serlin, 2005; Fink, 2012; Fraenkel et al., 2014). In order to ensure survey questions were clear, free of ambiguity, and asked questions that explained the outcome under investigation, field-testing was conducted. Field-testing is the recommended method to test for validity and to help ensure reliability of a survey instrument (Creswell, 2013; Fink, 2012). The purpose of a field-test is to allow a group of people, similar to the study population, the opportunity to examine and use the survey (Fink, 2012). Information provided from this group can help determine whether the survey is clearly understandable and if the questions being asked are not ambiguous (Fink, 2012). Conducting a field-test on a survey also helps identify ambiguous and weak questions, which can lead to weaknesses in the study (Fink, 2012).

The information gathered during a field-test is then used to modify the survey instrument so it can provide the researcher with the desired information (Fink, 2012). Field-testing a survey helps check for external validity, which shows how a survey's results can be used to generalize to the target population (Fink, 2012). The field-test process can also check a survey for content validity and help the researcher determine if the questions reflect the issue being researched (Fink, 2012). The survey to be used in this study was field-tested in order to ensure validity with a volunteer group of 10 students who were currently enrolled in computer science courses at the same two-year community college as the intended population, during the fall semester of 2014. Since these students were already majors in the computer science department, their input helped check the survey for content and external validity.

This pilot group completed the survey and provided feedback on terms, survey instructions, and the clarity of language. Survey responses obtained from the group were used by the researcher to determine if the collected information met the requirements for the study's research questions. After the field-test, students reviewed the survey and provided feedback, and the survey was adjusted according to the suggestions made by the group. Wording of questions was modified to remove ambiguity, and the order of the survey questions was changed to group questions that made sense together.

Reliability in a survey is a measurement of trustworthiness (Fink, 2012). Reliability of the survey used in this study was measured by allowing the pilot group from the field-study to take the survey a second time, once prior feedback had been implemented. By allowing the group to retake the survey, the results could be compared against the previous attempt to determine consistency in the information collected by the survey.

Population and Sample

The population for this study was students with a declared program of study in computer science at one two-year community college in southwest Missouri. At the southwest Missouri community college where this research was conducted, approximately 300 students were majors in the field of computer science (College Institutional Research, 2014). The community college chosen for this study is one of the largest in the state, with approximately 14,000 full-time students enrolled each year and is the largest community college in the southwest region of the state (Missouri Department of Higher Education, 2014). The sample group of students used in this study were chosen from this population since this is the largest available population in the region.

The sample for this study was the students enrolled in the first-year, introductory computer programming course required for computer science majors during the fall 2014 and spring 2015 semesters who chose to participate in the study. This sample consisted of 107 students enrolled in the course at the time the study took place. This study sample did not include students who held a status as dual-enrolled high school students, since they are not considered to be fully admitted to the college. No demographical or other information about these students was included in the sample data.

Instrumentation

To collect the data needed for this study, a researcher-developed survey was used to gather data from the enrolled students in the introductory computer science course. This survey was conducted anonymously using an online format through the FluidSurveys website. No identifying information was collected during the survey. The survey data was exported from the FluidSurveys website and loaded into an Excel spreadsheet which was kept password-protected and accessible only to the researcher. According to Fink (2012), when using surveys for quantitative research, it is important for survey participants to remain anonymous to promote truthfulness in responses. This survey was developed to be clear and unambiguous with close-ended questions to allow for specific student responses (Fink, 2012).

Student survey. A cross-sectional survey of 14 questions was designed to give the researcher quantitative data used in combination with the student grade data to answer the research questions (Appendix A). It is common for survey data to be collected when responses from people are needed to help give quantitative research data greater meaning (Fink, 2012). This survey was delivered to the students as an electronic survey that was emailed to students as a link. The survey was given unsupervised to all students who choose to participate in the study (Fink, 2012; Fraenkel et al., 2014) during two weeks of the course during the spring 2015 semester. A survey was chosen as the best method to collect quantitative data, such as the student's level of preparedness for the course based on previous mathematical education, the number of credit hours enrolled, and information regarding a student's previous experience in computer science or expectations of his or her first computer science course (Fink, 2012). According to Fraenkel, et al., (2014), a survey is the best tool to determine the characteristics of the sample. Students who choose to participate in the study had two weeks to complete the online survey.

Survey development. The survey was designed using a standardized format so the researcher could draw correct conclusions from the gathered information and to ensure the questions were presented in a clear manner and were easy to answer (Fraenkel et al., 2014). The survey consisted of closed-ended questions designed to give the researcher further information about the student, such as how many credit hours the student was enrolled in, how much time he or she spent each week preparing for the course, and whether the course met the student's expectations. One open-ended question was added to the survey to collect information about why students chose the field of computer science as their major. This question was used by the researcher to help put other student responses into perspective.

Data Collection

Once the approval of Lindenwood's IRB and the IRB of the participating community college was granted (see Appendix B & Appendix C), the data collection process began with an electronic communication to the students enrolled in the introductory computer programming course through their college email account from a private email account of the researcher. The student's email account information was provided by the college's Institutional Research department. This communication included a letter of introduction explaining the study and informed consent (see Appendix D). The letter outlined the assurances student responses would be kept confidential, no risks or benefits of participation were involved, and student participation was voluntary.

Survey data were collected within a two week period during the spring 2014 semester, before students completed their midterm exam. The anonymous surveys were given to students via a link to the FluidSurveys website. The survey tool prevented students from taking this survey twice. While taking the survey, students were asked to complete all survey questions, but were allowed to stop the survey and withdraw from the process at any time.

Final grade data for each student enrolled in the introductory computer programming course along with the grade of the highest math class he or she had completed was requested. This information was provided by the Institutional Research department at the college where the study was performed. The Institutional research department provided final course grades for the student's math courses and entry-level computer programming course.

At the end of the fall 2014 semester, a request was made to the participating community college for the necessary grade data. The quantitative data for each student were the grade for the highest math class the student had successfully passed and the final grade in his or her introductory programming course at the end of the fall 2014 semester. The research department of the community college assigned each student a random number and gave the student's math course score and programming course score. The researcher did not receive any identifying data for the students.

All of the data were provided by the selected school's institutional research department and given to the researcher in an anonymized format. The data given contained an anonymous student number, each student's final grade for his or her computer programming course, and the final grade for the highest math class the student had completed with a passing score. This information was stored in a password-protected folder by the researcher.

Data Analysis

The first step taken in the analysis of secondary data in this study was to divide the data into two groups (Group A and Group B) to address the first two research questions. Group A contained students identified as having passed a college-level math course, and Group B contained students identified as having passed intermediate-level math course. All sections of the introductory programming course use the college's standard percentage grading scale where a grade of A would be given for a 100% - 90%

and a B for 89% - 80%, and so on, with a passing grade for the course identified as a grade of C or better. For the purpose of this research study, success was determined to be a passing grade of C or better.

Since there are two options for the use of *t*-tests based on the variance of the data, an *F* test was performed on the data to determine if the variances were equal or unequal (Bluman, 2015). It was found that both Groups A and B contained data of unequal variance; therefore, the *t*-test for differences in means assuming unequal variances was used (Bluman, 2015). The data in both Group A and Group B met the criteria for the use of the *t*-test since the sample sizes were greater than 30, the population standard deviation was not known, and the samples were taken from approximately normally distributed populations (Bluman, 2015).

A *t*-test for differences in means assuming unequal variances was run on both Group A and Group B to determine if the average grades obtained in the math and computer programming courses were significantly different when compared to a grade of C, which was the minimum passing score for each course. According to Bluman (2015), a *t*-test is used to test the difference between means when the samples are independent and when the samples are taken from approximately normally distributed populations. These tests were performed using the IBM SPSS Statistics (SPSS) predictive analytics program to perform the statistical calculations (IBM SPSS Statistics, 2015). These tests were analyzed at an $\alpha = 0.05$ level of significance (Bluman, 2015).

A data set was generated from each question answered by the survey participants and descriptive statistics were used to analyze the data. Measures of central tendency were found for each question result, and dispersion measures, such as range and standard deviation, were used to determine how close the response values were to the central tendencies. For survey questions 10 through 14, Steven's Scale of Measurement (Ary, Jacobs, & Sorenson, 2010) was used to give context to the answers given since these questions were in a Likert Scale. An interval measurement scale was developed so the mean and standard deviation could be found for each question (Ary et al., 2010).

Using Steven's Scale of Measurement allows researchers to put Likert Scale items into a context (Ary et al., 2010), which can then be calculated. To find the mean for each Likert Scale question, each response was assigned an interval value of -2 for Strongly Disagree, -1 for Disagree, 0 for Neutral, 1 for Agree, and 2 for Strongly Agree. By evaluating the mean for each question, the researcher determined if the response was truly positive, neutral or negative, given how many standard deviations it fell from the mean (Ary et al., 2010).

Data from the surveys were compiled and analyzed for statistical significance and were used to give further meaning to the conclusions drawn from the results of the statistical analyses of the final course grade data collected. The survey data helped the researcher identify areas which could lead to corrections in the failure of computer science students in other colleges. According to Creswell (2013), surveys can be used in quantitative research in order to help generalize the conclusions drawn from analysis of a sample into a clearer picture of the larger population.

Summary

This quantitative study was designed to determine if there were predictors to success for students in their first computer programming course at a two-year college. Data were collected with permission from Lindenwood's IRB and the participating school from the sample of students identified. These data included the final grade in the programming course and the grades for the highest math class the student had completed by the end of the fall semester in 2014. Final grades from the courses were statistically analyzed to search for their ability to be used as predictors of computer programming success. A survey was also administered to all students who participated in the study in order to gather data about the number of credit hours they were enrolled in, whether they felt prepared for their computer science coursework through their previous mathematics courses, and other information that gave the researcher indicators for computer programming aptitude.

The data collected were analyzed using a *t*-test for differences in means assuming unequal variances to determine if there was a statistical difference between the grade the student received in his or her mathematics course and his or her grade in an introductory computer programming course (Bluman, 2015). The survey data were used to gather the student's perceptions of his or her aptitude and to look for other predictors of student success in computer programming courses. These indicators were used to identify students who would or would not be successful in computer programming courses.

In Chapter Four, the results of this study are presented. These results are described along with each of the statistical steps used to analyze the data. Finally, the outcome of the statistical test is reviewed along with the information gathered from the student surveys.

Chapter Four: Analysis of Data

The purpose of this study was to determine if there was a statistical difference between mathematical ability and computer programming ability in two-year community college students at one community college in southwest Missouri. The data in this study were gathered in order to help guide personnel in computer science programs in determining prerequisites and math placement scores. Since the problem of retention of students in computer science programs has become a growing concern (Hoda & Andreae, 2014; Hu, 2011; Scott & Ghinea, 2013), higher education institutions need to determine ways of identifying students who will be successful in the computer science major so they may better advise them at the beginning of their enrollment. In this chapter, the data collected are summarized and results of the descriptive and inferential analyses are presented.

This study was guided by three research questions that were evaluated using the quantitative methods outlined in Chapter Three. Student grade data for an entry-level computer programming course for the fall 2014 semester from 86 students were evaluated to answer the first two research questions and determine if there was a statistical difference between the grade students received in introductory computer programming and mathematics courses. The third research question was evaluated using survey results of 32 adult students who agreed to participate during the spring semester 2015. The survey data were also examined to find other predictors that point to student success in a computer programming major. The data from the survey were analyzed using the Microsoft Excel spreadsheet program and tests were performed to look for statistically significant differences in order to address each of the research questions.

Respondent Demographics

The target population for this study was students enrolled in their first computer programming course at a two-year community college in southwest Missouri. The list of student survey participants was furnished by the college where this research was performed. Out of a total of 107 potential respondents who fell within the population requirements, a total of 32 students responded to the survey. The secondary data, which included course grades, contained information from students who were enrolled in their first introductory-level computer programming course during the fall 2014 semester. The secondary data report contained information for 86 students. The data were provided by the Institutional Research department of the college where the research was conducted, as outlined in Chapter Three.

Data Analysis

Before survey data could be analyzed, the information was downloaded from the electronic survey instrument website, FluidSurveys, which was used to deliver the survey to the students. No identifying information was collected during the survey. The survey data were exported from the FluidSurveys website and loaded into an Excel spreadsheet so analysis of the results could be performed. The spreadsheet with the survey results was kept in a password-protected file accessible only to the researcher.

The secondary data report was delivered to the researcher from the Institutional Research department of the college where the study was performed. The report data were then added to the password-protected spreadsheet where the survey data were being kept. All of these steps were performed in compliance with the Lindenwood IRB and the IRB of the college where the research was conducted.

The total number of students in the secondary data was 86. The data were then divided into two groups. Group A was composed of students who had passed at least a college-level math class, and Group B was composed of students who had passed at least an intermediate-level math class. The college's course catalog defined a passing grade in this course as being a C or higher (Institution Course Catalog, 2015).

The data that were provided to the researcher came in the form of final letter grades. For the purposes of statistical analysis, a scale of A = 5, B = 4, C = 3, D = 2, and F = 1 was given to each grade. Since a grading scale is a scaled variable, there should not be any skewing of the output values by assigning these values (Bluman, 2015).

Research question one. *Is there a statistically significant difference between a student's previous success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?* For research question one, the results of Group A were reviewed since they had completed a college-level math course. The computer programming course grade breakdown for these students is outlined in Table 1.

Table 1

Final Grades for Group A Computer Programming Students

Grade	Distribution
A	10
B	25
C	4
D	1
F	1

Note. N = 41.

The grading guidelines for this particular course stated a grade of C or better was required to be counted as a passing grade (Institution Course Catalog, 2015). The observation was made that 39 students from Group A completed the computer programming course with a passing grade, and two students did not. The pass rate for students in Group A with college-level math skills was found to be 95.12%.

In order to determine the level of variability in the data, an *F* test was used. First, the confidence interval was set. According to Bluman (2015), the three most common confidence intervals are 90, 95, and 99%. Based on this information, the determination was made to use a confidence level of 95% or $\alpha = .05$. For Group A, the result was $p =$ 0.06 which was greater than the stated confidence level of $\alpha = .05$. A *p* value higher than a confidence level indicates a significant amount of variability in the two groups and the variance is unequal (Bluman, 2015).

A *t*-test for differences in means assuming unequal variances was conducted for Group A due to the unequal variance reported by the *F* test. The *t*-test was used to determine if the average course grade in their first computer science class was significantly different from the course grade in their math class.

Table 2

T-test for Group A

Statistic	Math Course	Programming Course
Mean	3.95	4.02
Variance	1.10	0.67
Observations	41	41
t Stat	-0.35	
p (*T* < = *t*) two-tail	0.73	
t Critical two-tail	1.99	

The *t*-test was performed in SPSS as a two-tailed test, which is shown in Table 2. The analysis produced a *p* value of 0.73. Since $p \geq 0.05$, this test failed to reject H_0 at the $\alpha = .05$ level. Based on the results of this analysis, there was enough evidence to support the claim there is no significant difference between a student's score in his or her introductory computer science course and the score in his or her math course for Group A.

Research question two. *Is there a statistically significant difference between a student's previous success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?* Using Group B from the grade report data as outlined for the second research question, it was noted 39 students completed the computer programming course with a passing grade, and six students did not. The pass rate for students in Group B with intermediate-level math skills was found to be 86.66%

The grade distribution for students who were identified as Group B are shown in Table 3. As noted, for this particular course, a grade of C or better was required to be counted as a passing grade (Institution Course Catalog, 2015). Also included in Group B were 18 students who successfully completed a mathematics placement test that indicated they had the skills needed to meet the prerequisites for the introductory programming course. These students were admitted to the course based on this demonstrated skill and had not attempted a further mathematics courses before taking the programming course.

Table 3

Final Grades for Group B Computer Programming Students

Grade	Distribution
A	10
B	21
C	8
D	4
F	2

Note. N = 45.

In order to determine the level of variability in the data, an *F* test was used for Group B. The confidence interval was set at a level of 95% or *α* = .05, the same as used for the analysis of Group A. The result was *p* = 0.74, which was greater than the stated confidence level of α = .05. A *p* value higher than a confidence level indicates a significant amount of variability in the two groups and the variance is unequal (Bluman, 2015).

A *t*-test for differences in means assuming unequal variances was conducted for Group B as for Group A with the information shown in Table 4.

Table 4

T-test for Group B

Statistic	Math Course	Programming Course
Mean	3.18	3.73
Variance	1.42	1.11
Observations	45	45
t Stat	-2.34	
p (*T* < = *t*) two-tail	0.02	
t Critical two-tail	1.99	

The *t*-test was performed in SPSS as a two-tailed test, which produced a *p* value of 0.02. Since $p \leq 0.05$, this test rejected the H_0 at the *α* = .05 level. Based on the results of this analysis, there is enough evidence to support the claim there is a significant difference between a student's course grade in his or her introductory computer science course and the course grade in his or her intermediate-level math course for Group B. Thus the null hypothesis was rejected.

Research question three. *What indicators do students in their first computer programming course report as supporting their aptitude for the field of computer science?* Data were collected using an electronic survey that was delivered to the institutional email accounts of students enrolled in the entry-level computer programming course. The survey remained open for data collection for three weeks during the spring 2015 semester. The survey consisted of 14 questions. The questions in the survey were designed to help identify other predictors which may point to a student's success in his or her computer programming course outside of the math factor that was evaluated in the first two research questions. Of the 107 students the survey was sent to, 32 students chose to respond, for a response rate of 30%. The results of the survey were analyzed using descriptive statistics to look for indicators for student success in computer programming courses. The results of each survey question are discussed in the following section.

Question one. *How many credit hours are you enrolled in this semester?* Analysis of the survey data for this question indicated this sample of students took an average of seven and a half credit hours in the fall 2014 semester. The majority of students, or 47%, reported taking six to 12 credit hours a semester while 11 students reported taking more than 12 credit hours. At a two-year, open-enrollment school in southwest Missouri where the research was conducted, 12 credit hours a semester is considered a full-time student (College Student Handbook, 2015).

Question two. *What is the highest level of mathematics that you have completed with a grade of C or better?* One of the key areas this research study focused on was the level of math the student had completed before taking his or her introductory computer programming course. Students were asked in the second survey question which math course they had completed with a grade of C or better and the distribution of answers is shown in Table 5.

Table 5

Completed Math Course from Survey Responses

Basic Algebra	Intermediate Algebra	Technical Math	College Algebra	Higher than College Algebra
6	12	1	3	10

Note. N = 32.

Question three. *What degree are you seeking at* [*Name of College*]*?* The majority of the students surveyed reported their majors as being in the field of computer programming, and 21 respondents stated they were pursuing an Associates of Applied Science in Computer Information Science (AAS CIS) from the college. One student noted he or she was obtaining a Certificate in Computer Information Science (CT CIS), and eight other respondents reported majoring in another field, such as the Associate of Arts (AA), or were non-degree seeking students. The distribution of responses is shown in Figure 1. At the time this study was conducted, the Computer Information Science department had 236 declared majors and degree seeking at either the Certificate or Associates degree level (Institution Major Report, 2015).

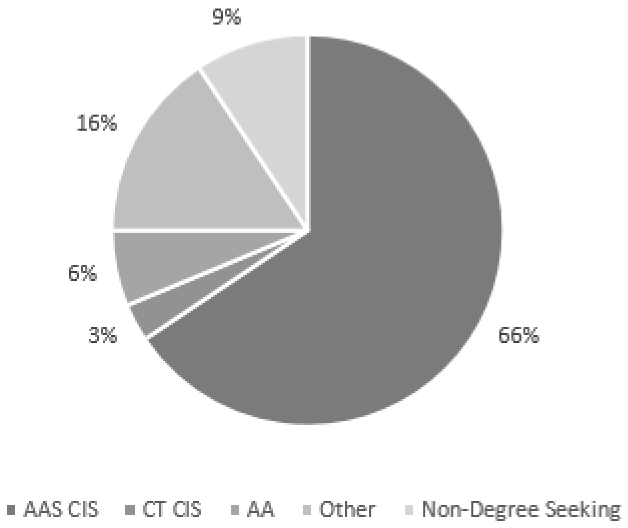


Figure 1. Distribution of degrees student respondents were seeking at their current institution.

Question four. *Did you have any previous programming experience before starting this course?* For question four, 50% of the survey respondents had prior programming experience. Sixteen students reported previous experience, and 16 reported they did not have prior knowledge or skill.

Question five. *Is this your first time taking this particular course?* This question was asked to determine how many students had previously taken this course either failed or had to retake it. Four students, or 12%, fell into this category and stated they had previously attempted this course.

Question six. *How much time did you spend outside of class on homework for this course?* Figure 2 displays the responses for question six and shows 56% of students reported spending one to three hours per week on homework, and 18% spent three to five hours a week. The average, six hours per week on homework in an introductory programming course, is higher than the average of three hours per class, or 14.8 hours per week on all courses for which students were enrolled (Young, 2002).

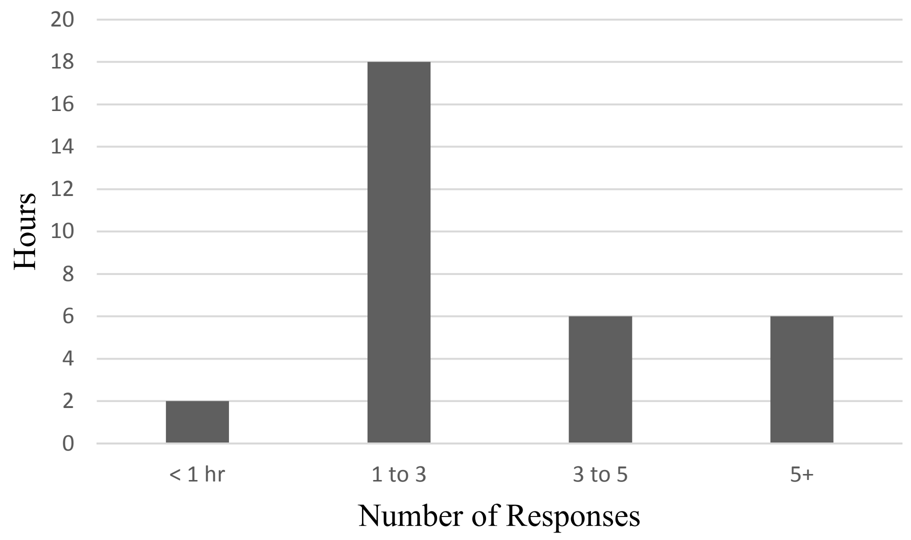


Figure 2. Student responses on the amount of time spent outside of class on programming homework.

Question seven. *Next semester do you plan to: (choose all that apply).* Of the respondents, 90% reported they would be taking another computer science course at the same school, and only 12% responded they would be changing majors after the semester. One student reported he or she would be transferring to another school.

Question eight. *What grade do you expect to receive in this course?* During the survey, students were asked what their expected final grade would be in the course, and none of the students reported they expected to receive a grade lower than a C (see Table 6).

Table 6

Student Expected Grade from Survey Responses

Expected Grade	A	B	C
Number of Responses	15	14	3

Note. N = 32.

These results were consistent with the results of question 7, which asked about the student's future plans. None of the students expected to fail the course and thus, could not move on to the next course in their major's sequence.

Question nine. *When do you normally take your classes?* Students were asked when they normally take their courses to determine if there was a large population of traditional students who take classes during daytime hours, or non-traditional students who normally take classes at night. Of the respondents, 19 said they took daytime classes, and 13 said they took evening classes.

Questions ten through fourteen. Four questions were given to students on the survey in a Likert Scale format, which were designed to be combined to measure a particular trait in the students surveyed. These questions were given to help determine if the students reported using skills in their computer programming courses which are found in the theory of MI (Gardner, 1993). The questions were designed to examine the two intelligences that correlate to computer science skills; Logical-Mathematical and Visual-Spatial. Questions 10-14 were analyzed using Steven's Scale of Measurement (Ary et al., 2010), and since they were Likert Scale items, an interval measurement scale was used to find the mean and standard deviation for each question. To find the mean for each Likert Scale question, each response was assigned an interval value of -2 for Strongly Disagree, -1 for Disagree, 0 for Neutral, 1 for Agree, and 2 for Strongly Agree. The results of the survey were calculated against these interval scores so descriptive statistics could be used for each of these questions. The calculated mean and standard deviation for each question is shown in Table 7.

Table 7

Descriptive Statistics for Likert Scale Survey Questions

Question Number	Mean	Std. Deviation
10	7.00	12.65
11	.60	8.29
12	-1.40	10.24
13	6.00	10.42

Note. $N = 32$.

Using Steven's Scale of Measurement allowed the researcher to put the student responses into a better context as to whether they were truly a positive or negative response. Each response was then evaluated to determine on which side of the mean it fell and how many standard deviations it was removed from the mean.

Question ten. *My level of computer programming skill has improved because of this course.* Students were asked if they felt their level of computer programming skill had improved because of the course, and 89% responded positively. The Agree category had a score of 14, and the Strongly Agree category had a score of 26, which were both above the mean and indicated a highly positive weighted response. Three students responded neutrally to this question, and 12% responded negatively.

Question eleven. *My previous mathematics courses helped prepare me for my first computer programming course.* This question related to the students' perceptions on how their previous mathematic courses had helped them in their computer programming class. There was no clear majority one way or the other for the responses to this question, since 44% of students reported it was beneficial, and 35% reported it was not, while 22% of students had a neutral viewpoint.

Question twelve. *When I solve problems in my programming course, I like to use pictures, mind-maps, flow-charts, and other visual tools.* This question asked if students used these tools to look for visual-spatial aspects of the theory of MI and to determine if students were using these internal visualization skills or intelligences when solving computer programming problems. Students had a strong negative stance on how they use visualizations or visual tools when solving these problems. The student respondents were unfavorable towards the visual tools and concepts for question 12 with 44% of the students responded negatively. The negative category for question 12 had a score of -16, which was one standard deviation below the mean of -1.40.

Question thirteen. *I have no trouble visualizing concepts in my mind such as arrays, memory storage, and program data flow.* This question was designed to ask students if they used visualizing concepts in their mind to look for visual-spatial aspects of the theory of MI, similar to question 12. The difference between the two lies in where the visualization takes place. Question 13 looks for students using internal visualization and imagination to understand programming concepts. Question thirteen received a positive response, with 68% of students in agreement.

Question fourteen. *What inspired you to become a major in Computer Science?* This question was presented in an open-response format so students could provide a short statement. Of the 32 student respondents, two opted not to answer this question. These responses fell into five categories the researcher used to organize student responses. The results of this categorization can be found in Table 8.

Table 8

Summary of Responses by Category for Survey Question Fourteen

Category	Number of Responses
Interest in the field as a hobby	12
Looking for a career that pays well	8
Enjoys math and problem solving	6
Family or advisor recommended	3
Religion	1

Note. $N = 30$.

Summary

This study had two purposes, first to determine if there was any statistically significant difference between a student's success in his or her mathematics courses and his or her success in an introductory computer programming course, and second, to look for predictors of student success in computer programming courses. The study, which was guided by three research questions, was conducted at a two-year college in southwest Missouri with 34 survey respondents who volunteered to participate in the spring semester 2015 and secondary data from 86 students who had completed the introductory computer programming course.

In this chapter, the data were presented and the analysis steps defined. A *t*-test for differences in means assuming unequal variances was performed on the data, and the results were compared with the critical values for the first two research questions. For research question one, H_0 was rejected and the determination was made that a statistically significant difference did not exist between the students' success in their introductory computer programming course and their success in a college-level math course. For research question two, H_a was supported. There was a statistically significant difference in the students' final course grade in an intermediate-level math course and their success in an introductory computer programming course.

Survey data were also reported in this chapter and evaluated. The average number of students surveyed were taking 7.5 credit hours per semester, and the majority were working towards an Associate of Applied Science degree in Computer Information Science. It was reported many students entered the major since they enjoyed computers as a hobby, while others stated they were attracted to computer science as a career that pays well. Half of the students in the introductory class stated they had some previous programming experience, and 12% were taking the course for a second time. These students visualized computer programming concepts in their mind but did not like to use external tools.

In Chapter Five, the findings from the study are presented. Each research question is discussed, and conclusions are revealed. Implications for practice are described, and recommendations for future research are suggested.

Chapter Five: Summary and Conclusions

Since 1953, when computer science was first taught in a university setting (Jones, 2001), the problem of student success in computer programming has existed. The research reviewed in Chapter Two contained a variety of factors, such as teaching methodologies, retention, and the student's cognitive abilities as indicators for student success in computer programming. A student's skills in mathematics has been often studied as a means of determining whether a student has the logic and cognitive skill to learn the concepts of programming (Ambrósio et al., 2011; Erdogan et al., 2008; Hu, 2011; Scott & Ghinea, 2013). The purpose of this study was to look for predictors of student success by looking at two factors: student success in previous mathematics courses and student perceptions of their own computer programming skill. By identifying predictors of student success, educators at two-year schools can use this information to help identify students who may or may not be prepared for entry-level computer science courses in a higher education institution.

This study was conducted at a two-year, open-enrollment school in southwest Missouri. Three research questions guided the study, which was quantitative in design. Survey data were collected from 32 students who agreed to participate in a survey during the spring semester of 2015. Secondary data from 86 students who were enrolled at the institution during the fall semester of 2014 were also collected. In this chapter are the findings from the statistical analysis of the data and conclusions. Implications for practice drawn from the results and recommendations for future research on this topic are also included.

Findings from Research

Research question one. The first research question guiding this study was: *Is there a statistically significant difference between a student's previous success in a college-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?* The H_0 accompanying this research question stated there was no statistically significant difference between the student's success in a college-level math course and his or her success in an introductory computer programming course. Since math scores have previously been determined to be a good indicator of success in computer programming courses (Coyle et al., 2014; Leal, 2013), the purpose of this question was to determine if a student who had taken a college-level math course showed any indication he or she would be successful in the introductory computer programming course. As outlined in Chapter Four, the students who passed at least a college-level math course were designated Group A for the purposes of testing this research question. Group A was tested using the *t*-test for differences in means assuming unequal variances. Since the *p* value for Group A was 0.72584 and the confidence level was $\alpha = .05$, the test failed to reject H_0 for Research Question One. There was sufficient evidence to state there was a no statistically significant difference between the success rates for students in college-level math courses and their entry level computer science course.

Research Question Two. The second research question guiding this study was: *Is there a statistically significant difference between a student's previous success in an intermediate-level math course and his or her success in the first course of a sequence of programming courses designed for computer science majors?* The H_0 accompanying this research question stated there was no statistically significant difference between the student's success in an intermediate-level math course and his or her success in an introductory computer programming course. This question was designed to help determine if students could be successful in their introductory computer programming course with a foundational level of math but without the skills higher-level math courses would provide. For many community college students, an intermediate level of mathematics is sometimes challenging and can be the only math course required for their computer science degree program (Jenkins & Cho, 2012; Scott-Clayton, 2011).

As described in Chapter Four, these students were identified as Group B for the purposes of analysis and comparison with Group A. The survey data showed 56% of the students who participated fell into Group B. The same *t*-test for differences in means assuming unequal variances was run for Group B using the same level of confidence. Since the *p* value for Group B was 0.021451736 and the confidence level was $\alpha = .05$, the test rejected $H_{0,}$ and the H_a was supported for Research Question Two. There was significant evidence to support the claim there was a statistically significant difference between a student's grade in an intermediate-level math course and his or her success in an entry-level computer programming course.

Research Question Three. The third research question that guided this study was: *What indicators do students in their first computer programming course report as supporting their aptitude for the field of computer science?* To gather information on these indicators, a cross-sectional survey of 14 questions was developed. Student responses to this survey were gathered during the spring 2015 semester from 32 volunteer participants. This survey was given to help identify if other factors could be identified which could predict student success in an introductory computer programming course.

Several predictors in the survey were identified such as the number of credit hours a student was enrolled in and if he or she had previous programming experience before entering the course. The majority of students were taking the course to ultimately obtain their A.A.S. degree in Computer Information Science, which is the highest degree in this major offered at this institution. Of the 32 survey respondents, it was noted 12% had previously attempted this course, and 90% intended to take the next computer science course in the sequence at this school. The responses in the survey were consistent with the secondary data provided by the college, which showed the students' highest math course completed.

As noted in Chapter Four, the last question of the survey was an open-response question that allowed students to provide the reason they were interested in becoming a computer science major. These responses were aggregated into five broad categories for analysis. These five categories were: interest in the field as a hobby, looking for a career that pays well, enjoys math and problem solving, family or advisor recommended, and religion. The majority of the responses fell into the interest in the field as a hobby category.

Conclusions

This study was designed to examine the differences between a student's mathematical knowledge and his or her ability to be successful in his or her first computer programming course. Many researchers have explored the idea that the subject of computer programming is hard to learn and have focused on the student's problem solving skills as the reason for the struggle (Porter et al., 2013a; Porter et al., 2013b; Reese et al., 2013; Stewart-Gardiner, 2011; Watson & Li, 2014). The relationship between math and problem solving skills is not a new concept (Boesch & Steppe, 2011; Leal, 2013; Scott & Ghinea, 2013), and this study added to the existing literature on student aptitude for computer programming and mathematics by looking at students at open-enrollment institutions. There are a different set of challenges for student success in computer programming that open-enrollment schools face (McClenney et al., 2012), since the incoming students may not have any background in mathematics. By investigating predictors for student success and a student's performance in math courses, there was the potential for educators to use the findings of this study to help identify students who may or may not be a good fit to major in computer science at two-year, open-enrollment schools.

For the first research question, the Group A students had completed at least a college algebra level math course before taking their entry-level computer programming course. A statistically significant difference was not found between the success for students in college-level math courses and their entry level computer science course. This was a surprising outcome since research has shown math scores to be a good indicator of success in computer programming courses (Coyle et al., 2014; Leal, 2013).

For the second research question, the Group B students had completed at least an intermediate-level math course, but nothing higher. According to the data analyzed in this study, there was a significant difference in the outcomes for these students in their entry-level computer programming course. This was also a surprising outcome since when these two groups were compared, the difference was very noticeable with 13% of the students from Group B who stayed in the course failing, compared with 4% failing from Group A. Students from the A Group were shown to be more successful in their entry-level computer programming course than students in Group B who did not have the higher-level of math knowledge.

For the third research question, a key finding came from the survey in which students reported they strongly disagreed with the statement: *When I solve problems in my programming course, I like to use pictures, mind-maps, flow-charts, and other visual tools.* According to the research of Gardner (2013) and Ghazi et al (2011), students who have strong Visual-Spatial skills would be better prepared for the academic challenges computer programming would pose. The fact students surveyed disagreed with this statement was a contradiction to the outcome of another survey question: *I have no trouble visualizing concepts in my mind such as arrays, memory storage, and program data flow.* Survey participants agreed with this second statement, which could mean they either misunderstood one of these questions or they had no trouble doing this mental visualization, but disliked the software tools and strategies taught such as flow-charts.

The theoretical framework used in this study was Howard Gardner's theory of multiple intelligences (MI), which has been used for many years to define student learning styles and a student's ability to process information in different formats (Gardener, 1993, 2013). The survey instrument designed for Research Question Three focused on Gardner's ideas of intelligences and how not all students have the same levels or sets of intelligences; therefore, each has strengths and weaknesses in different skill areas (Doyle, 2011; Gardener, 1993, 2013). As previously noted in Chapter Two, the two main intelligences from Gardner's theory helpful for computer programming students were the areas of Logical-Mathematical and Visual-Spatial (Gardener, 1993). The survey instrument had questions designed to examine a student's perspective of the use of these intelligences.

Previous research into using MI and the Logical-Mathematical intelligence to determine the aptitude of computer science students found a direct relation between the Logical-Mathematical skill and computer programming skill (Adorjan & Friss de Kereki, 2013; Korkmaz, 2012). Students who tested as having high aptitudes on tests designed to look for skills in the Logical-Mathematical intelligence were able to learn the foundation skills of computer programming and algorithm design faster and better than students who did not test into the high level of aptitude (Korkmaz, 2012). The conclusions of this study align with those of previous research, since it was found students with higher level mathematics skills performed better in their entry level computer programming course.

Despite the connections made in research between math skill and computer programming skill, the survey question that was used to examine a student's use of Logical-Mathematical skills in his or her computer programming class did not have positive results. In this question, *My previous mathematics courses helped prepare me for my first computer programming course,* 22% of the students gave a neutral answer and 35% gave a negative answer. According to the student responses, students did not necessarily make the connection that math coursework helped them be successful in their programming classes. This outcome could be related to students not using direct math skills in their programming course and not relating the problem solving skills they learned in their math course to the problems they were required to solve in the programming course. If students were given more math-based or related problems to solve in their programming course, it may have changed their answers to this question.

The survey questions where Visual-Spatial intelligences were addressed had mixed results. In the question, *I have no trouble visualizing concepts in my mind such as arrays, memory storage, and program data flow,* 69% of the students agreed they had and used this capability. As noted in Chapter Four, this positive response was a direct contradiction to the previous question that also looked at how Visual-Spatial skills students were using through visual tools or software. Of the students surveyed, 44% did not like to use visual tools such as pictures, mind-maps, or flow-charts, which are typically used in entry-level computer programming courses to help guide students in their program flow and logical skill building (Hoskey & Maurino, 2011; Scott & Ghinea, 2013). The results of survey responses indicated Visual-Spatial intelligence is important for computer programming students; however, students did not enjoy using tools which helped them exercise this intelligence. Students appeared to want to take a more internal, imaginative approach to visualizing computer programming logic and problem solving.

Implications for Practice

The findings and conclusions of this study can be used to help guide computer science instructors and curriculum developers with choices regarding entry-level programming course prerequisites and content delivery. This quantitative study, grounded in Gardner's theory of multiple intelligences, helped define some of the skills students need to be successful in introductory computer programming courses. The implications found from the results of this study are outlined in this section.

The underlying problem in computer science courses is retention (Hoda & Andreae, 2014; Hu, 2011; Scott & Ghinea, 2013). There is also the issue that incoming students lack the skills needed to be successful in their programming courses (Hoskey & Maurino, 2011). With many factors at play, such as how many credit hours a student is taking a semester, how much time programming students are spending on homework, and any outside experience or training these students could already have, it can be difficult to point to one area needing improvement in order to retain students (Cuseo, 2013; Goldrick-Rad, 2010; Stigler et al., 2010).

The implications of the results from the first research question, which indicated there was no significant difference between a student's grade in a college-level math course and his or her grade in the first computer programming course, could be used in the future for introductory computer programming courses. It appears there is a barrier to how much math knowledge is needed for a student to be successful in an entry-level computer programming course. Even though the results of the hypothesis test did not show significant results, students who had completed at least a college-level math class earned more A's than the group that had only had an intermediate-math course. The findings also showed students failed less in Group A than in Group B.

The implications of the results from the second research question, which indicated there was a significant difference between a student's grade in the intermediate-level math course and his or her grade in the first computer programming course, could be used when setting course prerequisites for entry-level computer programming courses. Further study would be needed on the success of students in entry-level computer programming courses that had not had previous math classes. At the college where this study was conducted, a minimum mathematics level was already in place for students entering the computer science program.

The analysis of the first two Research Questions have led to further thought about barriers to student success in computer programming courses. While further analysis of the secondary data does not answer the research questions, it does show some interesting aspects. For example, when the data are viewed through smaller sub-groups based on the number of students who achieved a course grade of A, B or C, as these would be considered passing grades, further details emerge. This disaggregated data allowed for a more in-depth look at how many students are achieving higher scores in the class versus students who are performing below the passing level.

Sub-groups for Group A were created for students passing the course and those who did not. Table 9 shows the central tendency for these two sub-groups. The data were disaggregated in this format to give a more in-depth look at how many students were achieving higher scores in the class verses students who were performing below the passing level. The mean score for successful students in Group A was 4.15, which is just over a grade of B in the introductory programming course. The unsuccessful students in Group A had an average grade of F for their attempts.

Table 9

Central Tendency for Group A Sub-groupings

	Mean	Median	Std. Deviation
Successful Students	4.15	4.0	.59
Unsuccessful Students	1.5	1.5	.71

Note. N = 41.

As with Group A, sub-groups for Group B were created for students passing the course and those who did not. Table 10 shows the central tendency for these two sub-groups. The mean score for successful students in Group B was 4.05, which was a grade of B in the introductory programming course. This was the same performance as Group A. However, it is noted that the standard deviation for the successful students in Group A was lower, at .59 than the same students in Group B at .69. These results indicated that while both sub-groups had attained almost the same mean grade, 95.12% of students in Group A had achieved passing scores versus 86.66% in Group B.

Table 10

Central Tendency Data for Group B Sub-groupings

	Mean	Median	Std. Deviation
Successful Students	4.05	4.0	.69
Unsuccessful Students	2.0	2.0	0

Note. N = 45.

By looking at the disaggregated data in combination with the results from the research questions, it became clear there were some other factors involved in the success of students in computer programming courses. These other factors, called confounding variables, are other aspects not considered in this study which may have affected the outcome of the research results (Fraenkel et al., 2014). Students who perform well in at least a college-level math class are more likely to perform well in their introductory computer programming course; however, the skills learned in the college-level math course may not be the cause. Students in the higher level math group may have been enrolled in college longer than students in the intermediate math group, which would have given them experience in study strategies and college-level class expectations.

Students who have been enrolled for several semesters may have more resources which allow them to be better students, thus earning them higher grades.

The implication of this additional information is students who have been in college long enough to work up to a college-level math class, or who entered into the college at this mathematics level, perhaps are more likely to succeed in passing an entry-level computer programming course. Whether the cause was from increased logical skills derived from their math courses or other factors, would be a subject for future investigation. While students in the intermediate-level math group are capable of passing an entry-level computer programming course, they do not appear to have the same level of success as students in the higher-level math group.

When considering the recommendation to increase the prerequisite on entry-level computer programming courses, it is important to remember that with increased prerequisites comes higher barriers for entry, especially at two-year, open-enrollment institutions where students are more likely to be entering the college and needing remedial math assistance (Jenkins & Cho, 2012). Students coming to a community college with only the most basic math skills would have three or four math courses to take before they could obtain the college algebra course prerequisite for the introductory computer programming course. This series of coursework could mean an additional two years of math instruction, along with other general education courses before a student could start the course sequence for their major.

Findings from the results of the third research question, which examined student aptitude in computer science, found many indicators in the survey responses. The number of credit hours a student is enrolled and whether he or she had previous programming knowledge before starting the course would have direct implications on the student's success (Barbatis, 2010; Barnett, 2011; Yates, 2010). The survey results also pointed to some changes, which could be made to computer science teaching methodologies to improve student outcomes.

Given the relationship between the number of credit hours a student is enrolled and student success, it is important to note 37% of the student respondents stated they were taking more than 12 credit hours a semester. A student who has a full-time load or classes may have trouble keeping up with coursework with outside demands such as work or family (Barbatis, 2010; Barnett, 2011; Yates, 2010). In question six of the survey, 56% of students also reported they spend only one to three hours per week on homework, which may be an indicator of why some students perform poorly in their computer programming course. The expectation is seven to 10 hours a week will be spent on homework, reading, and study for the course (Young, 2002). Helping students to understand the expectations for study outside of class and the amount of hours needed for homework may help some students to perform better in computer programming courses.

The responses to question 12 were unclear on whether students felt prior math coursework helped prepare them for their entry-level computer programming course. This result may mean the course itself does not contain many problems which use logarithmic thinking or mathematical formulas. However, this result may be consistent with the findings of the first two research questions given 43% of student respondents had taken at least a college-level math course. Students who had taken higher-level math may not utilize it in their entry-level computer programming course, thus the uncertain result.

Since students indicated they do use internal methods of visualization for problems but do not like to use the tools or methodologies taught, the recommendation for instructors in two-year computer science programs is to start looking for new ways to foster the use of student's ability to visualize these concepts. It is best for people who are strong in the Logical-Mathematical intelligence to learn and form concepts before they have to deal with details, which may be the underlying cause for student frustration (Gardner, 2013). Too many details are being presented to students in these courses, which is creating a negative learning environment for those with Visual-Spatial intelligences (Gardner, 2013).

One of the key findings for this study came from the final question of the student survey, which asked students why they are majoring in computer science. The findings revealed 40% of the respondents stated they entered into the major since they were interested in the field as a hobby. As discussed in Chapter Two, many students come to community colleges to take a few classes that interest them in a particular field, and the broad field of computer technology as a whole inspires many people to learn these skills without interest in joining the profession.

The fact 40% of the students enrolled in a computer programming course at a two-year year, open-enrollment school in southwest Missouri since they enjoy the field as a hobby ties into the results from question four: *Did you have any previous programming experience before starting this course?* Half of the student respondents stated they had previous programming experience, which may come from their own self-directed learning. Understanding why students begin computer programming courses can be helpful when recruiting students or seeking to retain them. At the college where this study was conducted, there may be a significant population of students only attending computer programming classes as a hobby without the intention of getting a degree, since 9% of the respondents stated they were non-degree seeking. When college administrators are reviewing the number of students who complete a degree versus the number of students who have started a degree path, this 9% may not be students the department is failing to retain. These hobbyist students never intended to finish a degree.

Recommendations for Future Research

Further research in the area of two-year computer science students is warranted to see how these differences truly affect their ability to be successful. Additional research could be conducted on students who have completed their entry-level computer programming course but continue to struggle with concepts in subsequent courses. Further studies could examine students who passed the first few computer science courses in the major, but ultimately were not successful in obtaining a degree. It is also a recommendation that a more comprehensive survey instrument be developed and administered to more students in order to achieve a larger representation of students' opinions on what helps them in courses and what does not help.

Additional research could also be performed on schools that implement the findings of this study and increase the mathematics prerequisite on their introductory computer programming course. The opportunity is present to investigate the need for increased math prerequisites and the number of students who are likely to overcome that barrier. As noted in Scott-Clayton's (2011) study, many students do not like to take college-level math courses and often put them off, even after completing a remedial sequence. Additional research could be performed on students' reasons and perceptions of their math classes while taking their computer science programming course sequence.

Another avenue for additional research could be found in the students who are not retained as computer science majors. As Hoskey and Maurino (2011) stated, 30% of new majors do not stay in the computer science program. There is an opportunity for research in determining why these majors are not staying in the program and finding out which program they are moving to when they drop their computer science major. Finding where students are going and their reasons for leaving could lead to answers of other questions regarding computer science major retention. Further qualitative research would be needed to explore student responses to why they did not finish their first computer programming course.

There is an opportunity for future research on students who test into math classes versus students who have completed math classes at the institution before taking their entry-level computer programming course. As noted in the implications section, it is unknown whether students may be performing better in their computer programming courses due to prior college experience and familiarity with the institution, course policies, and college-level expectations. Also, several students in the sample data provided for this study had taken a placement test upon entering the college to demonstrate their mathematics skill, which may have skewed the outcome of the statistical result. Further exploration into these topics could provide more information useful to computer science departments and provide evidence for student success in computer programming courses.

As noted in the study survey data, the majority of students were entering their first computer programming course or declaring a computer science major because they enjoyed computers as a hobby. These students could present another area for future research, since hobbyist students may be tied to the issue of student retention in the major of computer science. According to the American Association of Community Colleges (2015) many students attend community colleges to take classes which may pertain to a hobby. Further research may show the drop in majors is partially due to the amount of students who are taking these classes for fun, without the intention of pursuing a career in the field.

Finally, once this study was completed, the results from the research questions helped show how this study could be expanded or changed. Since there appears to be some confounding variables involved in the identification of student barriers to computer programming course success, an expanded study based on these results could be conducted using additional information. The number of credit hours a student has completed and his or her grade point average would help remove some of the ambiguity from the results of this study and help identify whether students are truly more successful due to the advanced math coursework or not. Additional identifiers outlined in this study from the survey could also be used to help identify factors which point to student success in their entry-level computer programming course.

Summary

The purpose of this study was to determine if there were indicators for student success in entry-level computer programming courses at a two-year, open-enrollment College in southwest Missouri. The review of literature presented in Chapter Two outlined some of the ways higher education institutions are working to improve student success in entry-level computer programming courses and to improve retention for computer science majors. It was noted the majority of research on these topics was performed at four-year universities that had different requirements for entry into the university and major (Clark et al., 2012). Therefore, this study focused on two-year students at an open-enrollment college to see if improvements could be made for student success in an environment with very few entry requirements.

This quantitative study, guided by three research questions, used secondary grade data and a survey instrument to gather information about students currently enrolled in their first computer programming course. The secondary data were statistically analyzed using the *t*-test for differences in means assuming unequal variances, and the results of these tests led the researcher to conclude students who passed a college-level math course were more likely to be successful in their entry-level computer programming course than students who had passed a lower-level math course. With 30% of students nation-wide in entry-level computer programming courses not being successful, institutions should look at the prerequisites for these courses and consider adding or increasing the math requirements for students entering their program (Hoskey & Maurino, 2011).

Gardner's (2013) theory of MI was used to give this study a foundation in the kinds of learning styles or mental abilities best suited for a computer science major. Gardner's MI theory also allows teachers to identify a student's particular strengths and aptitudes so lessons can be presented in a way most natural for the student to learn. Two intelligences were identified from Gardner's theory as being the most important for computer programming students: Logical-Mathematical and Visual-Spatial.

Questions from the survey instrument were designed to evaluate students' perceptions on these intelligences by determining if they use visual or logical tools when solving problems in their computer programming course. According to the results of this study, students did not feel their previous math experience was helping them in the course despite the secondary grade data analysis showing there was a significant difference. Students also did not enjoy using visual tools to solve problems in their computer programming course, but they did acknowledge they used internal visualization skills to help them solve problems or understand program logic.

The recommendation was made that computer science instructors work to find new ways to help students use their internal visualization or look for new tools and problem solving methodologies students enjoy using. By giving students the ability to solve problems on their own terms and not be confined to a particular tool, more students may be able to find success.

By conducting this research, many new areas for future research have been identified. The results further the information on computer science student enrollment in two-year, open-enrollment schools such as community colleges. As previously noted, there was a gap in the research for students at these particular schools, and further studies should be conducted in this area to provide more data. Professors and administrators at community colleges can use the findings of this study to help shape entry-level computer programming course curriculum and set the prerequisites of these courses appropriately.

Many students are unprepared for college level work in open-enrollment schools (Barrow, Brock, & Rouse 2013; Long, 2014; Morris, 2012; Scherer & Anson, 2014). Computer science departments appear to see students continue to fail or drop out before completing the first class (Porter et al., 2013a; Porter et al., 2013b; Reese et al., 2013; Stewart-Gardiner, 2011; Watson & Li, 2014). However, many open-enrollment schools still have created academic barriers for entry on some programs (Long, 2014). If creating barriers for entry into a major ultimately puts students in a better position to succeed, then an argument could be made that high prerequisites and barriers for entry into certain classes is in the student's best interest (Barbatis, 2010; Barnett, 2011; Yates, 2010).

In conclusion, while there may not be a clear difference between the amount of mathematical training and a student's success in an entry-level computer programming course, several factors were identified in this study which colleges can use to help guide their computer science program curriculum. Some of the barriers holding students back from success in their first computer programming course were identified, such as the number of credit hours the student is enrolled and the amount of time spent outside of class on homework, and this information can be used to advise students during their, course of study.

This study also applied the theory of MI and identified the intelligences which may point to student success in computer programming courses. The results of this study added to the continuing body of research on the theory of MI and how it applies to computer science students. These findings could be used by computer science faculty when building curriculum if they wish to target specific intelligences which have been identified as being important in computer science aptitude.

Students may be capable of successfully completing an entry-level computer programming course without the skills higher-level math classes provide, but further study is needed on this topic to have a clear conclusion. Colleges should continue to look at prerequisites and determine if they are currently set at the level that will provide the best outcome for student success. It is imperative to conduct research on other variables that may impact student achievement in computer programming courses.

Appendix A

Electronic Survey Instrument

1. How many credit hours are you enrolled in this semester?
    a. 3 - 6 credit hours
    b. 6 – 12 credit hours
    c. +12 credit hours

2. What is the highest level of mathematics that you have completed with a grade of C or better?
    a. MTH050 – Pre-Algebra
    b. MTH110 – Intermediate Algebra
    c. MTH105/TEC108 – Technical Math
    d. MTH130 – College Algebra
    e. Higher than MTH130

3. What degree are you seeking at [redacted]?
    a. Associate of Applied Science in CIS
    b. Certificate in CIS
    c. Associate of Arts Transfer Degree
    d. Other Degree
    e. Non-degree Seeking

4. Did you have any previous programming experience before starting this course?
    a. Yes
    b. No

5. Is this your first time taking this particular course?
    a. Yes
    b. No

6. How much time did you spend outside of class on homework for this course?
    a. Less than 1 hour a week
    b. 1 – 3 hours a week
    c. 3 – 5 hours a week
    d. +5 hours a week

7. Next semester do you plan to: (choose all that apply)
    a. Take another programming course at [redacted]
    b. Change majors and stay at [redacted]
    c. Transfer to another school for a CIS or related program
    d. Transfer to another school and change majors
    e. Not continuing education

8. What grade do you expect to receive in this course?
    a. A
    b. B
    c. C
    d. D
    e. F

9. When do you normally take your classes?
    a. Daytime (before 4 pm)
    b. Evening (after 4 pm)

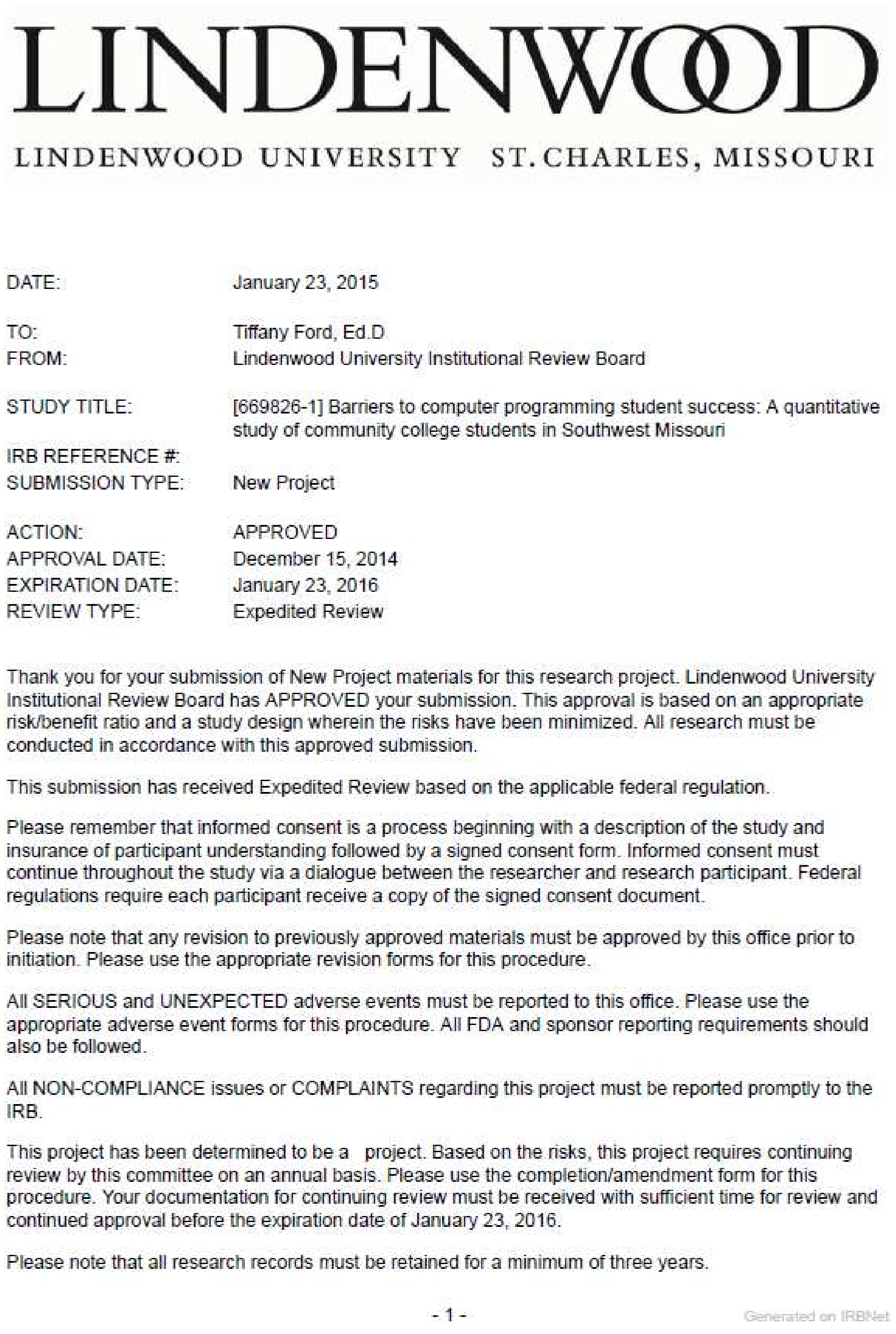
Please rate each statement using the following scale:

Question	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
10. My level of computer programming skill has improved because of this course.					
11. My previous mathematics courses helped prepare me for my first computer programming course.					
12. When I solve problems in my programming course, I like to use pictures, mind-maps, flow-charts and other visual tools.					
13. I have no trouble visualizing concepts in my mind such as arrays, memory storage, and program data flow.					

14. What inspired you to become a major in Computer Science?

Appendix B

IRB Approval from Lindenwood



LINDENWOOD

LINDENWOOD UNIVERSITY ST. CHARLES, MISSOURI

DATE:	January 23, 2015
TO:	Tiffany Ford, Ed.D
FROM:	Lindenwood University Institutional Review Board
STUDY TITLE:	[669826-1] Barriers to computer programming student success: A quantitative study of community college students in Southwest Missouri
IRB REFERENCE #:	
SUBMISSION TYPE:	New Project
ACTION:	APPROVED
APPROVAL DATE:	December 15, 2014
EXPIRATION DATE:	January 23, 2016
REVIEW TYPE:	Expedited Review

Thank you for your submission of New Project materials for this research project. Lindenwood University Institutional Review Board has APPROVED your submission. This approval is based on an appropriate risk/benefit ratio and a study design wherein the risks have been minimized. All research must be conducted in accordance with this approved submission.

This submission has received Expedited Review based on the applicable federal regulation.

Please remember that informed consent is a process beginning with a description of the study and insurance of participant understanding followed by a signed consent form. Informed consent must continue throughout the study via a dialogue between the researcher and research participant. Federal regulations require each participant receive a copy of the signed consent document.

Please note that any revision to previously approved materials must be approved by this office prior to initiation. Please use the appropriate revision forms for this procedure.

All SERIOUS and UNEXPECTED adverse events must be reported to this office. Please use the appropriate adverse event forms for this procedure. All FDA and sponsor reporting requirements should also be followed.

All NON-COMPLIANCE issues or COMPLAINTS regarding this project must be reported promptly to the IRB.

This project has been determined to be a project. Based on the risks, this project requires continuing review by this committee on an annual basis. Please use the completion/amendment form for this procedure. Your documentation for continuing review must be received with sufficient time for review and continued approval before the expiration date of January 23, 2016.

Please note that all research records must be retained for a minimum of three years.

- 1 -

Generated on IRBNet

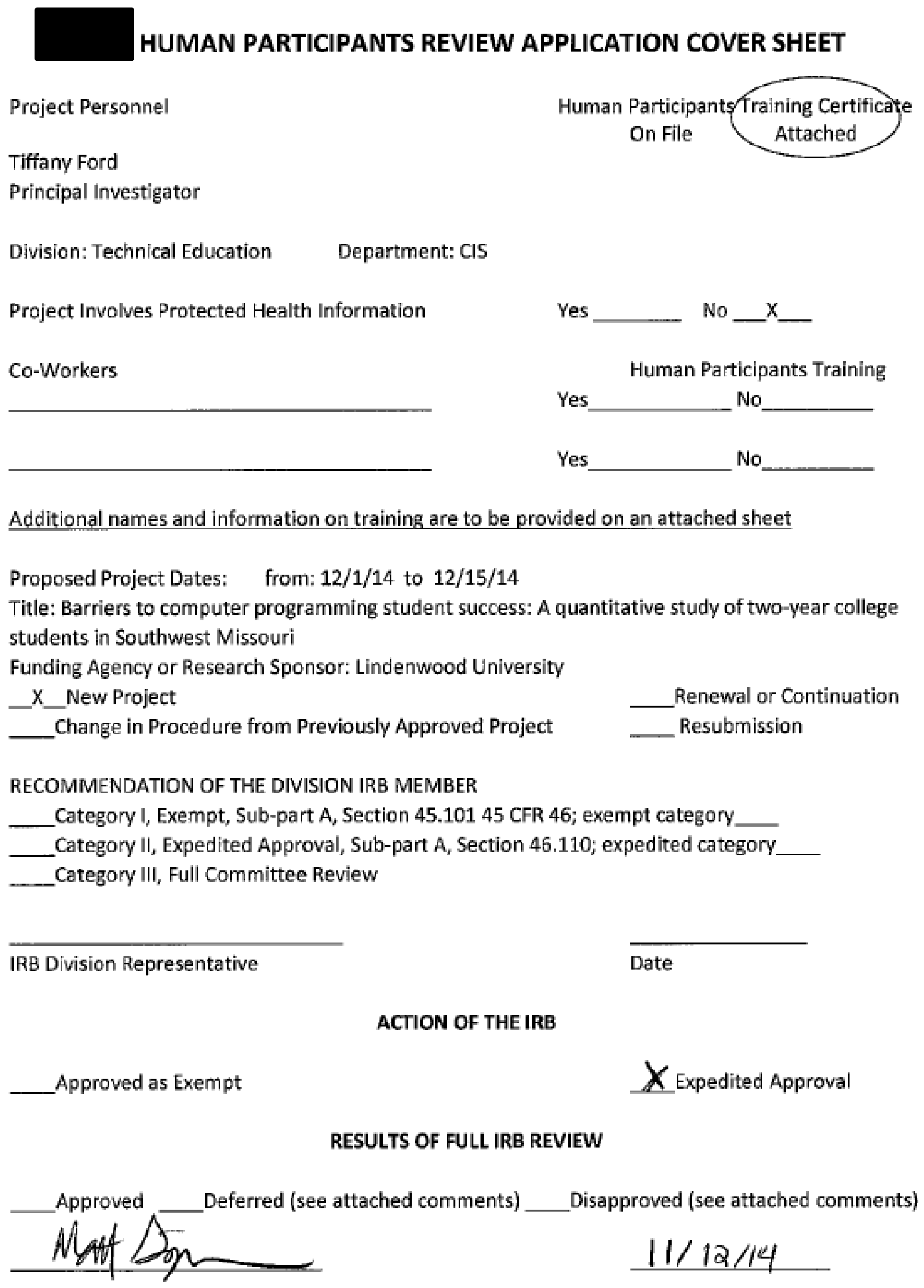
If you have any questions, please contact Robyne Elder at (314) 566-4884 or relder@lindenwood.edu. Please include your study title and reference number in all correspondence with this office.

If you have any questions, please send them to IRB@lindenwood.edu. Please include your project title and reference number in all correspondence with this committee.

This letter has been electronically signed in accordance with all applicable regulations, and a copy is retained within Lindenwood University Institutional Review Board's records.

Appendix C

IRB Approval from Research Site



[redacted] **HUMAN PARTICIPANTS REVIEW APPLICATION COVER SHEET**

Project Personnel — Human Participants [✓ Training Certificate Attached] On File

Tiffany Ford
Principal Investigator

Division: Technical Education — Department: CIS

Project Involves Protected Health Information — Yes ________ No ___X___

Co-Workers — Human Participants Training
________________________ Yes____________ No__________

________________________ Yes____________ No__________

Additional names and information on training are to be provided on an attached sheet

Proposed Project Dates: from: 12/1/14 to 12/15/14
Title: Barriers to computer programming student success: A quantitative study of two-year college students in Southwest Missouri
Funding Agency or Research Sponsor: Lindenwood University
__X__New Project ____Renewal or Continuation
____Change in Procedure from Previously Approved Project ____ Resubmission

RECOMMENDATION OF THE DIVISION IRB MEMBER
____Category I, Exempt, Sub-part A, Section 45.101 45 CFR 46; exempt category____
____Category II, Expedited Approval, Sub-part A, Section 46.110; expedited category____
____Category III, Full Committee Review

________________________ ________________
IRB Division Representative Date

ACTION OF THE IRB

____Approved as Exempt X Expedited Approval

RESULTS OF FULL IRB REVIEW

____Approved ____Deferred (see attached comments) ____Disapproved (see attached comments)

[illegible] 11/12/14

Appendix D

Letter of Introduction and Informed Consent

LINDENWOOD

INFORMED CONSENT FOR PARTICIPATION IN RESEARCH ACTIVITIES

Barriers to Computer Programming Student Success: A Quantitative Study of Two-year College Students in Southwest Missouri

Principal Investigator Tiffany Ford

Telephone: [redacted] E-mail: [redacted]

1. You are invited to participate in a research study conducted by Tiffany Ford under the guidance of Dr. Sherry DeVore. The purpose of this research is to examine the difference between math ability and computer programming ability for students enrolled in two-year schools in southwest Missouri.

2. a) Your participation will involve:
   - ➢ Completion of a survey involving your computer programming experience, computer programming courses, and math courses you have taken. This survey will only be completed once.

   b) The amount of time involved in your participation will be approximately 20 minutes. The survey consists of nine questions.

Approximately 60 students will be involved in this research. This survey will be given to all students enrolled in CIS120 – Problem Solving and Programming Concepts at the [redacted]

3. There are no anticipated risks associated with this research.

4. There are no direct benefits for you participating in this study. However, your participation will contribute to the knowledge about how math skills relate to computer programming ability.

5. Your participation is voluntary and you may choose not to participate in this research study or to withdraw your consent at any time. You may choose not to answer any questions that you do not want to answer. You will NOT be penalized in any way should you choose not to participate or to withdraw.

6. We will do everything we can to protect your privacy. As part of this effort, your identity will not be revealed in any publication or presentation that may result from this study and the information collected will remain in the possession of the investigator in a safe location.

7. If you have any questions or concerns regarding this study, or if any problems arise, you may call the Investigator, Tiffany Ford ([redacted]) or the Supervising Faculty, Dr. Sherry DeVore (417-881-0009). You may also ask questions of or state concerns regarding your participation to the Lindenwood Institutional Review Board (IRB) through contacting Dr. Jann Weitzel, Vice President for Academic Affairs at 636-949-4846.

By proceeding to the next page of this survey you are giving your informed consent to participate in the research described above. After you have completed the survey, a "Thank you" email with a copy of your informed consent will be emailed to you.

Thank you in advance for your time and assistance.

Tiffany Ford

References

Achieving the Dream. (2015). Helping more community colleges succeed. *Lumina Foundation*. Retrieved from http://achieveingthedream.org

Adair, K., & Swinton, O. H. (2012). Lab attendance and academic performance. *ISRN Education*. DOI:10.5402/2012/364176

Adorjan A., & Friss de Kereki, I. (2013). Multiple intelligence approach and competencies applied to Computer Science 1. *IEE Frontiers in Education Conference,* 1170-1172.

Adadi A., Lister, R., & Teague, D. (2014). Falling behind early and staying behind when learning to program. In *25th Anniversary Psychology of Programming Annual Conference*.

Ahanbor, Z., & Sadighi, F. (2014). The relationship between multiple intelligences, learning styles and gender. *Modern Journal of Language Teaching Methods*, *4*(1), 176-184.

Akkuzu & Akçay, N. H. (2011). The design of a learning environment based on the theory of multiple intelligence and the study its effectiveness on the achievements, attitudes and retention of students. *Procedia Computer Science*, *3*, 1003-1008.

Alice Educational Software (2015). *Carnegie Mellon University*. Retrieved from http://www.alice.org/index.php

Ambrósio, A. P., Costa, F. M., Almeida, L., Franco, A., & Macedo, J. (2011). Identifying cognitive abilities to improve CS1 outcome. In *Frontiers in Education Conference (FIE), 2011* (pp. F3G-1). IEEE.

American Association of Community Colleges (2015). Students at community colleges. *Community College Trends and Statistics*. Retrieved from http://www.aacc.nche.edu/aboutcc/trends/Pages/default.aspx

American Psychological Association (2010). *Publication manual of the American Psychological Association* (6th ed.). Lancaster, PA: Lancaster.

Amresh, A., Carberry, A. R., & Femiani, J. (2013, October). Evaluating the effectiveness of flipped classrooms for teaching CS1. In *Frontiers in Education Conference, 2013 IEEE* (pp. 733-735). IEEE

Archambault, L., Kennedy, K., & Bender, S. (2013). Cyber-truancy: Addressing issues of attendance in the digital age. *Journal of Research On Technology In Education (International Society For Technology In Education)*, *46*(1), 1-28.

Ary, D., Jacobs, L. C., & Sorensen, C. (2010). *Introduction to research in education* (8th ed.). Belmont, CA: Thomson Wadsworth.

Atwood, J. (2007, November 25). The two types of programmers. *Coding Horror*. Retrieved from http://blog.codinghorror.com/the-two-types-of-programmers/

Avancena, A. T. S., Nishihara, A., Vergara, J. P., & City, Q. (2012). Development of online cognitive and algorithm tests as assessment tools in introductory computer science courses. In I*nternational Conference on Cognition and Exploratory Learning in Digital Age*, CELDA (45-248).

Barbatis, P. (2010). Underprepared, ethnically diverse community college students: Factors contributing to persistence. *Journal of Developmental Education*, *33*(3), 14-24.

Barnett, E. A. (2011). Validation experiences and persistence among community college students. *The Review of Higher Education*, *34*(2), 193-230.

Barrow, L., Brock, T., & Rouse, C. E. (2013). Postsecondary education in the United States: Introducing the issue. *The Future of Children*, *23*(1), 3-16.

Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010). Using scalable game design to teach computer science from middle school to graduate school. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (24-228). ACM.

Berkeley College (2015). Attendance policy. *Berkeley College 2015 Student Handbook.* Retrieved from: http://berkeleycollege.edu/files_bc/Student_Handbook.pdf

Blaho, M., Foltin, M., Fodrek, P., & Murgaš, J. (2012). Students' perspective on improving programming courses. *International Journal of Education and Information Technologies*, *6*(1), 17-24.

Bluman, A. (2015). *Elementary statistics: A step by step approach* (9th ed.). New York, NY: McGraw-Hill.

Boesch, C. & Steppe, K. (2011). Case study on using a programming practice tool for evaluating university applicants. *Conference Proceedings of 2nd International Conference on Computer Science Education: Innovation and Technology*, Singapore, December 5-6. doi:10.5176/2251-2195_CSEIT37

Bornat, R. (2011). *Some problems of teaching (learning) first-year programming (plus a glimmer of hope).* Jisc TechDis Publication *11*.

Braught, G., Wahls, T., & Eby, L. M. (2011). The case for pair programming in the computer science classroom. *ACM Transactions on Computing Education* (TOCE), *11*(1), 2. Association of Computing Machinery

Carbonaro, M., Szafron, D., Cutumisu, M., & Schaeffer, J. (2010). Computer-game construction: A gender-neutral attractor to Computing Science. *Computers & Education*, *55*(3), 1098-1111.

Chambliss, D. F. & Takacs, C. G. (2014). *How college works*. Cambridge, MA: Harvard University Press.

Clark, T. J., Waller, L. R., Lumadue, R., & Hendricks, L. (2012). A comparison of retention rates among America's 2-year institutions of higher education. *Focus on Colleges, Universities, and Schools*, *6*,(1).

The College Board. (2013). *AP program participation and performance data 2013 research and development.* Retrieved from http://research.collegeboard.org/programs/ap/data/participation/2013

Conrad, C. F. F., & Serlin, R. C. (2005). *The Sage handbook for research in education: Engaging ideas and enriching inquiry.* Thousand Oaks, CA: SAGE Publications.

Copyright Law of the United States of America (n.d.). *Definitions*. Retrieved from http://www.copyright.gov/title17/92chap1.html#101

Coyle, T. R., Purcell, J. M., Snyder, A. C., & Richmond, M. C. (2014). Ability tilt on the SAT and ACT predicts specific abilities and college majors. *Intelligence, 46*, 18-24.

Creswell, J. W. (2014) *Educational research: Planning, conducting, and evaluating quantitative and qualitative research, enhanced*. New York, NY: Pearson.

Creswell, J. W. (2013). *Research design: Qualitative, quantitative, and mixed methods approaches*. New York, NY: SAGE Publications.

Cuseo, J. (2013). Student success: Definition, outcomes, principles and practices. In *The Big Picture*. National Resource Center for the First-year Experience & Students in Transition, University of South Carolina.

Davies, S., Polack-Wahl, J. A., & Anewalt, K. (2011). A snapshot of current practices in teaching the introductory programming sequence. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (pp. 625-630). Association of Computing Machinery.

de Castell, S., Jenson, J., & Larios, H. (2015). Gaming experience and spatial learning in a virtual morris water maze. *Journal For Virtual Worlds Research*, *8*(1).

DeClue, T., Kimball, J., Lu, B., & Cain, J. (2011). Five focused strategies for increasing retention in Computer Science 1. *Journal of Computing Sciences in Colleges*, *26*(5), 252-258.

Dehnadi, S., & Bornat, R. (2006). *The camel has two humps* (working title). Middlesex University, UK.

Doyle, H. J. (2011). *Community college and technical college students' perceptions of their learning success based upon understanding multiple intelligences: A mixed method research study* (Doctoral dissertation). Retrieved from ProQuest Dissertations and Theses. (OCLC Number. 767615770)

Erdogan, Y., Aydin, E., & Kabaca, T. (2008). Exploring the psychological predictors of programming achievement. *Journal of Instructional Psychology, 35*(3), 264-270.

Esteves, M., Fonseca, B., Morgado, L., & Martins, P. (2011). Improving teaching and learning of computer programming through the use of the second life virtual world. *British Journal of Educational Technology, 42*(4), 624-637.

Federal Student Aid (2014). *Federal student aid handbook, 2014–2015*. Washington, DC: U.S. Department of Education.

Felder, R. M., & Silverman, L. K. (1988). Learning and teaching styles in engineering education. *Journal of Engineering Education*, *78*(7), 674-681.

Felder, R. M., & Brent, R. (2005). Understanding student differences. *Journal of Engineering Education*, *94*(1), 57-72.

Felker, C. (2013). *Maybe not everybody should learn to code. Slate*. Retrieved from http://www.slate.com/articles/technology/future_tense/2013/08/everybody_does_not_need_to_learn_to_code.html

Fields, D. A., & King, W. L. (2014). "So, I think I'm a programmer now:" Developing connected learning for adults in a university craft technologies course. In Learning and becoming in practice: *The International Conference of the Learning Sciences,* (*1*, 927-936).

Fink, A. G. (2012). *How to conduct surveys: A step-by-step guide*. Thousand Oaks, CA: Sage Publications.

Fraenkel, J. R., Wallen, N. E., & Hyun, H. H. (2014). *How to design and evaluate research in education* (9th ed.). New York, NY: McGraw-Hill.

Furnham, A. (2014). Increasing your intelligence: Entity and incremental beliefs about the multiple "intelligences." *Learning and Individual Differences*, *32*, 163-167.

Gardner, H. (1985). *Frames of mind: The theory of multiple intelligences*. New York, NY: Basic Books.

Gardner, H. (1993). *Multiple intelligences: The theory in practice*. New York, NY: Basic Books.

Gardner, H. (2013). *The unschooled mind: How children think and how schools should teach*. New York, NY: Basic Books.

Gentry, R., & Lackey, T. K. (2011). *Simply gifted: Their attributes through the eyes of college students.* Online Submission. Paper presented at the International Conference "Peace through Understanding" (Jackson, MS, Apr 4-8, 2011)

Ghazi, S., Shahzada, G., Gilani, U., Shabbir, M., & Rashid, M. (2011). Relationship between students self-perceived multiple intelligences and their academic achievement. *International Journal of Academic Research*, *3*(2), 619-623.

Goldrick-Rab, S. (2010). Challenges and opportunities for improving community college student success. *Review of Educational Research*, *80*(3), 437-469.

Goleman, Daniel. (1986). Rethinking the value of intelligence tests. *New York Times Educational Life supplement.*

Gregory, G. H., & Chapman, C. (2012). *Differentiated instructional strategies: One size doesn't fit all*. New York, NY: SAGE Publications.

Griffiths, I. (2013). Chapter 14: Dynamic typing. In *Programming C# 5.0: Building Windows 8, web, and desktop applications for the .NET 4.5 framework* (509-511). Sebastopol, CA: O'Reilly.

Groff, J. S. (2013). Expanding our "frames" of mind for education and the arts. *Harvard Educational Review, 83*(1), 15-39. 266.

Hakimzadeh, H., Adaikkalavan, R., & Batzinger, R. (2011). Successful implementation of an active learning laboratory in computer science. In *Proceedings of the 39th Annual ACM SIGUCCS Conference* (83-86). Association of Computing Machinery.

Haonkonen, J. (2013). If you think “computers” is a hobby, I may have bad news for you. *Vornasblogi.* Retrieved from http://blog.vornaskotti.com/2013/08/02/if-you-think-computers-is-a-hobby-i-may-have-bad-news-for-you/

Hars, A., & Ou, S. (2001). Working for free? Motivations of participating in open source projects. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on IEEE* (9).

Harvey, K., Bright, S., &Wamsley, A. (2014). Meeting federal financial aid regulations: Attendance and participation tracking. *Annual Conference Collection of Papers 2015.* Higher Learning Commission.

Haungs, M., Clark, C., Clements, J., & Janzen, D. (2012). Improving first-year success and retention through interest-based CS0 courses. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (589-594). Association of Computing Machinery.

Hazzan, O., Lapidot, T., & Ragonis, N. (2011). *Guide to teaching computer science: An activity-based approach*. New York, NY: Springer.

Hoda, R., & Andreae, P. (2014). It's not them, it's us! Why computer science fails to impress many first years. In *Australasian Computing Education,* (159-162).

Hoskey, A., & Maurino, P. S. (2011). Beyond introductory programming: Success factors for advanced programming. *Information Systems Education Journal, 9*(5), 61-70. http://isedj.org/2011-9/ ISSN: 1545-679X.

Hu, C. (2011). Computational thinking: What it might mean and what we might do about it. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (223-227). Association of Computing Machinery.

IBM SPSS Statistics (2015). *International business machines corporation*. Retrieved from http://www-01.ibm.com/software/analytics/spss/

Jackson, J. (2014). IDC: Hobbyist programmers on the rise. *PCWorld*. Retrieved from http://www.pcworld.com/article/2082140/idc-hobbyist-programmers-on-the-rise.html

Jenkins, P. D., & Cho, S. W. (2012). *Get with the program: Accelerating community college students' entry into and completion of programs of study. CRCC working paper no. 32.* Community College Research Center (CCRC), Columbia University.

Jesse, D. (2013). Colleges chase pell grant scammers. *USA Today.* Retrieved from http://www.usatoday.com/story/news/nation/2013/02/16/colleges-chase-pell-grant-scammers/1925013/

Jones, K. (2001). A brief informal history of the computer laboratory. *University of Cambridge*. Retrieved from http://www.cl.cam.ac.uk/events/EDSAC99/history.html

Kay, J. S. (2011). Contextualized approaches to introductory computer science: The key to making computer science relevant or simply bait and switch?. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (177-182). Association of Computing Machinery.

Kazimoglu, C., Kiernan, M., Bacon, L., & Mackinnon, L. (2012). A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Social and Behavioral Sciences*, *47*, 1991-1999.

Kendall, K. D., & Schussler, E. E. (2012). Does instructor type matter? Undergraduate student perception of graduate teaching assistants and professors. *CBE-Life Sciences Education*, *11*(2), 187-199.

Korkmaz, O. (2012). The impact of critical thinking and Logico-Mathematical Intelligence on algorithmic design skills. *Journal of Educational Computing Research, 46*(2), 173-193.

Kossey, J., & Brown, V. (2012). QR Codes: A quick response for education. In *Society for Information Technology & Teacher Education International Conference* (*1*, 3676-3680).

Lack, A. P., Bruce, K. B., Homer, M., Noble, J., Ruskin, A., & Yannow, R. (2013). Seeking grace: A new object-oriented language for novices. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (129-134). Association of Computing Machinery.

Lail, A. A. (2009). Are new faculty prepared to teach diverse learners? *Inquiry, 14*(1), 29–40.

Latham, A. S. (1997). Quantifying MI's gains. *Educational Leadership*, *55*(1), 84-85.

Leal, J. P. (2013). Testing the perception of time, state and causality to predict programming aptitude. In *Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on* (721-726). IEEE.

Leufer, T. & Cleary-Holdforth, J. (2010) Reflections on the experience of mandating lecture attendance in one school of nursing in the Republic of Ireland. *All Ireland Journal of Teaching and Learning in Higher Education, 2*(1), 18.1–18.14.

Levison, A. (1995). Making the transition from computer hobby to serious business. *Online*, *19*(4), 14-21.

Linden Lab (2015). What is second life? Retrieved from http://secondlife.com/whatis/

Long, B. T. (2014). Proposal 6: Addressing the academic barriers to higher education: policies to address poverty in America. *The Hamilton Project, Harvard School of Education*.

Macfarlane, B. (2013). The surveillance of learning: A critical analysis of university attendance policies. *Higher Education Quarterly*, *67*(4), 358-373.

Maguire, P., Maguire, R., Hyland, P., & Marshall, P. (2014). Enhancing collaborative learning using pair programming: Who benefits? *AISHE-J: The All Ireland Journal of Teaching and Learning in Higher Education*, *6*(2).

Manley, E. D., & Urness, T. M. (2014). Video-based instruction for introductory computer programming. *Journal of Computing Sciences in Colleges*, *29*(5), 221-227.

Marburger, D. R. (2006). Does mandatory attendance improve student performance? *Journal Of Economic Education*, *37*(2), 148-155.

Mazlack, L. J. (1980). Identifying potential to acquire programming skill. *Communications of the ACM*, *23*(1), 14-17. Association of Computing Machinery

McClenney, K., Marti, C. N., & Adkins, C. (2012). Student engagement and student outcomes: Key findings from CCSSE validation research. *Community College Survey of Student Engagement*.

McFarlane, D. A. (2011). Multiple intelligences: The most effective platform for global 21st century educational and instructional methodologies. *College Quarterly*, *14*(2).

McKinney, L., & Novak, H. (2015). FAFSA filing among first-year college students: Who files on time, who doesn't, and why does it matter? *Research in Higher Education*, *56*(1), 1-28.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239-264.

Melguizo, T., Bos, J., & Prather, G. (2011). Is developmental education helping community college students persist? A critical review of the literature. *American Behavioral Scientist*, *55*(2), 173-184.

Missouri Department of Higher Education. (2014). Fall 2014 enrollment report for Missouri comprehensive public & independent institutions. *Missouri Department of Higher Education*. Retrieved from http://dhe.mo.gov/data/

Mok, H. N. (2014). Teaching tip: The flipped classroom. *Journal of Information Systems Education*, *25*(1), 7.

Monaghan, D. B., & Attewell, P. (2015). The community college route to the bachelor's degree. *Educational Evaluation and Policy Analysis*, *37*(1), 70-91.

Morris, D. B. (2012). *Community college selective enrollment and the challenge to open access* (Doctoral dissertation). Retrieved from ProQuest Dissertations and Theses. (OCLC Number. 828856498).

National Science Foundation. (2012). Science and engineering indicators 2012. *National Science Foundation.* Retrieved from http://www.nsf.gov/statistics/seind12/c2/c2s2.htm

Nilsen, H., & Larsen, E. Å. (2011). Using the personalized system of instruction in an introductory programming Course. *NOKOBIT*.

Nikula, U., Gotel, O., & Kasurinen, J. (2011). A motivation guided holistic rehabilitation of the first programming course. *ACM Transactions on Computing Education (TOCE)*, *11*(4), 24. Association of Computing Machinery

NPR Staff. (2014). Computers are the future, but does everyone need to code? *NPR: All Tech Considered*. Retrieved from http://www.npr.org/sections/alltechconsidered /2014/01/25/266162832/computers-are-the-future-but-does-everyone-need-to-code

Oram, A., & Greg, W. (2011). *Making software: What really works, and why we believe it.* Sebastopol, CA: O'Reilly Media.

Piteira, M., & Costa, C. (2012). Computer programming and novice programmers. In *Proceedings of the Workshop on Information Systems and Design of Communication* (51-53). Association of Computing Machinery.

Pollack, G. (2014, April 16). No, not everyone needs to learn to code - But here's what they should know. *Huffington Post*. Retrieved from http://www.huffingtonpost.com/gregg-pollack/no-not-everyone-needs-to-_b_5155549.html

Porter, L., Bailey-Lee, C., & Simon, B. (2013a). Halving fail rates using peer instruction: A study of four computer science courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (177-182). Association of Computing Machinery.

Porter, L., Guzdial, M., McDowell, C., & Simon, B. (2013b). Success in introductory programming: What works? *Communications of the Association of Computing Machinery*, *56*(8), 34-36.

Radermacher, A., Walia, G., & Rummelt, R. (2012). Improving student learning outcomes with pair programming. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (87-92). Association of Computing Machinery.

Reese, D. S., Jankun-Kelly, T. J., Henderson, L., & Lee, S. (2013). Impact on retention from a change in undergraduate computing curricula. In *Proceedings of the 2013, ASEE Southeast Section Conference*. American Society for Engineering Education

Rizvi, M., Humphries, T., Major, D., Jones, M., & Lauzun, H. (2011). A cs0 course using Scratch. *Journal of Computing Sciences in Colleges*, *26*(3), 19-27.

Scherer, J. L., & Anson, M. L. (2014). *Community colleges and the access effect: Why open admissions suppresses achievement*. New York, NY: Palgrave Macmillan.

Scott-Clayton, J. (2011). The shapeless river: Does a lack of structure inhibit students' progress at community colleges? *CCRC Working Paper No. 25, Assessment of*

Evidence Series. New York, NY: Columbia University, Teachers College, Community College Research Center.

Scott, M. J., & Ghinea, G. (2013). Implicit theories of programming aptitude as a barrier to learning to code: are they distinct from intelligence?. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (347-347). Association of Computing Machinery.

Scott, M. J., & Ghinea, G. (2014). Measuring enrichment: the assembly and validation of an instrument to assess *student self-beliefs in CS1. In Proceedings of the Tenth Annual Conference on International Computing Education Research* (123-130). Association of Computing Machinery.

Shein, Esther. (2014). Should *everybody* learn to code? *Communications of the ACM, 57*(2), 16-18. Association of Computing Machinery

Snow, R. E. (1992). Aptitude theory: Yesterday, today, and tomorrow. *Educational Psychologist*, *27*(1), 5-32.

Spearman, C. (1904). "General intelligence," objectively determined and measured. *American Journal of Psychology, 15*(2), 201–293. DOI: 10.2307/1412107

Sternberg, R.J. (2004). Definition of intelligence. *Human Intelligence: Historical Influences, Current Controversies, Teaching Resources.* Retrieved from http://www.intelltheory.com/

Stewart-Gardiner, C. (2011). Improving the student success and retention of under achiever students in introductory computer science. *Journal of Computing Sciences in Colleges, 26*(6), 16-22.

Stigler, J. W., Givvin, K. B., & Thompson, B. J. (2010). What community college developmental mathematics students understand about mathematics. *MathAMATYC Educator*, *1*(3), 4-16.

Tabarrok, A. (2012). Tuning in to dropping out. *The Chronicle of Higher Education*. Retrieved from http://chronicle.com/article/Tuning-In-to-Dropping-Out/130967/

Tarrant County College. (2015). Mandatory attendance policy. *Courses and Programs*. Retrieved from http://www.tccd.edu/Courses_and_Programs/Mandatory_Attendance/

Teague, D. & Lister, R. (2014). Longitudinal think aloud study of a novice programmer. *Sixteenth Australasian Computing Education Conference,* (ACE '14), 41-50.

Tillmann, N., De Halleux, J., Xie, T., & Bishop, J. (2012a). Pex4Fun: Teaching and learning computer science via social gaming. In *Software Engineering Education and Training (CSEE&T), 2012 IEEE 25th Conference* 90-91.

Tillmann, N., Moskal, M., de Halleux, J., Fahndrich, M., Bishop, J., Samuel, A., & Xie, T. (2012b). The future of teaching programming is on mobile devices. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education* 156-161

U.S. Copyright Office. (2014). *Definitions*. Copyright Law of the United States of America. Retrieved from http://www.copyright.gov/title17/92chap1.html#101

Vaughn, G. B. (2006). *The community college story* (3rd ed.). American Association of Community Colleges.

Veilleux, N., Bates, R., Allendoerfer, C., Jones, D., Crawford, J., & Floyd Smith, T. (2013). The relationship between belonging and ability in computer science.

In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (65-70). Association of Computing Machinery.

Vihavainen, A., Paksula, M., & Luukkainen, M. (2011). Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (93-98). Association of Computing Machinery.

Walker, H. M., Hirakawa, A., & Steinbach, R. (2011). A system to place incoming students in computer science, mathematics and statistics. *Journal of Computing Sciences in Colleges*, *27*(1), 24-31.

Watson, C., & Li, F. W. (2014). Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (39-44). Association of Computing Machinery.

Weimer, M. (2012). Do Attendance Policies Influence Student Learning? Attendance Policies: Research Update *The Teaching Professor, 25*(5), 4-5

Wulf, T. (2005). Constructivist approaches for teaching computer programming. In *Proceedings of the 6th Conference on Information Technology Education* (245-248). ACM.

Yates, K. J. (2010), "Graduation rates: A comparison of first-time, full-time freshmen who entered a community college prepared and those who entered underprepared for college-level work." *Electronic Theses and Dissertations.* Paper 1674.

Young, J. R. (2002). Homework? What homework. *The Chronicle of Higher Education*, *49*(15), A35-A37.

Zeidenberg, M. (2015). Valuable learning or "spinning their wheels?" Understanding excess credits earned by community college associate degree completers. *Community College Review*, *43*(2), 123-141.

Vita

Tiffany Ford completed an Associate degree in Computer Information Technology and an Associate degree in Internet Application Development from The Ozarks Technical Community College (OTC), in 2005. She completed a Bachelor of Applied Science degree in Technology Management from Missouri State University in 2007, and she completed a Master of Science degree in Career and Technical Education Leadership from the University of Central Missouri, in 2010. Tiffany currently serves as the Department Chair of Computer Information Science at OTC and has held that position since August, 2011. She holds a lifetime certification to teach technical education in the State of Missouri.

Tiffany was a faculty member in the Computer Information Science department at the Springfield campus of OTC from 2007 through 2011. Before her transition into teaching, she worked in OTC's Information Technology department as the Desktop Deployment Systems Administrator from 2005 through 2007. Prior to working at OTC, Tiffany held various positions as a computer programmer at companies in the Springfield area.

Tiffany is currently a member of the following professional organizations: Association for Information Technology Professionals, Association of Computing Machinery, Association for Women in Computing, and the Missouri Community College Association. She is also a SkillsUSA advisor, works with Springfield area professional User Groups for computer programming, and is involved with local interest groups to promote computer science to young girls.