Using Visual Technologies in the Introductory Programming Courses for Computer Science Majors

by

Kellie W. Price

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computing Technology in Education

Graduate School of Computer and Information Sciences
Nova Southeastern University
2013

An Abstract of a Dissertation Submitted to Nova Southeastern University
in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Using Visual Technologies in the Introductory Programming Courses for Computer Science Majors

by
Kellie W. Price
April 2013

Decreasing enrollments, lower rates of student retention and changes in the learning styles of today's students are all issues that the Computer Science (CS) academic community is currently facing. As a result, CS educators are being challenged to find the right blend of technology and pedagogy for their curriculum in order to help students persist through the major and produce strong graduates.

Visual technologies are being explored as a way to present difficult programming concepts in a manner that is easier to visualize and simpler to use. Visual technologies can make learning programming easier by minimizing the syntax of the programming language being used and providing visual feedback to the students to aid in conceptualization of the programming constructs.

The goal was to improve student retention and performance by incorporating visual technologies in the introductory programming course, CS1, at East Tennessee State University (ETSU). The ADDIE approach to instructional design was used to develop and implement a curriculum that incorporated visual technologies in CS1 at ETSU. Subsequently, quasi-experimental research methods, using the Post-Test Only Nonequivalent Groups Design approach, were used to perform assessment on the effects of the revised curriculum on student performance in the course and retention in the major as compared to student performance and retention as measured prior to the course redesign.

The results of the study indicate a positive impact on student performance in CS1 and student retention in the major as a result of the use of two types of visual technologies in CS1 at ETSU. Visual technologies supporting algorithm development, such as RAPTOR, had a positive impact on student performance in the area of problem solving and algorithm development as well as the use of decision and repetition constructs in programming. Visual technologies supporting program development, such as Alice, had a positive impact on student performance in the area of object-oriented programming concepts such as objects and classes. The combination of these two types of visual technologies showed evidence of improvement among student performance as a whole in the course and slight improvement in student persistence in the major.

Acknowledgements

"I can do all things through Christ which strengtheneth me." Phil. 4:13
I owe God all of the glory, honor and praise for every accomplishment in my life.

I am forever grateful to everyone who has been there for me with your prayers, friendship, love and support. This accomplishment is as much yours as it is mine.

I owe my advisor, Dr. Trudy Abramson, a debt of gratitude for not giving up on me and for patiently encouraging and supporting me at a time when I was ready to give up. You have been an inspiration to me and I am very thankful for having had the pleasure of working with you. Without you, I would not have reached this goal. Thank you also to Dr. Amon Seagull and Dr. Antonio Rincon for serving on my committee. Your patience, encouragement and feedback were very much appreciated and crucial to my success.

Thank you to Dr. Donald Sanderson, Dr. Suzanne Smith and Dr. Sarah Stoeklin for helping me identify and develop an idea that I was passionate about. I would especially like to thank Dr. Smith for picking up where Dr. Sanderson left off. You were an unbelievably dedicated mentor without whom I would not have been successful. Thank you also to my colleagues, Dr. Tony Pittarese and Dr. Marty Barrett for helping with editing and advice; Dr. Edith Seier for providing crucial guidance with the statistical areas of the research; all of my colleagues at ETSU who have fully supported me along the way; and Dr. Terry Countermine and Mrs. Rikki Cornett Young for sharing the journey with me throughout my entire college career. You have both been an incredible source of inspiration, strength, encouragement and friendship every step of the way.

Thank you to my family and friends who have been my prayer warriors, my constant source of encouragement and inspiration and who never gave up on me. God has blessed me with a support system of so many incredible family members and friends that I could never name them all. To Julieann, Amy, Sonya, Kim, Jana and Melissa – thank you for your unwavering love and support and for being there for me in every way imaginable.

Dedication

To my children, Ashleigh and Noah
You are my inspiration, my joy and my special gifts from God
who have sacrificed more than anyone along this journey

To Mama, Daddy, Momaw, Granddaddy, Sissy, Sara, Brooke, Scot and my entire family
Your constant love, support, patience, encouragement and prayers
have made all of my accomplishments possible

To Gran and Dr. Donald Sanderson
You were two very special sources of inspiration, strength and encouragement,
and though you are not here to share in the joy of this accomplishment with me today,
you are rejoicing with me in my heart

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Background

Traditionally, the computer science (CS) curriculum in colleges and universities begins with a two-semester sequence of programming courses. These courses, commonly known as CS1 and CS2, are designed to teach the basics of program development and a programming language. For the first course, CS1, the main focus of program development is the design of algorithms. An algorithm is a step-by-step procedure for solving a problem and serves as the blueprint from which a program is developed. CS1 also introduces basic programming concepts such as selection, repetition, and input and output statements. Upon successful completion of CS1, students are able to design and develop basic programs in a given programming language. In the CS2 course, more advanced programming and development concepts are introduced such as data structures, files, error handling and debugging. Upon successful completion of CS2, students are able to design, develop and test intermediate-level programs in a given programming language. Typically, the same programming language is used for the two-semester sequence.

During the past decade, the CS academic community has experienced higher failure rates and lower student retention in the CS1 and CS2 courses (Zweben, 2008; Vegso, 2008; Yadin, 2011; Soe, Guthrie, Yakura, & Hwang, 2011; Guthrie, Yakura, &

Soe, 2011; Becerra-Fernandez, Elam, & Clemmons, 2010; Sloan & Troy, 2008; Ali, 2009; Moskal, Lurie & Cooper, 2004; Forte & Guzdial, 2004; Chen & Morris, 2005; Herrmann et al., 2003; Talton, Peterson, Kamin, Israel, & Al-Muhtadi, 2006; Boyer, Dwight, Miller, Raubenheimer, Stallman, & Vouk, 2007). Student retention in this context refers to the successful completion of CS1 and progression to CS2. A student who is not retained is one who does not successfully complete CS1 and chooses not to repeat it or a student who does successfully complete CS1 but chooses not to progress to CS2 and subsequently leaves the major or the university. To address these two problems, various solutions have been proposed. The focus of many solutions is improving students' understanding of algorithm development and programming concepts through changes in the curriculum or the use of visual technologies.

The investigation site was the Department of Computing at East Tennessee State University (ETSU). ETSU is a regional university in northeast Tennessee offering over 100 programs of study, including two-year associate degrees, bachelor's, master's, educational specialist, doctor of medicine, doctor of pharmacy, doctor of education, and doctor of philosophy degrees. The Department of Computing at ETSU is housed within the College of Business and Technology. The department offers undergraduate degrees in three different concentrations, Computer Science, Information Systems and Information Technology and graduate degrees in Computer Science and Information Technology. According to the Fall 2011 issue of *The ETSU Fact Book* (http://www.etsu.edu/opa/factbook.aspx), there were 383 undergraduate and 52 graduate majors in the CS department.

Problem Statement

Higher rates of student failure and lower rates of student retention in introductory programming courses are significant problems in the CS academic community (Yadin, 2011; Soe et al., 2011; Guthrie et al., 2011; Becerra-Fernandez et al., 2010; Sloan & Troy, 2008; Ali, 2009). These problems become even more significant when coupled with declining enrollment rates. According to the Computing Research Association (Zweben, 2008), the number of new undergraduates declaring CS as a major dropped dramatically from 2000 to 2006 by as much as 53% in Ph.D. granting institutions. The decline is even greater among all degree-granting institutions where the percentage dropped by approximately 70% between 2000 and 2005 (Vegso, 2008). The ACM and IEEE Computer Science Curriculum Revision (2008) reported that there was a decrease in the number of graduates because enrollments had dropped by as much as 70% from their peak in 2001. While the number of students declaring CS as a major in Ph.D. granting institutions continued to decline throughout 2007, a slight increase has been seen between 2007 and 2010 according to the Computing Research Association. However, about one-third of the departments surveyed are still reporting decreases in total enrollment (Zweben, 2011).

The decline in new undergraduates entering the major has been anecdotally attributed to factors such as the dot-com bust and outsourcing (Manaris, 2007; Yadin, 2011; Soe et al., 2011; Becerra-Fernandez et al., 2010). However, trends in CS education beginning a few years prior to 2001 may have had a significant impact on the recruitment and, more importantly, the retention of students (Manaris, 2007). Prior to 2001, relatively simple programming languages such as Pascal, which was designed specifically for teaching introductory programming concepts, were used in CS1 and CS2. After 2001, the trend became to use complex, industry-strength programming languages such as Java and C++ (Yadin, 2011; Blake, 2011; Soe et al., 2011; Sigle, 2008; Ali, 2009). According to McCauley and Manaris (2002), the use of industry-strength programming languages in the introductory programming courses had increased from 39% in 1996 to 89% in 2002 and their use continues to be popular today.

Another trend that may have had a negative effect on recruitment and retention of CS students was the switch from a procedural paradigm to an object-oriented paradigm for program development (Manaris, 2007; Yadin, 2011; Ali, 2009). A procedural paradigm is a relatively simple approach to algorithm development and program design which focuses on procedures. The object-oriented paradigm is a more complex abstract approach which focuses on objects and classes. The switch from the procedural paradigm to the object-oriented paradigm has required additional topics to be taught in the introductory programming courses. The use of the object-oriented paradigm increased from 36% in 1995 to 82% in 2002 (Manaris, 2007) and continues to be heavily utilized today.

The impact of these two trends in CS education has been significant in the introductory programming courses. The result has been an increase in the number and difficulty of the topics being presented and an increase in the complexity of the programming languages being used (Manaris, 2007; Yadin, 2011; Sigle, 2008). These factors may have had an effect on the failure rate thus potentially impacting the retention of majors.

In addition to the difficulties caused by these two trends, the learning style of today's students has drastically changed. This generation of students, often referred to as Millenials (Stamey & Sheel, 2010; Oblinger, 2003; Frand, 2000), tend to be visual learners who prefer technology to textbooks, view active participation as more important than obtaining knowledge, and expect to learn in the same manner in which they live – immersed in animation and graphics through games and other recreational activities (Howles, 2007; Sigle, 2008) – approaches not typically used in introductory programming courses.

In an attempt to address the issues associated with the new learning styles of CS majors, the use of more complex programming languages and the exposure to more difficult programming concepts in the introductory programming courses, many CS educators are exploring new visual technologies to enhance student learning. There are a variety of different technologies, but most can be categorized as those used to enhance algorithm development (Gudmundsen, Olivieri, & Sarawagi, 2011; Carlisle, 2009; Yoo, Yoo, Seo, & Pettey, 2011) and those used to support programming concepts (Yadin, 2011; Stolee & Fristoe, 2011; Davies, Polack-Wahl, & Anewalt, 2011; Guthrie et al., 2011).

Some of the visual technologies being incorporated into CS courses support programming development through the use of visual programming environments. Examples include Kodu (Stolee & Fristoe, 2011), Alice (Davies et al., 2011; Guthrie et al., 2011), Greenfoot (Davies et al., 2011), and Scratch (Davies et al., 2011; Guthrie et al., 2011). Visual technologies being used to enhance algorithm development include Visual Logic (Gudmundsen et al., 2011), RAPTOR (Carlisle, 2009), and AlgoTutor (Yoo et al., 2011). Others are using combinations of visual technologies such as Alice and robotics (Wellman, Davis, & Anderson, 2009). In addition to using visual programming environments or visual algorithm development tools, some universities and colleges are also using other visual technologies such as board games (Drake & Sung, 2011) and visual learning objects/modules (Miller, Soh, Samal, Nugent, Kupzyk, & Masmaliyeva, 2011; Stone & Clark, 2011; Yim, Garcia, & Ahn, 2010) to capture the attention of students who are visual learners.

CS educators are also exploring other pedagogical approaches to enhance student learning such as adding a third course to the introductory programming course sequence. This third course, commonly known as CS0, is a prerequisite for the traditional CS1. The focus of this course is algorithm development and basic programming concepts (Pearce & Nakazawa, 2008; Sloan & Troy, 2008). It allows many of the topics in CS1 to be introduced in CS0, thus reducing the number of new concepts taught in the CS1 course. Many CS0 courses use the visual technologies previously listed and have reported a positive effect on the success of computer science students in CS1 (Pearce & Nakazawa, 2008; Sloan & Troy, 2008).

The Department of Computing at ETSU has experienced the same trends and issues in enrollment and retention as other CS departments nationwide, as seen in university data (http://www.etsu.edu/iep/fb.htm). There was a 26% decrease in undergraduate enrollment from 2000 to 2009. During that same time span, the number of degrees conferred increased by 48% from 2000 to 2005 but then decreased by 16% from 2005 to 2009 due to the decrease in enrollment. Many changes occurred within the department, as they did in CS departments nationwide, including a switch to the object-oriented programming paradigm and the adoption of an industry-strength programming language in the introductory programming courses.

The retention rates in the introductory programming courses at ETSU also reflect the rates nationwide. From Fall 2009 to Spring 2011, 20% of the students enrolled in the introductory programming courses were not retained in the major beyond their first semester and 35% of the students were not retained in the major beyond the freshman year (T. Countermine, ETSU CS department chair, personal communication, July 20, 2011). As a result, the CS department at ETSU has tried different approaches to increasing the student success and retention rates. Like many other CS departments nationwide, ETSU has incorporated a CS0 course which uses visual technologies such as RAPTOR and Alice to introduce basic programming and algorithm concepts to students who are not prepared to enter the CS1 programming course. However, the use of these visual technologies has not been implemented beyond the CS0 course.

Decreasing enrollments, lower rates of student retention and changes in the learning styles of today's students are all issues that the CS academic community is currently facing. As a result, CS educators are being challenged to find the right blend of technology and pedagogy for their curriculum in order to help students persist through the major and produce strong graduates.

Dissertation Goal

The goal was to improve student retention and achievement with curricula that incorporate visual technologies beyond the CS0 programming course. The steps taken were identification of the factors contributing to the reduction in the retention rate for introductory programming courses in the CS curriculum, development of a list of the most popular visual technologies currently used in introductory programming courses to address retention problems, and evaluation of those technologies based upon the technology's level of appropriateness for CS1, coverage of CS1 concepts, usability and resource requirements.

Based on those findings, the generic analysis, design, development, implementation, evaluation (ADDIE) instructional design approach was followed to develop a new CS1 course that incorporated visual technologies. Quasi-experimental research was performed to evaluate the new instructional approach, focusing on student achievement and retention.

Research Questions

All questions relate to the introductory programming course, CS1, in a CS curriculum.

1. What are the factors attributable to poor performance and low retention rates and what solutions have been reported?
2. How can the introductory course, CS1, be redeveloped and implemented to incorporate visual technologies?
3. What are the outcomes of teaching the redesigned course?
4. What conclusions may be drawn regarding the value of the new introductory curricula in terms of student performance and retention?

Relevance and Significance

CS education has reached a critical point where the gap between established teaching methods and learning styles of today's CS students is greater than ever. This gap between instructional delivery and student learning, which will be referred to as the Millennial CS Education Gap, can have a profound effect on the enrollment numbers in CS departments nationwide (Dillon, Anderson, & Brown, 2012). For departments that may be experiencing declining enrollment and retention statistics, it is disconcerting that the current economic environment may eventually force enrollment and retention rates to become a significant factor in their funding formula. Therefore, it is crucial to the success of these departments that CS educators find a way to bridge the Millennial CS Education Gap.

Many are trying to bridge this gap through increased learning and retention with the development of simpler, more intuitive integrated development environments (IDEs) and visual approaches to learning (Dillon et al., 2012; Gudmundsen et al., 2011; Sigle, 2008; Carlisle, 2009; Yoo et al., 2011; Stolee & Fristoe, 2011; Davies et al., 2011). However, the tools alone cannot bridge the gap between student learning and instructional delivery. These tools must be integrated into the course in a manner that will maximize their potential for increasing student understanding of programming concepts, thus increasing student learning and ultimately having a positive effect on student retention and enrollment rates.

Limitations and Delimitations

The treatment group consisted of all students enrolled in the CS1 course at ETSU during the Fall 2012 semester. The number of sections of CS1 offered and the number of students enrolled in those sections was beyond the control of the investigator. In addition, while it was guaranteed that at least two professors would be teaching CS1 during the Fall 2012 semester, the number of sections assigned to each professor was another limitation beyond the control of the investigator.

Definitions and Acronyms

Acronyms

ABET: Accreditation Board for Engineering and Technology

ACM: Association for Computing Machinery

ADDIE: Analysis, Design, Development, Implementation and Evaluation

CS: Computer Science

ETSU: East Tennessee State University

HTML: Hypertext Markup Language

IDE: Integrated Development Environment

IEEE: Institute of Electrical and Electronics Engineers

OO: Object-Oriented

UML: Unified Modeling Language

Definitions

Algorithm: a well-defined, ordered set of steps for solving a specific problem (Gaddis, 2011b).

Class: a description of a particular type of object that serves as a blueprint for creating and using objects in an object-oriented programming language. For example, a blueprint that provides a detailed description of a house to be built is a real-world example of a class (Gaddis, 2011a).

Compile: the process in which the source code, written in a programming language, is checked for errors and, if error-free, is translated into a form that can be executed by the computer (Gaddis, 2011b).

Count-controlled repetition: a form of looping in which a block of code is repeated a specific number of times (Gaddis, 2011b).

CS majors/CS students: for this research, this refers to students who have declared Computing as their major at ETSU. It includes students in all three of the concentrations, Computer Science, Information Systems, and Information Technology, housed in the Department of Computing Sciences at ETSU (Author).

CSCI Introductory Programming Committee: a committee in the Department of Computing at ETSU whose charge is to make decisions regarding the introductory programming courses within the department. Other responsibilities of the committee include making sure all sections of a particular programming course are kept in sync, ensuring continuity throughout the sequence of introductory programming courses, language and textbook adoption, and acting on issues that arise regarding student learning, success and retention. Members of this committee include three tenured Full Professors, three tenured Assistant Professors and one Lecturer. All members of the committee are actively involved in teaching one or more of the courses in the introductory programming sequence (Author).

CS0: a typical pre-programming course completed before the first semester programming course that focuses on algorithm development and basic programming concepts. It is generally taught using only pseudocode or visual programming technologies to avoid the use of complex syntax while learning basic programming concepts (Mitchell, 2001; Davies et al., 2011).

CS1: typical first semester programming course that introduces basic programming concepts (Davies et al., 2011).

CS2: typical second semester programming course that introduces more advanced programming and development concepts such as data structures, files, error handling and debugging (Davies et al., 2011).

Debugging: the process of identifying and correcting errors in a computer program (Sprankle & Hubbard, 2009).

Desk checking: the process of manually checking an algorithm for logic errors using a pen-and-paper technique (Author).

Drag-and-drop: a feature in an environment where operations can be performed by dragging visual objects, such as blocks of text, icons or programming instructions, across the screen and dropping them in a new location (Parker, 2002).

Event-controlled repetition: a form of looping in which a block of code is repeated until a certain condition is true (Gaddis, 2011a).

Flowchart: a diagram that graphically depicts the sequence of steps in an algorithm (Gaddis, 2011a)

IDE: Integrated Development Environment. Software that includes a text editor, compiler, debugger and other programming features all in one integrated package allowing for tasks to be performed by the click of a button (Gaddis, 2011b).

Industry-strength programming languages: programming languages that are not designed specifically for the purpose of learning to program but for actual use in the software development industry (Author).

Iteration: (aka Repetition or Looping) a programming construct that causes a set of statements to be repeated (Gaddis, 2011b).

Javadoc documentation: formatted HTML documents that display the documentation (programmer comments) taken from the Java source code in a more readable format (Gaddis, 2011b).

Methods: the behaviors of an object or procedures that it can perform (Gaddis, 2011b).

Object: an object is a specific instance of a class. It is created from the class and has all of the characteristics of the class. For example, a house built using a blueprint is a real-world example of an object (Gaddis, 2011a).

Object-oriented paradigm: a more modern, complex, abstract approach to programming that promotes code that is reusable and is based upon the use of objects and classes (Gaddis, 2011b).

Objects-first: the sequence in which programming topics are presented in a CS1 course where the concept of objects and classes is introduced early in the sequence of topics before other basic procedural programming concepts have been introduced (Beaubouef & Mason, 2005).

Objects-later: the sequence in which programming topics are presented in a CS1 course where the concept of objects and classes is introduced later in the sequence of topics after other basic procedural programming concepts have been introduced (Beaubouef & Mason, 2005).

Post-condition event-controlled looping: a type of repetition in which the block of code is executed before checking the condition to determine if it should be repeated (Gaddis, 2011a).

Pre-condition event-controlled looping: a type of repetition in which the condition is checked before executing the block of code to determine if it should executed/repeated (Gaddis, 2011a).

Procedural paradigm: a relatively simple approach to algorithm development and program design focused on the use of procedures to perform tasks (Gaddis, 2011b).

Pseudocode: an algorithmic solution written in an outline format in a cross between human language and programming language (Gaddis, 2011b).

Selection: (aka Decision Structure) a programming construct in which one of two sets of instructions is executed based upon the result of some condition (Sprankle & Hubbard, 2009).

Student retention: the successful completion of CS1 and progression to CS2 (Author).

Syntax errors: mistakes made by the programmer that violate the rules of a programming language (Gaddis, 2011b).

UML: provides a standard set of diagrams used to graphically depict classes and their relationships in an OO system (Gaddis, 2011b).

Variable: a named location for storage of data in a computer's memory (Gaddis, 2011b).

Visual algorithm development tools: tools that use drag-and-drop technology, graphics and visual representations of programming constructs to support the algorithm development process (Author).

Visual development environment: an integrated development environment that has some of the following additional features to support the programming process - drag-and-drop technology, visual representations of programming constructs, built-in UML modeling tools, graphics, the ability to create animations and programs (Author).

Organization of the Study

The study began by identifying and analyzing both the factors affecting student success and retention in introductory programming courses and the solutions that had been proposed to address these factors. One category of solutions in particular, the use of visual technologies, was examined in relation to their use and effectiveness as it related to success and retention in the introductory programming courses.

Based upon the findings presented in the review of literature, a CS1 course was developed using the ADDIE approach to instructional design that incorporated the solutions utilizing visual technologies that had been reported as most effective and that were most appropriate for the use in CS1 based upon factors such as the coverage of CS1 concepts, usability and resource requirements. Details about the quasi-experimental research that was conducted to evaluate the effectiveness of the new instructional approach once implemented is also presented.

Chapter 2

Review of Literature

The literature was reviewed with respect to the introductory programming courses in CS, factors affecting student success and retention and the solutions proposed to address these factors. There are many reasons that students may drop, fail or withdraw from college-level classes including personal, financial, and employment related issues. These factors can affect any student in any major. However, there are additional factors that particularly affect CS students. The common factors across colleges and universities that seem to have a significant impact particularly during their freshman year are misconceptions about the CS field (Beaubouef & Mason, 2005; Ruslanov & Yolevich, 2010; Biggers, Brauer & Yilmaz, 2008), being under-prepared due to a lack of problem solving and mathematical abilities (Sloan & Troy, 2008; Beaubouef & Mason, 2005; Moskal et al., 2004), number and complexity of topics being introduced (Urness & Manley, 2011; Yadin, 2011; Ali, 2009; Sigle, 2008; Manaris, 2007; Chen & Morris, 2005), the use of industry-strength programming languages (Beaubeouf & Mason, 2005; Manaris, 2007; Carlisle, 2009; Yadin, 2011; Blake, 2011; Soe et al., 2011; Sigle, 2008; Ali, 2009; McCauley & Manaris, 2002; Moskal et al., 2004), poorly designed introductory programming courses (Carlisle, 2009; Beaubouef & Mason, 2005; Gal-Ezer & Harel, 1998), and teaching and delivery styles that do not relate to today's visual learners (Beaubouef & Mason, 2005; Stamey & Sheel, 2010; Oblinger, 2003; Frand, 2000; Howles, 2007; Sigle, 2008; Carlisle, Wilson, Humphries & Hadfield, 2004; Cardellini, 2002). The literature review examined these problems, the proposed solutions and visual technology's role in these solutions.

Misconceptions about the CS Field

Many students are not retained beyond the first semester because of misconceptions about the CS field (Beaubouef & Mason, 2005). The students may be very technology savvy and may have performed well in courses that teach word processing, spreadsheet and internet skills. These students, therefore, have the misconception that because they are proficient in the use of computers they will do well as CS majors. What they do not realize is that CS is not only about using technology but also about developing technology for others to use. Ruslanov and Yolevich (2010) found that the vast majority of college students surveyed did not know what CS majors learn. Therefore, low retention rates in CS courses may be attributed in part to misperceptions about the major or the field in general (Biggers et al., 2008).

Like other universities nationwide, ETSU has attempted to address this issue in a freshman student-in-university course designed specifically for CS majors. The course introduces students to the different fields and job opportunities in CS, the courses required for each field, and how certain academic strengths and weaknesses may affect the students' success in the major courses. For some students, this type of course does help to clarify misconceptions about the field of CS. However, the course at ETSU is only available to incoming freshman and therefore is not beneficial to transfer students or students who did not start out at the university with CS declared as their major (T. Franklin, ETSU CS department advisor, personal communication, June 1, 2012).

Under-prepared for an Introductory Programming Course

A lack of problem solving and mathematical abilities (Beaubouef & Mason, 2005; Moskal et al., 2004) is a prevalent problem among students in introductory CS courses. Lack of pre-college preparation in these areas sets students up for failure in introductory programming courses. An introductory programming course or sequence of courses typically has a very complex set of topics that are being introduced (Yadin, 2011; Chen & Morris, 2005). These topics must be covered in the CS1 course in order to prepare students for the next programming course, CS2, which builds off of CS1. In CS1, students are expected to master problem solving, algorithm development, basic programming constructs, the syntax of a computer language, and a programming environment. This complexity and the number of topics can be overwhelming to students who have not been exposed to courses with this level of complexity at this point in their college career (Urness & Manley, 2011). Additionally, because of the large number of topics in these introductory programming courses, the pace of the CS1 course is very fast. Students with weak backgrounds in math and problem solving are often unable to keep up with the pace of learning in CS1 (Beaubouef & Mason, 2005; Moskal et al., 2004).

If a fair number of students in the course are under-prepared, it can cause the class population to be bimodal. A bimodal class is one in which students fall at opposite extremes of the spectrum ranging from those who struggle due to being under-prepared to those who are bored because they have prior programming experience (Hughes & Peiris, 2006). While students who are not prepared for a programming course are not likely to persist, students who are well prepared are at risk of becoming bored and are more likely to leave the CS major (Sloan & Troy, 2008).

A pedagogical approach that CS educators are exploring is adding a third course to the introductory programming course sequence. This third course, commonly known as CS0, is a prerequisite for the traditional CS1 course. The focus of this course is typically algorithm development and basic programming concepts (Pearce & Nakazawa; 2008; Sloan & Troy, 2008; Anewalt, 2007; Dierbach, Taylor, Zhou, & Zimand, 2005; Pearce & Nakazawa, 2008; Cliburn, 2006; Mitchell, 2001). The use of CS0 courses levels the playing field in the CS1 course by helping students who are weak in problem solving skills to become better prepared for CS1 thus reducing the bimodal problem occurring in CS1 courses. It also allows many of the basic topics from CS1 to be introduced in CS0, thus reducing the number of new concepts taught in the CS1 course. The inclusion of a CS0 course has had positive success in many colleges and universities in increasing the success of computer science students in CS1 (Stamey & Sheel, 2010; Sloan & Troy, 2008; Browne, Lowe, Wells, & Berry, 2006; Anewalt, 2007; Dierbach et al., 2005; Pearce & Nakazawa, 2008; Cliburn, 2006; Mitchell, 2001; Moskal et al., 2004).

Visual technologies are commonly used in CS0 courses. Visual programming environments and visual algorithm development tools have been successfully incorporated in CS0 (Moskal et al., 2004; Mullins, Whitfield & Conlon, 2009; Garlick & Cankaya, 2010). These environments and tools are designed to be more accommodating to visual learners.

Visual programming environments simplify complex programming concepts by allowing students to see visual representations of these concepts and to develop programs in a more visual manner. A popular example of a visual programming environment that has been successfully used in CS0 courses is the Alice programming environment. Alice, which was developed by Carnegie Mellon University, is an interactive environment utilizing drag-and-drop technology.

Objects and classes, fundamental concepts in OO programming introduced in CS1, are abstract and difficult for students in introductory programming courses to understand. Alice uses physical entities such as people, animals or cars to represent these concepts. A set of pre-made classes are provided in Alice for use in building objects. An example of classes provided in Alice is illustrated in Figure 1. Every class is a 3-dimensional character with a unique name.

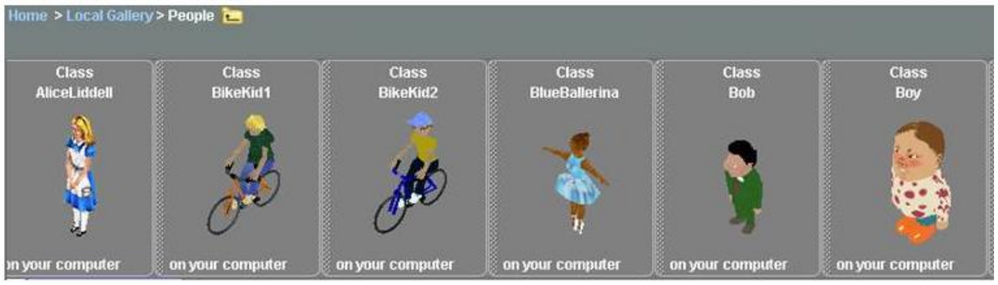


Figure 1. Example classes provided by Alice.

These objects are created and used in virtual worlds within Alice. In the Alice environment, students first select a template to begin creating their virtual world. As seen in Figure 2, examples include water, sand, grass, dirt, snow and space.

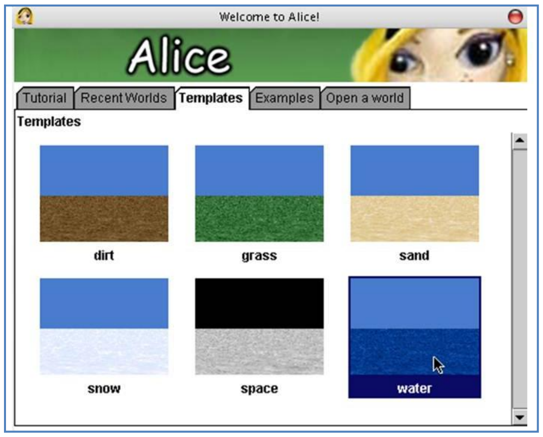
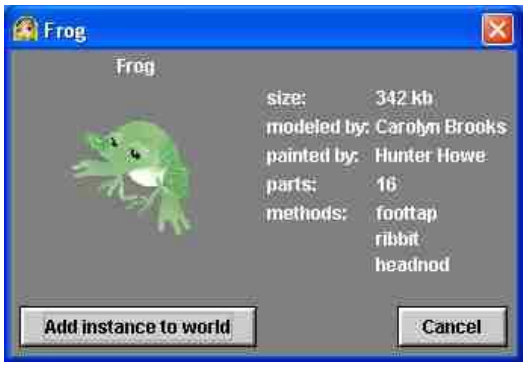


Figure 2. Alice templates for creating a virtual world.

Then students begin to put objects into their virtual world by selecting a class and creating an object (instance) of that class in the world as shown in Figure 3.



size:	342 kb
modeled by:	Carolyn Brooks
painted by:	Hunter Howe
parts:	16
methods:	foottap ribbit headnod

Figure 3. Adding an object to the virtual world in Alice.

As seen in Figure 4, the object is placed in the virtual world. Figure 4 also illustrates that every object is provided with a set of primitive methods that allow students to give action to the objects created in their world. Examples of these methods include moving, turning, rolling and resizing as seen in the bottom right of Figure 4. Certain objects also have custom methods for added functionality that is unique to the type of object being created. For example, the frog object has the custom methods of foottap, ribbit and headnod as seen in Figures 3 and 4.

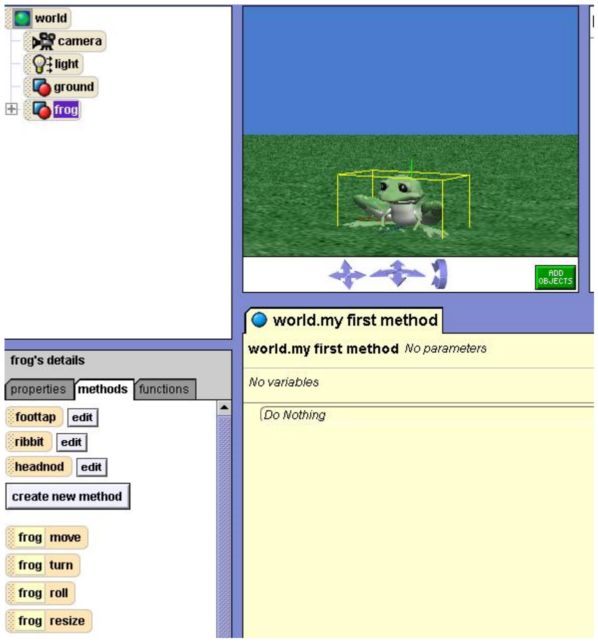
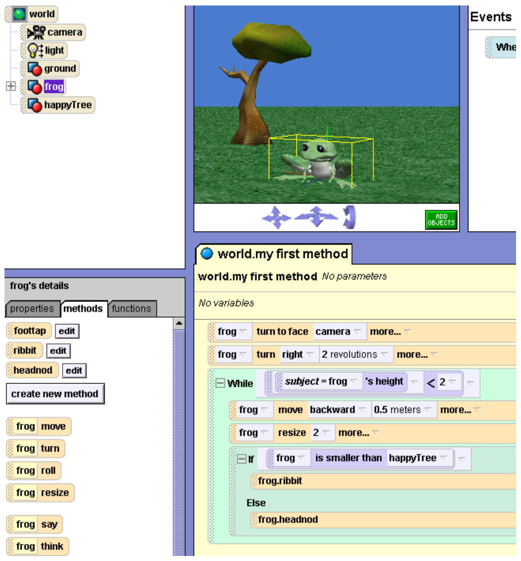


Figure 4. An object and its methods in the virtual world in Alice.

Along with all the predefined methods, Alice provides all the fundamental programming constructs of sequence, selection, and iteration (count-controlled and event-controlled repetition). Students can use these programming constructs and the provided methods to create animations and games. Figure 5 shows an animation with the frog and happyTree objects which includes sequence, selection and event-controlled iteration.



```
world.my first method No parameters
No variables
frog turn to face camera more...
frog turn right 2 revolutions more...
While subject = frog 's height < 2
    frog move backward 0.5 meters more...
    frog resize 2 more...
    If frog is smaller than happyTree
        frog.ribbit
    Else
        frog.headnod
```

Figure 5. An Alice animation with code.

One reason that Alice has been proven successful in CS0 courses is that students can see objects performing tasks on the screen in their virtual world as a 3D animation. In CS1, the OO programming concepts of objects and classes are difficult hurdles for students because they are so abstract and text-based only. Working with these tangible objects in CS0 helps to prepare students for these abstract concepts when encountered in CS1 (Gaddis, 2011a; Brown, 2008; Mullins, Whitfield & Conlon, 2008).

Another reason for using Alice in CS0 courses is that Alice eliminates the need for students to learn the syntax of a programming language (Gaddis, 2011a). The Alice environment allows students to use point-and-click with drag-and-drop technology to put objects in the world as seen in Figures 1 and 3 and to select methods as seen in Figure 4. The student *writes the program* by selecting a method on the left and dragging it into the method editor where the actions of the animation or game are created. Syntax errors, which are errors that violate the rules of a programming language, can be a major hurdle for beginning programmers. The fact that students can only point-and-click with drag-and-drop technology completely eliminates these errors.

Another benefit of using Alice in CS0 is that the visual nature of its 3-dimensional environment allows students to create animations and games, thus igniting their imagination and increasing their motivation and effort (Gaddis, 2011a, Brown, 2008). As a result, instructors can potentially see an increase in student interest, understanding and retention (Anewalt, 2007; Brown, 2008). However, students can be distracted by spending too much time on creating elaborate animations without understanding the intended programming concepts.

A benefit for instructors teaching CS0 with Alice is the support provided by Carnegie Mellon University. Carnegie Mellon provides the Alice software for free, provides training workshops for college and university faculty nationwide, and maintains a website that provides resources for teaching Alice and an online community of Alice users (http://www.alice.org). Another advantage for instructors is the wide variety of textbooks designed for CS0 courses that utilize Alice as the visual programming environment (Herbert, 2011; Shelly, Cashman & Herbert, 2007; Dann, Cooper & Pausch, 2011; Adams, 2007; Lewis & DePasquale, 2008).

Visual algorithm development tools, the other category of visual technologies used in CS0, support the development of algorithms in a more visual manner. Ideally, a program is designed using an algorithm before it is written in a programming language. The algorithm acts as the blueprint for the program and is where much of the problem solving in the programming process occurs. Students struggle with algorithm development because it is traditionally a paper-and-pencil activity. An example of a visual algorithm development tool is RAPTOR. RAPTOR was developed by the Department of Computer Science at the U.S. Air Force Academy. In the novice mode, RAPTOR's environment visually supports the algorithm development process.

The RAPTOR environment is based on flowcharts. Flowcharting is a technique for developing algorithms. It is made up of a set of symbols that represent the different activities within an algorithm including input, output, selection, iteration and assignment. It should be noted that RAPTOR only supports post-condition event-controlled iteration (i.e., a do…until loop).

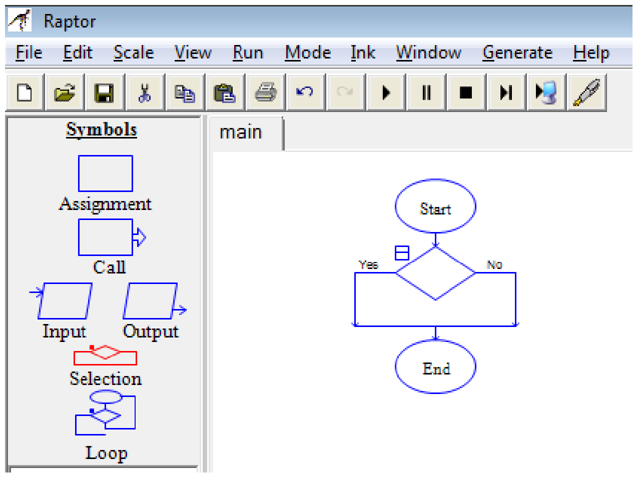


Figure 6. Flowcharting symbols available in RAPTOR.

Figure 6 illustrates the symbols that are available in RAPTOR's drag-and-drop environment for these basic algorithmic activities which are explained in Figure 7.

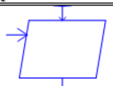
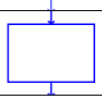
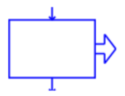
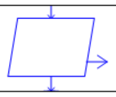
Purpose	Symbol	Name	Description
INPUT		input statement	Allow the user to enter data. Each data value is stored in a ***variable***.
PROCESSING		assignment statement	Change the value of a ***variable*** using some type of mathematical calculation.
PROCESSING		procedure call	Execute a group of instructions defined in the named procedure. In some cases some of the procedure arguments (i.e., ***variables***) will be changed by the procedure's instructions.
OUTPUT		output statement	Display (or save to a file) the value of a ***variable***.

Figure 7. Flowcharting symbols with descriptions.
Note. From *Introduction to programming with RAPTOR, June, 2012,* by W. Brown.

The most unique feature of the RAPTOR environment and its best benefit is the ability to execute an algorithm. Traditionally, the correctness of algorithms is checked through a manual activity called desk checking. RAPTOR automates the desk checking process by providing dialog boxes for input and output operations and by highlighting each executed step of an algorithm as it is encountered. Figure 8 shows the green highlighting provided by this step-by-step execution. RAPTOR also shows the content of variables in memory during this automated desk checking.

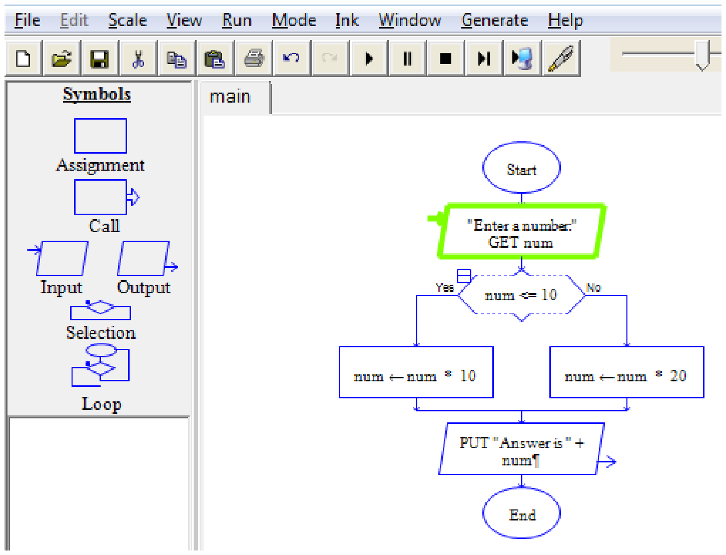
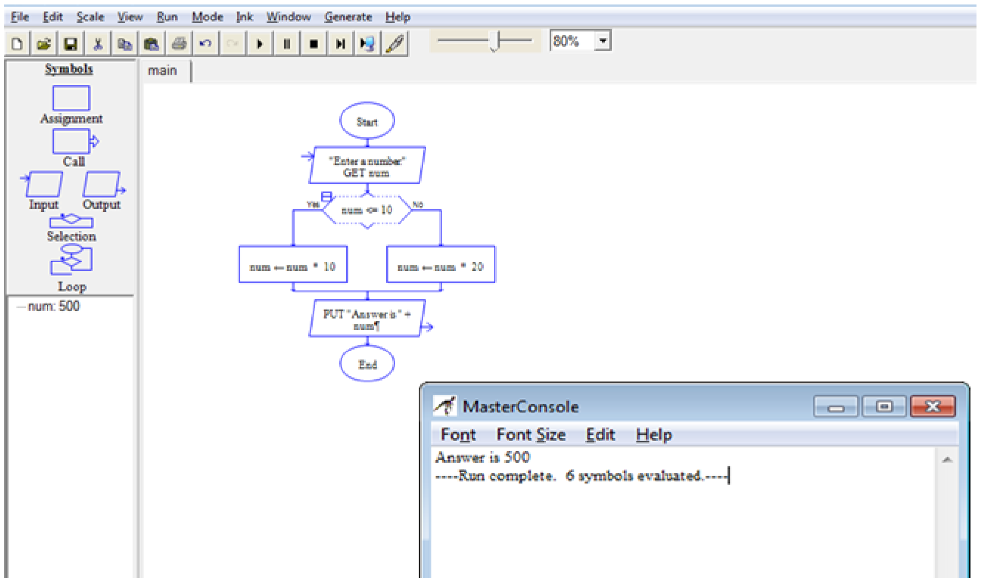


Figure 8. Step-by-step execution of an algorithm in RAPTOR.

The output for an algorithm is displayed in RAPTOR's master console window. Figure 9 shows an example of an algorithm and the output as a result with an input of 25.



```
Answer is 500
----Run complete.  6 symbols evaluated.----
```

Figure 9. Algorithm execution with output in RAPTOR.

The RAPTOR software is provided for free on the website (http://raptor.martincarlisle.com/). The website also provides resources for teaching RAPTOR including handouts and an online forum. However, there are a limited number of textbooks designed for CS0 courses that utilize RAPTOR as the visual algorithm development environment (Venit & Drake, 2011). More recent versions of RAPTOR include OO design elements from UML.

Switching Programming Paradigms and Languages

Another trend in CS education was the switch from a procedural paradigm to an object-oriented paradigm. The use of the object-oriented paradigm increased from 36% in 1995 to 82% in 2002 (Manaris, 2007). While the procedural paradigm is a relatively simple approach to algorithm development and program design, the object-oriented paradigm is a more complex abstract approach. This switch, driven by advancements in software development, has resulted in the inclusion of object-oriented related topics in the introductory programming courses. This addition has resulted in more topics and more complexity in the CS1 course (Manaris, 2007; Yadin, 2011; Ali, 2009; Sigle, 2008).

As a result of switching to the OO programming paradigm, the use of industry-strength programming languages became commonplace in introductory programming courses (Beaubouef & Mason, 2005; Manaris, 2007; Carlisle, 2009). Prior to 2001, more elementary programming languages such as Pascal, which was specifically designed for teaching introductory programming concepts, were used in CS1 and CS2. After 2001, the trend became to use more complex, industry-strength programming languages such as Java and C++ (Yadin, 2011; Blake, 2011; Soe et al., 2011; Sigle, 2008; Ali, 2009).

According to McCauley and Manaris (2002), by 2002 the use of industry-strength programming languages in the introductory programming courses had increased to 89% from 39% in 1996. In an attempt to give students experience in languages that are used in industry to better prepare them for careers outside of the university and to give them depth of exposure in a programming language as required by accrediting bodies, many departments added to the complexity in their introductory programming courses potentially having a negative impact on retention (Moskal et al., 2004).

In order to address this increased complexity, there are two schools of thought on how to introduce OO topics in CS1 courses. The dilemma that instructors are faced with is whether to adopt an objects-early or objects-late approach. While neither approach seems to be the obvious choice, some CS1 instructors prefer the objects-late approach because it gives the students the opportunity to focus on problem solving and foundational programming constructs such as data types, input and output, decisions and repetition (Yadin, 2011; Nesbit, 2009; Pillay & Jugoo, 2005; Beaubouef & Mason, 2005). Instructors who favor this approach believe that by waiting to introduce the OO concepts later in the semester it gives the students time to develop some programming maturity without overwhelming them with complex OO concepts at the beginning of the semester (Yadin, 2011; Mannila, Peltomäki & Salakoski, 2006; Eckerdal, Thuné & Berglund, 2005; Bruce, Buckingham, Hynd, McMahon, Roggenkamp & Stoodly, 2004). Another concern among those who prefer the objects-late approach is that the objects-early approach tends to focus more on the OO techniques and less on the basic problem solving and programming constructs such as algorithm development, selection, and repetition (Beaubouef & Mason, 2005).

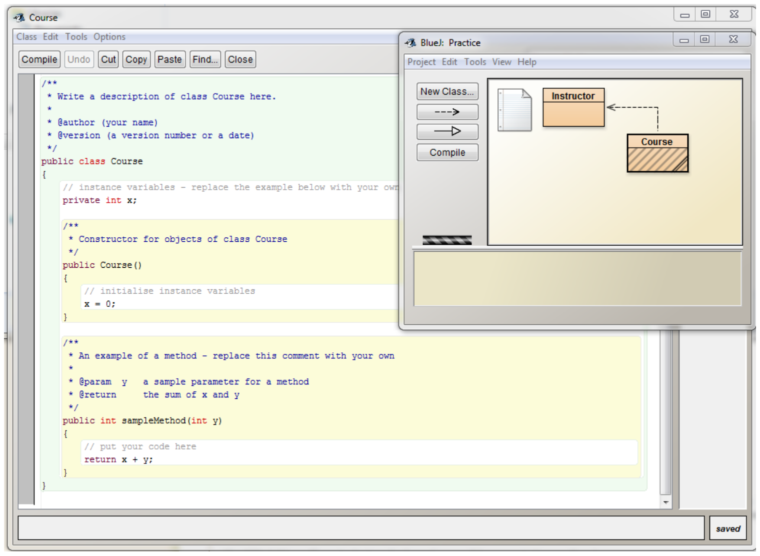
Another approach to addressing this increased complexity in CS1 is the inclusion of a CS0 course in the introductory programming sequence. As discussed previously, CS0 can be beneficial in introducing algorithm development, basic programming concepts and introductory object-oriented topics (Pearce & Nakazawa; 2008; Sloan & Troy, 2008; Anewalt, 2007; Dierbach et al., 2005; Cliburn, 2006; Mitchell, 2001). In CS0, students can be introduced to these topics in a setting and time-frame that is less pressured. This reduces the number of new concepts taught in the CS1 course, thus potentially reducing the complexity of the course and increasing the students' chance for success and retention.

A simpler integrated development environment (IDE) is also used to address the complexity introduced by switching to the OO paradigm. An IDE is software that allows the programmer to write, debug, compile and execute programs from within the same environment. Some IDEs can be quite complex and have a high learning curve, but a simpler IDE intended for use in teaching OO programming can reduce the learning curve for students who are unfamiliar with IDEs.

The BlueJ programming environment is an example of an IDE designed for teaching Java in CS1 (Gross & Powers, 2005; Pears et al., 2007). BlueJ, a visual programming environment, simplifies the programming process by removing some of the complexity of the development environment and providing graphical representations of the classes and objects within a project (Kouznetsova, 2007).

The BlueJ environment provides two windows in which the students work to create programs. One window contains a graphical class structure where students create the classes and objects needed for their program. Figure 10 shows the Instructor and

Course classes and their relationship. The Java code for each class can be displayed in an editor window by double-clicking on that class' icon. When a student creates a class in the class structure window, the BlueJ environment automatically generates the Java code for that class including a class attribute, a constructor method and a generic class method. This feature is beneficial to CS1 students because it reduces the amount of code to be written by the student and thus reduces syntax errors. Figure 10 shows the Java code created when the Course class is added to the class structure for this program.



```
/**
 * Write a description of class Course here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Course
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class Course
     */
    public Course()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

Figure 10. BlueJ environment.

The BlueJ environment also gives a visual indicator of which classes have been compiled. In Figure 10, the Instructor class has been compiled, but the Course class has not as indicated by the diagonal lines in the class icon for Course. In the Java editor within the BlueJ environment, students can modify the generated code and add to it to complete their programs.

The BlueJ software is provided for free on the website (http://www.bluej.org/). The website also provides resources for teaching BlueJ including tutorials and an online forum. There are several textbooks designed for CS1 courses that utilize BlueJ as the visual programming environment (Barnes & Kölling , 2012; Riley, 2003; Bhuta, 2007). Another advantage of BlueJ is its strong support for documentation. The BlueJ environment provides the ability to create and view Javadoc documentation.

Another IDE designed for teaching an OO language in CS1 is Greenfoot. Greenfoot, which is also a visual programming environment, combines text-based Java with animation. The Greenfoot environment provides 2D animation for the creation of games and simulations.

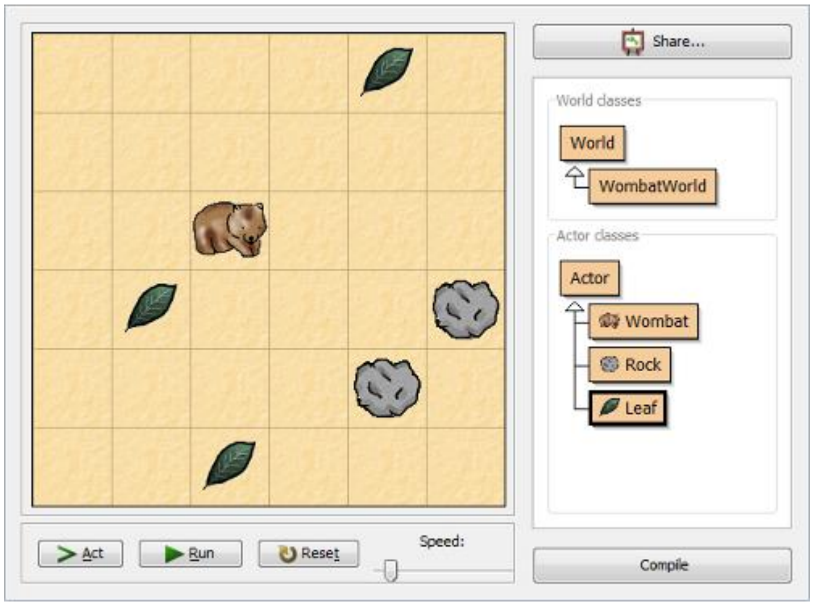


Figure 11. Greenfoot environment.

The Greenfoot environment allows the students to create animations in which they can add objects from a collection of pre-defined classes. Examples of these classes include Wombat, Rock and Leaf as seen in Figure 11. These classes have pre-defined methods associated with them; for example, the Animal class provides methods such as act, canSee, eat, move and turn. Figure 11 also shows the 2-dimensional grid in which all animations are created. Using the Java editor within the Greenfoot environment, students complete their programs by modifying the provided code or creating additional code.

The Greenfoot software is provided for free on the website (http://www.greenfoot.org). The website also provides extensive resources for teaching Greenfoot including video-based tutorials, example animations, and an online discussion group. Currently there is only one textbook designed for CS1 courses that utilizes Greenfoot as the visual programming environment (Kölling, 2010).

Learning Styles of Today's Students

The learning style of today's students has drastically changed from CS students of the past. The current generation, often referred to as Millenials (Stamey & Sheel, 2010; Oblinger, 2003; Frand, 2000), tend to be visual learners who prefer technology to textbooks, view active participation as more important than obtaining knowledge and therefore expect to learn in a manner in which they are immersed in animation and graphics through games and other recreational activities (Howles, 2007; Sigle, 2008). This is not an approach typically used in introductory programming courses due to the highly textual nature of many programming languages and the emphasis on the use of pseudocode, a solution written in an English outline format in algorithm development (Carlisle et al., 2004; Cardellini, 2002).

Many studies performed in CS programs focusing on student learning styles and their impact on student learning and success in introductory CS and similar engineering courses have reported that an overwhelming number of the students enrolled in these courses are visual learners (Chen & Lin, 2011; Gomes & Mendes, 2010; Gomes & Mendes, 2008; Gomes & Mendes, 2007; Gomes, Carmo, Bigotte, & Mendes, 2006; Kuri & Truzzi, 2002). It has been estimated that as many as 75% to 83% of modern CS majors are visual learners (Thomas, Ratcliffe, Woodbury, & Jarman, 2002). Students' learning styles impact the way that they approach learning new concepts and apply new skills (Chamillard & Sward, 2005). Because visual learners retain more from things that they see, flowcharts, diagrams, and pictures are more beneficial to them than written and spoken explanations (Chamillard & Karolick, 1999). By understanding the learning styles of students and how those learning styles affect their comprehension and mastery of the subject matter, instructors of introductory level CS courses can better equip themselves to present the basic problem solving techniques and programming constructs in a manner which will accommodate all learning styles, including visual learners (Chamillard & Sward, 2005; Grant, 2003; Burgess & Hanshaw, 2006).

Several approaches to helping visual learners have been described in previous sections. Visual programming environments and visual algorithm development tools such as Alice, RAPTOR, BlueJ and Greenfoot have been developed to bring a more interactive, simplistic, and creative approach to teaching introductory programming courses. Some of these are more appropriate for CS0 and some for CS1, but all attempt to replace the highly textual nature of programming with visual representations and animations.

Learning to program can be an overwhelming task for students regardless of their learning style. The number and complexity of topics, the use of languages not designed for teaching, and students lacking basic problem solving skills can combine to make learning programming in an introductory programming course an overwhelming task. As a result, CS instructors are desperate to find ways to overcome these obstacles. Herbert (2011) proposes that to make learning programming easier, one must minimize the syntax of the programming language being used and provide visual feedback to the students to aid in conceptualization of the programming constructs. In addition, Adams (2007) suggests that instructors must be able to capture the students attention through intriguing examples that will motivate today's CS students to learn. As a result, visual technologies are being explored as a way to present difficult concepts in a manner that is easier to visualize and simpler to use.

Poorly Designed Introductory Programming Courses

Another factor affecting student retention is the fact that many introductory programming courses are simply poorly designed (Beaubouef & Mason, 2005; Carlisle, 2009). It can be extremely difficult to design a course or sequence of courses that can successfully teach today's students to program. Ironically, while college level instructors of CS courses typically have the background knowledge required of a practitioner or researcher in the CS field, they often lack exposure to the educational background that is often necessary to convey knowledge correctly, reliably and effectively (Gal-Ezer & Harel, 1998). Given that CS is a constantly evolving field, the introductory programming courses are quite complex in nature and CS education has not historically been accommodating to visual learners. With lower enrollments in CS, retaining the students that are already in the major is crucial. Because the most crucial point for retention is in the introductory CS courses (Moskal et al., 2004; Forte & Guzdial, 2004; Chen & Morris, 2005; Herrmann et al., 2003; Talton et al., 2006; Boyer et al., 2007), CS educators must design and deliver courses that will enhance student success and retention.

Relationship of the Literature to the Study

Considering the factors identified as affecting both student success and retention in introductory programming courses and the solutions that had been proposed to address these factors, this study focused on one category of solutions in particular, the use of visual technologies. Each of the visual technologies presented in the literature review was examined in relation to its use and effectiveness as it relates to student success and retention in the introductory programming courses. A rubric was developed in order to rate the visual technologies in each category (visual programming environments and visual algorithm development tools) that had been reported in the literature as beneficial to student learning and success in introductory programming courses. The rubric was then used to rate the visual technologies according to criteria such as academic acceptance, availability of supporting textbooks, availability of student/academic versions, financial costs, system requirements, usability, OO support, and appropriateness for teaching fundamental programming concepts.

Ultimately, the goal of the study was to develop and implement curricula that incorporated visual technologies in the introductory CS programming courses in order to improve student retention and achievement. Based upon the findings presented in the literature review and the rating of the visual technologies as appropriate for use in CS1, the CS1 course at ETSU was redeveloped to incorporate multiple visual technologies in an attempt to effectively increase student learning in CS1 therefore potentially increasing student performance and retention.

Chapter 3

Methodology

Problem and Goal

Decreasing enrollments, lower rates of student retention and changes in the learning styles of today's students are all issues that the CS academic community is currently facing. As a result, CS educators are being challenged to find the right blend of technology and pedagogy for their curriculum in order to help students persist through the major and produce strong graduates. Learning to program can be an overwhelming task for introductory programming students for many reasons. The number and complexity of topics, the use of languages not designed for teaching, and students lacking basic problem solving skills can combine to make learning programming in an introductory programming course (i.e., CS1) an overwhelming task. Visual technologies have been explored as a way to present difficult concepts in a manner that is easier to visualize and simpler to use. Visual technologies make learning programming easier by minimizing the syntax of the programming language being used and providing visual feedback to the students to aid in conceptualization of the programming constructs. The goal was to improve student retention and performance by incorporating visual technologies in the CS1 programming course at ETSU.

Research Design

Instructional design and quasi-experimental research methods were used to develop and implement a curriculum that incorporates visual technologies in the introductory CS programming course, CS1. The CS1 course was redesigned and redeveloped to incorporate visual technologies using the ADDIE approach. The ADDIE approach is a generic approach to instructional design which involves a five-phase process: Analysis, Design, Development, Implementation and Evaluation (Molenda, 2003; http://www.instructional.org/models/addie.html). Each phase of the ADDIE approach is described in detail in the approach section pertaining to the redesign of the CS1 course.

A quasi-experimental study, using the Post-Test Only Nonequivalent Groups Design approach, was implemented to perform assessment during the evaluation phase of the ADDIE approach to the redesign of the CS1 course at ETSU. Quasi-experimental designs are well suited for evaluation of educational programs when random assignment of control and treatment groups is not feasible (Gribbons & Herman, 1997; Trochim & Donnelly, 2008; Gay & Airasian, 2003). This type of educational research is frequently used when entire classrooms or entire sections of a course are being evaluated (Gribbons & Herman, 1997; Trochim & Donnelly, 2008; Gay & Airasian, 2003). This approach was, therefore, the most appropriate for the proposed study.

In experimental research, the researcher manipulates the independent variable(s) and observes the effect on the dependent variable(s) (Gay & Airasian, 2003). The independent variable was the curriculum used in the CS1 course at ETSU. The dependent variables were the effects that the implemented curriculum had on student performance in CS1 and retention as measured by persistence to CS2.

The effects of the curriculum were observed through the comparison of two groups. The control group was the collection of students who took the original version of the CS1 course (instruction without the use of visual technologies) at ETSU during the first semester of the study while the treatment group was the collection of students enrolled in the CS1 course during the second semester of the study in which the revised curriculum (instruction with the use of visual technologies) was implemented.

Analysis of the data collected from both the control group and the treatment group has provided conclusions on the effect of the newly designed curriculum with visual technologies on student performance and retention in the introductory programming curriculum. There are many different kinds of observational data measures that have been used to evaluate the impact of visualization in novice programming environments. Some of the most common data measures include grades on coursework throughout the semester, final grades for the course, and course retention rates (Gross & Powers, 2005; Moskal et al., 2004; Naps et al., 2003). Therefore, these measures are what were collected and analyzed in this research in regard to determining the relationship between the independent and dependent variables among the control and treatment groups.

Instrumentation

Instruments were developed to assess student learning throughout the CS1 classes. One set of instruments were the checkpoints that were given after a topic had been presented to the students and after the students had completed a programming assignment on that topic. The checkpoints were administered through an online course delivery system. The course delivery system provided statistics on the overall student performance on each checkpoint as well as student performance on each question of the checkpoints. All checkpoints were verified and approved by the CSCI Introductory Programming Committee at ETSU. This process involved the CSCI Introductory Programming Committee evaluating the checkpoint and providing feedback on recommended changes. The checkpoint was then modified based on the feedback and was reevaluated by the committee. The checkpoints can be found in Appendix A.

In addition to the checkpoints, a short questionnaire was administered at the beginning of the semester to collect information about the students' majors. It was also used to determine if any students were repeating the course or if they were under 18 so their data could be excluded from the data collection set. The questionnaire is included in Appendix A.

Experts

Throughout the redesign process and upon completion of the course redesign, the new curriculum and data collection instruments were presented to the CSCI Introductory Programming Committee in the Department of Computing at ETSU for review. This standing departmental committee is responsible for any decisions regarding the introductory programming courses within the department. Members of this committee include four tenured Full Professors each with a Ph.D. in CS, two tenured Assistant Professors each pursuing a Ph.D. in CS, one tenure-track Assistant Professor with a Ph.D. in CS and one Lecturer with a Ph.D. in IS. All members of the committee are actively involved in teaching one or more of the courses in the introductory programming sequence. Their detailed information is provided in Table 1.

Table 1

Experts/Members of the CSCI Introductory Programming Committee at ETSU

Name	Degree Held	Rank	# years teaching experience	Freshman level courses taught
Dr. Don Bailes	Ph.D. in CS	Full Professor	40	CS1, CS2
Dr. Gene Bailey	Ph.D. in CS	Full Professor	45	CS0, CS1, CS2
Dr. Marty Barrett	Ph.D. in CS	Full Professor	26	CS1, CS2
Dr. Suzanne Smith	Ph.D. in CS	Full Professor	27	CS0, CS1, CS2
Dr. Jessica Keup	Ph.D. in IS	Lecturer	6	CS0
Mr. Jeff Roach	Pursuing Ph.D. in CS	Asst. Professor	10	CS1, CS2
Dr. Jay Jarman	Ph.D. in CS	Asst. Professor	7	CS1, CS2
Mrs. Kellie Price	Pursuing Ph.D. in Computing Technology in Education	Asst. Professor	19	CS0, CS1, CS2

Approach

Research Question 1: What are the factors attributable to poor performance and low retention rates and what solutions have been reported?

The literature served to identify the major factors that are attributable to the poor performance and low retention rates in introductory CS programming courses. The literature review focused on the factors affecting student success and retention in introductory programming courses and the approaches being proposed to alleviate these factors such as the use of visual technologies. Two specific categories of visual technologies were researched, those being used to support algorithm development and those supporting program development.

Research Question 2: How can the introductory course, CS1, be redeveloped and implemented to incorporate visual technologies?

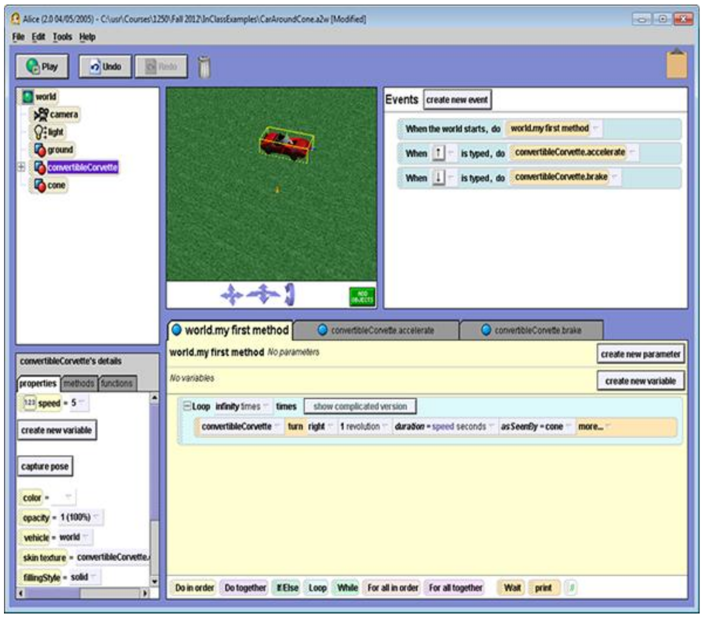
Using the ADDIE approach to instructional design, the CS1 course was redesigned and redeveloped to incorporate visual technologies. The following sections outline what took place during each phase of the course redesign.

Analysis

During the analysis phase, the learning styles of today's students, the visual technologies appropriate for use, and the instructional goals and objectives for the course were identified relative to the CS1 course at ETSU. The literature review focused on the learning styles of today's CS students as one of the factors attributable to poor performance and low retention rates due to teaching and delivery styles that do not relate to today's visual learners (Beaubouef & Mason, 2005; Stamey & Sheel, 2010; Oblinger, 2003; Frand, 2000; Howles, 2007; Sigle, 2008; Carlisle et al., 2004; Cardellini, 2002). Visual technologies, that have been developed to support learning programming at the introductory level, were also a focus of the literature review.

As a result of the literature review, a list of visual technologies used to support both algorithm and programming development that are most appropriate for CS1 was developed. There were several factors considered in selecting visual technologies to be used in CS1 at ETSU, including but not limited to the learning styles of CS students, goals and objectives for the CS1 course, cost, availability, and support for topics presented in CS1. The list of visual technologies and the factors to be considered were presented to the Introductory Programming Committee for evaluation and selection of one or more technologies to be incorporated into CS1 at ETSU. Ultimately, it was decided that two visual technologies, Alice and RAPTOR, would be appropriate for use in the CS1 course at ETSU.

There were several considerations that led the Introductory Programming Committee to select Alice for incorporation into CS1. In CS1, the OO programming concepts of objects and classes are difficult hurdles for students because these topics are abstract and text-based only. Alice has been proven successful in CS0 courses because it allows students to see objects performing tasks on the screen in their virtual world as a 3D animation as shown in Figure 12. Working with these tangible objects in an environment such as Alice helps to prepare students for these abstract concepts when encountered in CS1 (Gaddis, 2011a; Brown, 2008; Mullins, Whitfield & Conlon, 2008) as demonstrated in Figure 13.



```
Events
When the world starts, do world.my first method
When ↑ is typed, do convertibleCorvette.accelerate
When ↓ is typed, do convertibleCorvette.brake

world.my first method No parameters
No variables
Loop infinity times times
    convertibleCorvette turn right 1 revolution duration = speed seconds asSeenBy = cone more...
```

world.my first method
convertibleCorvette.accelerate
convertibleCorvette.brake
convertibleCorvette's details
speed = 5

A program using the convertibleCorvette class in Alice, with a class attribute speed and two class methods accelerate and brake.

The visual representation below of executing the program that allows the user to accelerate and brake while the car is moving.

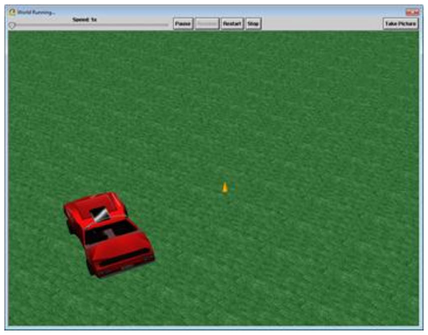
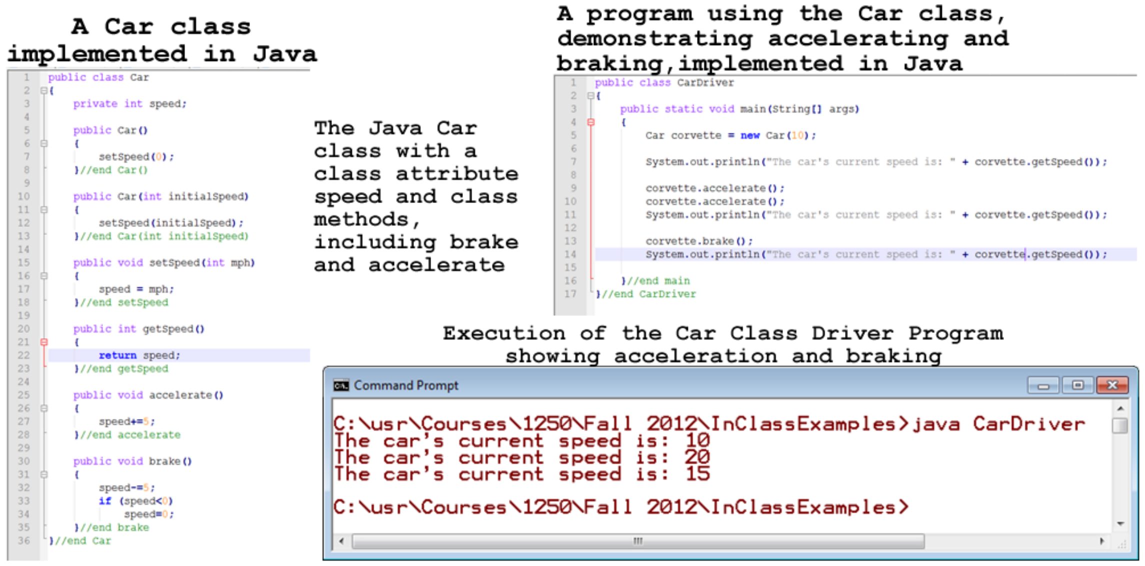


Figure 12. An Alice application that allows acceleration and braking using a Car class.



A Car class implemented in Java
The Java Car class with a class attribute speed and class methods, including brake and accelerate
A program using the Car class, demonstrating accelerating and braking,implemented in Java
Execution of the Car Class Driver Program showing acceleration and braking
Command Prompt

```
public class Car
{
    private int speed;

    public Car()
    {
        setSpeed(0);
    }//end Car()

    public Car(int initialSpeed)
    {
        setSpeed(initialSpeed);
    }//end Car(int initialSpeed)

    public void setSpeed(int mph)
    {
        speed = mph;
    }//end setSpeed

    public int getSpeed()
    {
        return speed;
    }//end getSpeed

    public void accelerate()
    {
        speed+=5;
    }//end accelerate

    public void brake()
    {
        speed-=5;
        if (speed<0)
            speed=0;
    }//end brake
}//end Car
```

```
public class CarDriver
{
    public static void main(String[] args)
    {
        Car corvette = new Car(10);

        System.out.println("The car's current speed is: " + corvette.getSpeed());

        corvette.accelerate();
        corvette.accelerate();
        System.out.println("The car's current speed is: " + corvette.getSpeed());

        corvette.brake();
        System.out.println("The car's current speed is: " + corvette.getSpeed());

    }//end main
}//end CarDriver
```

```
C:\usr\Courses\1250\Fall 2012\InClassExamples>java CarDriver
The car's current speed is: 10
The car's current speed is: 20
The car's current speed is: 15

C:\usr\Courses\1250\Fall 2012\InClassExamples>
```

Figure 13. A Java implementation of an application using a similar Car class that allows for acceleration and braking.

Another benefit of using Alice is that the visual nature of its 3-dimensional environment allows students to create animations and games, thus igniting their imagination and increasing their motivation and effort (Gaddis, 2011a, Brown, 2008) and potentially causing an increase in student interest, understanding and retention (Anewalt, 2007; Brown, 2008). Another consideration for the use of Alice is the support provided by Carnegie Mellon University. Carnegie Mellon provides the Alice software for free, provides training workshops for college and university faculty nationwide, and maintains a website that provides resources for teaching Alice and supports an online community of Alice users (http://www.alice.org). Another consideration was that several instructors on the Introductory Programming Committee were already familiar with Alice having attended Alice workshops and having used Alice in the CS0 course at ETSU.

The selection of RAPTOR for CS1 at ETSU was based upon several considerations as well. Ideally, a program is designed using an algorithm, acting as the blueprint for the program, before it is written in a programming language. Developing the algorithm is where much of the problem solving in the programming process occurs. Many students tend to struggle with algorithm development because it is traditionally a paper-and-pencil activity and their algorithms tend to lack the detail needed to adequately solve the problem at a level that can easily be translated into a computer program. Figure 14 illustrates the different levels of detail that are often included in student algorithms. Algorithms A and B lack sufficient detail for a complete solution while Algorithm C has enough detail to be thoroughly tested and translated directly into code. If a student writes an algorithm that looks like Algorithm A or Algorithm B, it will be hard for the student to adequately test the algorithm through desk checking (i.e., manually traversing through the steps of the solution with real data as input and observing whether or not the resulting actions or output was correct) because it lacks sufficient detail. This lack of detail would also cause the algorithm to be too difficult to translate directly into code. Even if the student writes an algorithm with sufficient detail, such as Algorithm C, it will be time consuming to desk check the algorithm by stepping through the steps one at a time and performing the math, ultimately making it less likely that the student will check it at all and therefore simply assume that the algorithm will work. As has been observed in CS1 at ETSU over the years, many students do not write an algorithm unless forced to and the result is generally a solution similar to Algorithm A.

Algorithm A

```
Get the hours worked
Get the pay rate
Calculate wages
Display the wages
```

Algorithm B

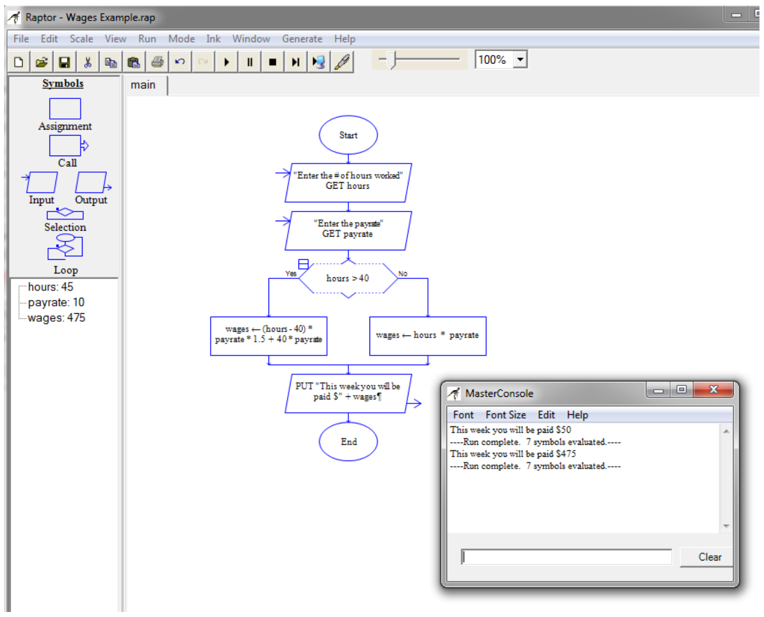
```
Ask the user for the hours worked
Ask the user for the pay rate
If hours worked > 40
  Calculate wages with overtime
Otherwise
  Calculate wages without overtime
Display the wages to the user
```

Algorithm C

```
Ask the user for the hours worked
Ask the user for the pay rate
If hours worked > 40
  wages = (hours-40)*pay rate*1.5 + 40*pay rate
Otherwise
  wages = hours * pay rate
Display the wages to the user
```

Figure 14. Examples of algorithms written in pseudocode.

In contrast, RAPTOR is an algorithm development tool whose environment visually supports the algorithm development process. RAPTOR, which represents algorithms using flowcharts, is beneficial to students because it has an easy-to-use, drag-and-drop environment and it represents the solution to a given programming problem in a visual manner as opposed to a text-based solution. Another benefit of RAPTOR was its ability to execute an algorithm. Traditionally, the correctness of algorithms is checked through a manual activity called desk checking. RAPTOR automates the desk checking process by providing dialog boxes for input and output operations and by highlighting each executed step of an algorithm as it is encountered. Figure 15 illustrates the same algorithm as shown in Figure 14, with sufficient detail to be able to execute the solution. When the Play button is clicked in RAPTOR, each step of the flowchart will be highlighted as it is reached. The user will be prompted for input and the output will be displayed when appropriate.



```
hours: 45
payrate: 10
wages: 475
```

```
Start
"Enter the # of hours worked"
GET hours
"Enter the payrate"
GET payrate
hours > 40
Yes: wages ← (hours - 40) * payrate * 1.5 + 40 * payrate
No: wages ← hours * payrate
PUT "This week you will be paid $" + wages¶
End
```

```
This week you will be paid $50
----Run complete.  7 symbols evaluated.----
This week you will be paid $475
----Run complete.  7 symbols evaluated.----
```

Figure 15. Example of an algorithm written as a flowchart in RAPTOR.

RAPTOR visually animates the solution, making the desk checking process much more interactive and easier to see where problems may occur, and therefore making it more likely that students will develop and test an algorithmic solution before writing code. Because RAPTOR has the ability to execute a solution, the solution must have enough detail in order to be successfully executed in RAPTOR. This accomplishes the goal of forcing the students to solve the problem at the level of detail needed, not only to translate it into code, but to also adequately test the solution before writing the program in a computer language. Finally, the RAPTOR software, user manuals, and resources for teaching RAPTOR are provided for free on the website (http://raptor.martincarlisle.com/).

The learning objectives of the CS1 course remained unchanged. These learning objectives/outcomes have been approved by the CSCI department and are tied directly to ABET learning outcomes. ABET is the accrediting agency for post-secondary education programs in computing and engineering, and the curriculum at ETSU is accredited in CS, IS and IT. While the course learning outcomes, which are listed in Appendix B, were not changed, the curriculum and instructional methods that were used to achieve these learning outcomes were modified. The use of visual technologies pertained to two of these learning outcomes which are to develop an object-oriented design and to program in Java, an object-oriented programming language.

Design

During the design phase, the selected visual technologies, Alice and RAPTOR, were incorporated into the course content and instructional delivery methods. A detailed course calendar was also created outlining the coverage of the topics for the course on a semester timeline. The course calendar can be found in Appendix B.

It was also during this phase that the assessment techniques to be used for the quasi-experimental part of this research project were designed. It was decided that student performance would be assessed throughout the semester at several key points: upon completion of each major topic and upon completion of the course. The major topics in the course selected for assessment were selection (decisions), repetition (looping), and objects and classes. Assessment was to be performed through the use of checkpoints (i.e., quizzes covering the respective topics). It was determined that evaluating student learning at certain points throughout the semester as major topics were presented would allow for the evaluation of student performance on a topic-by-topic basis as well as on the course as a whole. This type of evaluation was helpful in determining if the use of visual technologies was particularly beneficial in some aspects of the course more than others.

Development

During the development phase, detailed lecture notes and in-class exercises were developed to incorporate the use of RAPTOR to present the course material related to selection and repetition and the use of Alice to present the topics of objects and classes. A handout on problem solving and programming, included in Appendix C, was also developed; this handout was designed for the students in CS1 during the treatment semester. The handout explains the process of developing a solution to a problem using both pseudocode and flowcharts. It highlights the need for both testing the solution and including enough detail in a solution so that it is not only testable but ready to be translated into a programming language. This handout was designed as part of the students' introduction to RAPTOR.

Checkpoints for the selected topics were also developed during this phase. The checkpoints, which were developed by the researcher, were evaluated and approved by the CSCI Introductory Programming Committee. It was determined that all sections of the CS1 course would administer the same checkpoints throughout the semester.

A questionnaire was also developed to collect demographic information from the students enrolled in CS1 during the second semester of the study. Information collected included the students' majors, whether or not they had previously attempted the CS1 course at ETSU, and whether or not they were at least 18 years of age. Students who indicated they had previously attempted CS1 or were under 18 years of age were excluded from the study.

Implementation

The implementation phase was the phase during which the CS1 course was taught in its original form the first semester and in its revised form the following semester. Prior to the implementation of the revised CS1 course, all instructors of the course were trained on the new visual technologies being used throughout the course. The instructors were also provided with the newly developed course materials including lecture notes, in-class exercises, and checkpoints for use in the second semester of the study.

Early in the second semester of the study, prior to any checkpoints being administered, the treatment group was given the demographics questionnaire. The questionnaire collected the information needed to determine if they should be excluded from the study based on age and whether or not they were repeating the course. They were also presented with a description of the study to be conducted, and IRB consent forms were distributed. ETSU's IRB required the students' consent before they could be included in the study. The researcher was not aware of which students or how many students had elected to participate in the study until after the final grades had been recorded as required by ETSU's IRB procedures.

For introduction of selection and repetition topics, RAPTOR, the visual technology supporting algorithm development, was incorporated into both the lecture and hands-on activities in the labs. When introducing decisions using the if-else statement, during the first semester of the study whiteboards and overhead presentations were used to present students with a sample problem and then class discussion would follow to engage the students in developing a solution to the problem. This solution was written in pseudocode which is a textual solution to algorithm development (as shown in Figure

14). The pseudocode was then tested by manual desk checking. The pseudocode was used as the guide from which the actual Java solution to the problem was created. With this scenario, it has been observed that students at ETSU would skip the step of creating the algorithm (i.e., writing down the pseudocode solution) and would begin immediately writing the Java code. This approach usually resulted in a poorly designed and tested solution and a much longer coding process because the debugging phase in this scenario included debugging the solution to the problem as well as the code that was written to solve the problem. In contrast, when using RAPTOR, the students were presented with a problem, and class discussion would follow to engage the students in developing a solution to the problem. However, instead of writing the solution out in pseudocode on a whiteboard, the solution was implemented in RAPTOR using flowcharts. This gave the students a more visual picture of the solution. The biggest advantage, however, was that the solution could then be executed and students could actually give input, watch the flow of execution, and see the output. Alice, the visual technology supporting program development, was used to introduce the topics of objects and classes in both the lecture and the lab. During lectures, an Alice world was created, and the instructor would begin to place things in the world as a way of introducing the classes, such as a bunny, that are provided in Alice. An instance of the bunny class would then be added to the world (explaining that this was an object, an instance of the bunny class). Eventually adding more instances of the bunny class to the world would demonstrate that each one was its own instance of the bunny class and had its own characteristics, identity, and actions that could be performed. This allowed the students to see these abstract object-oriented concepts of objects and classes using real-world examples. The goal was to introduce these abstract concepts visually, instead of just through the Java programming language, therefore helping students to better understand the concepts. Students were then given problems to solve using Alice in the lab. The problems selected were similar to problems that they were soon to be given to solve in the Java programming language. By being given the same problem to be implemented both in Alice and in Java, students were able to make the connection between objects and classes they could see and manipulate in Alice and the objects and classes they would need to implement in Java code.

Evaluation

The revised instructional approach was evaluated. As described in the section pertaining to research question 3, analysis was performed to evaluate the effectiveness of using visual technologies in the CS1 course with respect to student performance and retention rates. The evaluation phase began at the close of the second semester of the study. Due to ETSU IRB restrictions, evaluation could not occur until final grades had been recorded at the end of the treatment semester.

Research Question 3: What are the outcomes of teaching the redesigned course?

During the evaluation phase of the ADDIE approach to the redevelopment and redesign of the CS1 course at ETSU, assessment was performed using the Post-Test Only Nonequivalent Groups Design approach. This quasi-experimental design was chosen because it is well suited for evaluation of educational programs when random assignment of control and treatment groups is not feasible (Gribbons & Herman, 1997; Trochim & Donnelly, 2008; Gay & Airasian, 2003).   Most often with this approach, intact groups that are as similar as possible are used as the treatment and control groups (Trochim & Donnelly, 2008).

The control group was the collection of students who took the original version of the CS1 course (instruction without the use of visual technologies) at ETSU during the first semester of the study. The treatment group was the collection of students enrolled in the CS1 course during the second semester of the study, in which the revised curriculum (instruction with the use of visual technologies) was implemented. During a typical semester, there are four to five sections of the CS1 course with a total of 65-120 students and two to three instructors. These two groups have been considered equivalent since the students had met the same admission standards for the university and the same prerequisite requirements for the course.

The control group consisted of a total of 93 students, 47 CS majors and 46 non-CS majors. Of the 93, 10 dropped the course, 52 passed and 31 failed. In addition, 7 of the 93 had completed CS0 prior to taking CS1 and 5 of those 7 successfully completed CS1. The treatment group consisted of 80 students who elected to participate in the IRB study, 62 CS majors and 18 non-CS majors. Of the 80, 3 dropped, 53 passed and 24 failed. In addition, 4 of the 80 had completed CS0 prior to taking CS1 and 2 of those 4 successfully completed CS1.

The courses, which are scheduled for two hours twice a week, were taught in a lecture/lab format. The first two-hour class meeting each week was done in a conventional lecture methodology in which the instructor covered topics regarding problem solving, algorithm development, the software development process, programming and the syntax of the programming language being used. During the second two-hour class meeting each week, the students were in a computer lab. Instruction typically took place during the first hour of the lab. This instruction generally consisted of additional lecturing on the topics presented that week or an exercise in which the instructor stepped through the solution to a given problem incorporating that week's lecture topics. The second hour was typically used for completion of hands-on, in-class exercises by the students that related directly to the lecture topics for that week.

Using the assessment instruments developed, the instructors collected data at key points throughout each semester as well as at the end of each semester to evaluate student performance. Student retention was measured by observing whether or not the students remained in CS1 (did not withdraw from the course prior to the end of the semester) and persisted in the program by enrolling in the CS2 course upon completion of the CS1 course with a passing grade of C- or better.

Research Question 4: What conclusions may be drawn regarding the value of the new curricula in terms of student performance and retention?

Analysis has been performed on the data collected from both the control group and the treatment group during both semesters. This analysis provides conclusions on the effect of the inclusion of visual technologies on student performance and retention in the introductory programming course. The results of the analysis are presented in Chapter 4 and conclusions on the effect of visual technologies on student performance and retention are presented in Chapter 5.

The analysis of student performance included the use of a *t*-test. A *t*-test for independent samples is appropriate for determining whether the observed difference between two independent groups is significant. It works by determining if the difference is significantly larger than the difference expected solely based on chance (Gay & Airasian, 2003). Therefore, the *t*-test for independent samples was used to determine the significance of difference between the control group and treatment group on the following measures: calculated final grade and checkpoints for each major course topic. A chi-squared test was used for comparison of persistence rates to CS2 between the control group and treatment group. In the comparison of retention rates in CS1, the chi-squared test was originally used, but yielded warnings due to the small sample sizes in the retention data. As a result, the Fisher's exact test was used instead. Because retention and persistence in the major do not apply to non-majors, these two tests were only applied to data collected pertaining to participants in each group who were CS majors.

Data Collection

Using the assessment instruments developed, the instructors collected data at key points throughout each semester as well as at the end of each semester to evaluate student performance. Data collected includes student scores on the checkpoints given upon completion of the coverage of each major topic as well as the final grade assigned to the student for the course.

Student retention was measured by observing whether or not the students remained in CS1 (did not withdraw from the course prior to the end of the semester). Student persistence in the major was measured by observing whether or not the majors enrolled in the CS1 course persisted in the program by enrolling in the CS2 course upon completion of the CS1 course with a grade of C- or better or by re-enrolling in the CS1 course if they did not make a satisfactory grade. This enrollment data was obtained from the student information system at ETSU upon completion of the second semester of the study.

Resources

Technologies

Part of the research was to determine the pedagogical approach and which visual technologies would be most effective when integrated into the introductory programming courses. The visual technologies chosen to be incorporated into the CS1 course at ETSU were Alice and RAPTOR, both of which were free to the students and the university. Both software packages were installed in the lecture rooms and computer labs used by CS1 students. Many of the CS1 students also installed RAPTOR and Alice on their own personal computers as well.

Other technologies used included D2L, Microsoft Excel and Minitab. Microsoft Excel was used to compile the data collected from the checkpoints and the student enrollment data. Minitab was used to perform the analysis on the data. D2L (Desire2Learn) is ETSU's online learning management system through which course materials and checkpoints were administered to the students.

People

Experts within the department were solicited to review and evaluate the visual technologies proposed, the manner of teaching and implementing these technologies in

CS1, and the assessments used. These experts are members of the CSCI Introductory Programming Committee, a standing committee in the Department of Computing at ETSU whose charge is to make decisions regarding the introductory programming courses within the department.

Permissions

IRB permissions were received from both Nova Southeastern University and East Tennessee State University. The IRB approvals are included in Appendix C. Students who elected to participate in the second semester of the study as part of the treatment group were required to sign an Informed Consent Document giving their permission to be included in the study. These documents were collected and held by a departmental colleague approved by the ETSU IRB and released to the researcher after final grades for the course had been recorded.

Summary

Incorporating visual technologies in teaching CS1 has helped to determine whether such technologies effectively increased student retention and performance. Based upon the level of effectiveness observed from teaching CS1 with visual technologies as compared to teaching CS1 without visual technologies, the redesigned course or portions of the redesigned course will be proposed for adoption by the entire Department of Computing at ETSU. These results will also be presented to the CS academic community through publications and conferences.

Ideally, both CS students and faculty will benefit from this practice. Students will benefit from an approach that satisfies their learning styles and their love of technology. Since higher rates of student failure and lower rates of student retention in introductory programming courses are significant problems in the CS academic community, any approach that positively affects these rates will be welcomed by CS faculty.

Chapter 4

Results

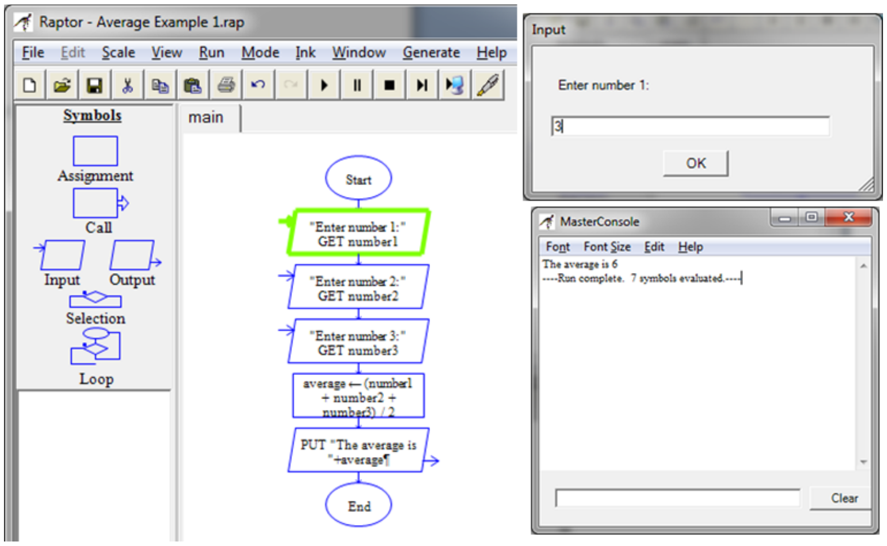
The fact that an overwhelming majority of today's CS students are visual learners (Howles, 2007; Sigle, 2008; Thomas, Ratcliffe, Woodbury, & Jarman, 2002) is one of the factors attributable to the decreasing enrollments and lower rates of retention for many CS departments (Chen & Lin, 2011; Gomes & Mendes, 2010; Gomes & Mendes, 2008; Gomes & Mendes, 2007; Gomes, Carmo, Bigotte, & Mendes, 2006; Kuri & Truzzi, 2002). Therefore, the challenge that CS educators currently face is to find the right blend of technology and pedagogy for their curriculum which will accommodate the current learning styles of today's CS students in order to help them be successful in their courses and persist through the major. The introductory sequence of courses in the CS major is where CS departments typically see the lowest retention rates (Zweben, 2008; Vegso, 2008; Yadin, 2011; Soe, Guthrie, Yakura, & Hwang, 2011; Guthrie, Yakura, & Soe, 2011; Becerra-Fernandez, Elam, & Clemmons, 2010; Sloan & Troy, 2008; Ali, 2009; Moskal, Lurie & Cooper, 2004; Forte & Guzdial, 2004; Chen & Morris, 2005; Herrmann et al., 2003; Talton, Peterson, Kamin, Israel, & Al-Muhtadi, 2006; Boyer, Dwight, Miller, Raubenheimer, Stallman, & Vouk, 2007). Successful completion of the introductory sequence of programming courses can be an overwhelming and sometimes impossible task for introductory programming students for many reasons, including the number and complexity of topics being presented, the use of languages not designed for teaching, and students' deficiency in basic problem solving skills (Urness & Manley, 2011; Yadin, 2011; Chen & Morris, 2005; Beaubouef & Mason, 2005; Moskal et al., 2004). These factors, combined with teaching that does not accommodate the visual learning style of the majority of CS majors, can combine to make learning programming in an introductory programming course (i.e., CS1) a difficult task. As a result, visual technologies have been explored as a way to present difficult introductory programming concepts in a manner that is easier to visualize and simpler to use.

The goal was to improve student retention and performance in the CS1 course at ETSU. The course has been redesigned and redeveloped to incorporate the use of visual technologies in introducing the topics of selection, repetition, objects and classes. Data collected throughout the course has been used to determine the effectiveness of the visual technologies on student performance in CS1 and persistence in the major.

Implementation

During the first semester of the study, the major topics in CS1 were presented to the control group in a manner which did not involve the use of visual technologies. Course topics were presented in a lecture format through the use of presentation slides and examples worked interactively in class. In-class examples involved the presentation of a problem, solving the problem by writing an algorithm solution in pseudocode, desk-checking the algorithm and then translating that algorithm into Java code for testing. All of these activities were done using paper and pencil or the whiteboard. The lab sessions were meant to supplement the lecture presentations by allowing the students to follow the same manual process of writing an algorithmic solution and then implementing that solution into Java code. An observation of this method of teaching was that, unless students were forced to turn in the algorithmic solution, most students would skip that step and begin the coding process immediately. If forced to submit an algorithmic solution, many students would complete the coding process and then write up a corresponding algorithm for submission, thus defeating the purpose for writing an algorithmic solution.

During the second semester of the study, the major topics in CS1 were presented to the treatment group in a manner which incorporated the use of visual technologies. RAPTOR was introduced in the first half of the semester when presenting the topics of selection (executing decisions in a computer program) and repetition (causing sections of program code to repeat). By using RAPTOR, students were able to visually see the creation of a solution using flowcharts. RAPTOR also allowed the algorithmic solution to be executed as though it were a program. With this feature, the students could see the steps executed and test the solution prior to ever writing any code, as can be seen in Figure 16.



```
"Enter number 1:"
GET number1
"Enter number 2:"
GET number2
"Enter number 3:"
GET number3
average ← (number1 + number2 + number3) / 2
PUT "The average is "+average¶
```

Enter number 1:

3

```
The average is 6
----Run complete.  7 symbols evaluated.----
```

Figure 16. Execution of a flowchart solution in RAPTOR.

As a RAPTOR flowchart was being executed, the students could observe the outcomes associated with corresponding decisions and visually identify where problems existed in the proposed solution. In other words, students could ensure their proposed solution was correct before they began writing program code. Students were not required to use RAPTOR outside of class; however, they responded positively to the use of RAPTOR in the classroom. As a result, many students gravitated toward the use of RAPTOR to develop algorithmic solutions for their assigned projects as opposed to the alternative of manually writing out pseudocode (i.e., textual algorithms). The positive response from students was in the form of verbal feedback to the instructors and written feedback on student evaluations of the courses. Students indicated that the use of RAPTOR helped them to visualize the concepts thus understanding them better and that they preferred to develop their solutions in a more visual, rather than textual, manner. Students also liked the fact that they could more easily test their solutions before implementing them in the Java programming language.

Alice was introduced in the second half of the semester when presenting the topics of objects and classes. By using Alice, students were able to see visual representations of classes, as shown in Figure 17, and the instantiation of objects of those classes, as shown in Figure 18. They were able to visualize the concepts associated with objects and classes, such as the state (current properties of the object), behavior (actions associated with the object) and identity (unique name) of an object as illustrated in Figure 19.

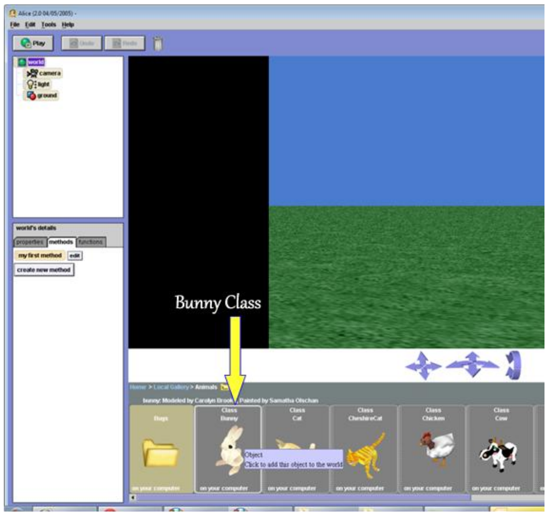


Figure 17. Selecting a Class in Alice.

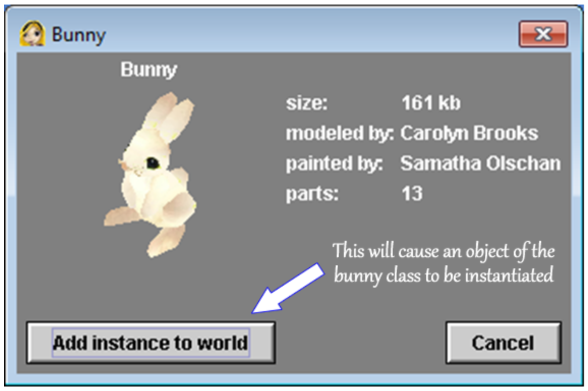

Bunny
Bunny
size: 161 kb
modeled by: Carolyn Brooks
painted by: Samatha Olschan
parts: 13
This will cause an object of the bunny class to be instantiated
Add instance to world
Cancel

Figure 18. Instantiating an object of a class in Alice.

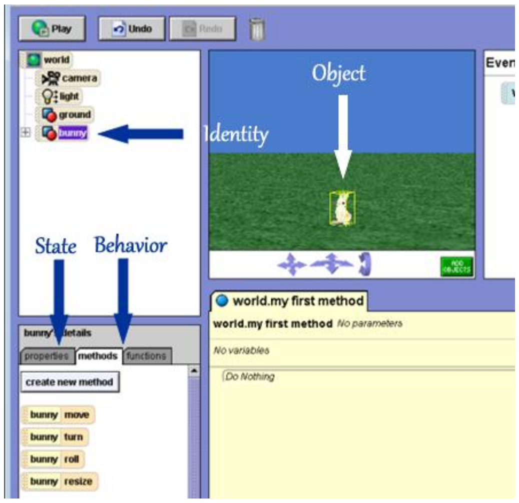


Figure 19. State, Behavior and Identity of an object in Alice.

To evaluate student comprehension and mastery of major topics in CS1, checkpoints were administered as quizzes to the control group during the first semester of the study and to the treatment group during the second semester of the study. Each checkpoint was administered to the students after the topic had been presented and students had been given the opportunity to complete lab exercises and a major programming assignment on that topic. Once final grades were recorded at the end of the second semester of the study, grades from these checkpoints were compiled into an Excel spreadsheet, and Minitab was used to analyze the results.

Evaluation

To determine if there were any statistically significant improvements in student performance in the treatment group as compared to the control group, *t*-tests were performed on the data. A P-value of 0.05 or less as a result of the *t*-test indicates statistically significant improvement between the performance of the control group and the treatment group. A P-value between 0.05 and 0.10, although not considered statistically significant, indicates improvement between the performances of the two groups.

Selection

The selection checkpoint was designed to test students' comprehension of decision constructs in programming. The checkpoint questions were designed to cover simple if/else statements, nested if/else statements, switch statements, complex logic and some syntax related to the Java programming language. Because the use of RAPTOR does not directly pertain to switch statements, complex logic and Java syntax, questions related to those topics were not included in the comparisons. Questions 1, 2, 10, and 11 relate to simple if/else statements; and questions 3, 4, and 6 relate to nested if/else statements. As a result, the set of questions 1, 2, 3, 4, 6, 10, and 11 comprise the selection checkpoint as it relates to the comparisons being performed.

As can be seen in Table 2, the results of the performance on the selection checkpoint are positive. When comparing the treatment and control groups in their entirety with a *t*-test, the difference was significant at the 0.05 level. The treatment group showed statistically significant improvement with an average of 92.4, compared to an average of 83.2 for the control group. Then, a two-way analysis of variance was performed to analyze the method of teaching and major simultaneously. Majors performed better than non-majors in both groups (treatment and control). The treatment group performed better on average than the control group for both majors and non-majors, showing statistically significant improvement. The results of the Two-way ANOVA, as shown in the Interaction Plot for Selection in Figure 20, indicate statistical significance in the improvement in the mean score for the treatment group when compared to the control group, regardless of major.

Table 2

Comparison of Student Performance on Selection Topic – Statistical Analysis

Using a *t*-test to compare performance between the Control Group and Treatment Group					
		N	*Mean*	*Std. Dev.*	*P-Value*
Entire Class	Control Group	74	83.2	16.6	0.000 ✦
	Treatment Group	77	92.4	10.0	
Means by Treatment and Major					
Control	Major	40	84.6	16.3	
	Non-Major	34	81.5	17.1	
Treatment	Major	60	92.9	9.7	
	Non-Major	17	90.8	11.2	
Two-Way ANOVA to analyze simultaneously Treatment and Major					
Factor	*Treatment*		*Major*		*Interaction*
P-Value	0.001 ✦		0.290		0.835

✦ *indicates a P-value < 0.05 indicating improvement of statistical significance*
* *indicates a P-value between 0.05 and 0.10 indicating positive improvement*

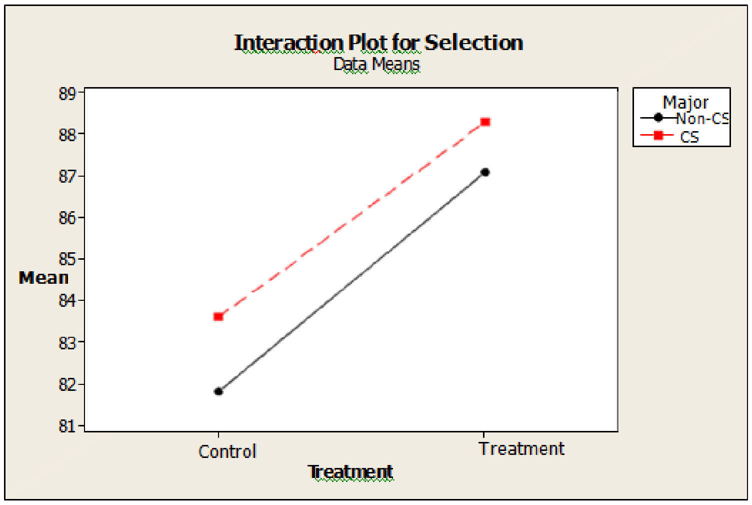


Figure 20. Two-way ANOVA Interaction Plot for Selection.

The complete results of the selection checkpoint are shown in Table 3. As can be seen, the results of the performance on the simple if/else statements subsection of the selection checkpoint are very positive overall. Average scores were 9 to 11 points higher for the treatment group as compared to the control group. The results of the nested if/else statements subsection were also positive overall with average scores ranging from 4 to 10 points higher for the treatment group as compared to the control group.

Table 3

Comparison of Student Performance on Selection Checkpoint – Descriptive Statistics

		Entire Class			*Majors*			*Non-Majors*		
		N	**Mean**	**Std. Dev.**	**N**	**Mean**	**Std. Dev.**	**N**	**Mean**	**St. Dev.**
Question 1	Control	74	98.6	11.6	40	100.0	0.0	34	97.1	17.1
	Treatment	77	100.0	0.0	60	100.0	0.0	17	100.0	0.0
Question 2	Control	74	83.8	37.1	40	80.0	40.5	34	88.2	32.7
	Treatment	77	90.9	28.9	60	91.7	27.9	17	88.2	33.2
Question 10	Control	74	86.5	34.4	40	92.5	26.7	34	79.4	41.0
	Treatment	77	94.8	22.3	60	96.7	18.1	17	88.2	33.2
Question 11	Control	74	63.5	48.5	40	67.5	47.4	34	58.8	50.0
	Treatment	77	92.2	27.0	60	95.0	22.0	17	82.4	39.3
If/Else Subtopic	Control	74	**83.1**	19.9	40	**85.0**	20.3	34	**80.9**	19.5
	Treatment	77	**94.5**	11.9	60	**95.8**	9.4	17	**89.7**	17.8
Question 3	Control	74	87.8	32.9	40	92.5	26.7	34	82.4	38.7
	Treatment	77	81.8	38.8	60	80.0	40.3	17	88.2	33.2
Question 4	Control	74	98.6	11.6	40	97.5	15.8	34	100.0	0.0
	Treatment	77	98.7	11.4	60	98.3	12.9	17	100.0	0.0
Question 6	Control	74	63.5	48.5	40	62.5	49.0	34	64.7	48.5
	Treatment	77	88.3	32.3	60	88.3	32.4	17	88.2	33.2
Nested If/Else Subtopic	Control	74	**83.3**	22.2	40	**84.2**	21.3	34	**82.4**	23.5
	Treatment	77	**89.6**	16.5	60	**88.9**	17.0	17	**92.2**	14.6
Overall Performance	Control	74	**83.2**	16.6	40	**84.6**	16.3	34	**81.5**	17.1
	Treatment	77	**92.4**	10.0	60	**92.9**	9.7	17	**90.8**	11.2

Repetition

The repetition checkpoint was designed to test students' comprehension of looping constructs in programming. The checkpoint questions were designed to cover two types of looping, count-controlled loops and event-controlled loops, as well as a looping algorithm written in pseudocode. Because the use of RAPTOR does not directly pertain to complex logic and Java syntax, questions related to those topics were not included in the comparisons. Questions 1, 2, 3, and 6 relate to count-controlled loops; questions 4, 5, and 7 relate to event-controlled loops; and question 10 is a pseudocode-based looping algorithm question. As a result, the set of questions 1, 2, 3, 4, 5, 6, 7, and 10 comprise the repetition checkpoint as it relates to the comparisons being performed.

The results of the performance on the repetition checkpoint, as shown in Table 4, indicate improvement that is on the verge of statistical significance. When comparing the treatment and control groups in their entirety with a *t*-test, although not significant at the 0.05 level, the difference was on the verge of statistical significance with a P-value of 0.054. The treatment group showed positive improvement overall with an average of 74.8 compared to an average of 68.5 for the control group. Then, a two-way analysis of variance was performed to analyze the method of teaching and major simultaneously. Majors performed better than non-majors in both groups (treatment and control). The treatment group performed better on average than the control group for both majors and non-majors. However, there is so much individual variability in the scores that the differences are not significant when tested. The results of the Two-way ANOVA can also be seen in the Interaction Plot for Repetition in Figure 21. This visually demonstrates the fact that the mean score for the treatment group on the repetition checkpoint is higher than that for the control group, regardless of major. The graph also indicates that there is a more significant improvement for majors than for non-majors.

Table 4

Comparison of Student Performance on Repetition Topic – Statistical Analysis

Using a *t*-test to compare performance between the Control Group and Treatment Group					
		N	*Mean*	*Std. Dev.*	*P-Value*
Entire Class	Control Group	65	68.5	24.6	0.054*
	Treatment Group	68	74.8	20.4	
Means by Treatment and Major					
Control	Major	38	70.4	23.9	
	Non-Major	27	65.8	25.9	
Treatment	Major	54	88.6	11.2	
	Non-Major	17	85.3	13.7	
Two-Way ANOVA to analyze simultaneously Treatment and Major					
Factor	*Treatment*		*Major*	*Interaction*	
P-Value	0.346		0.131	0.642	

✦ indicates a P-value < 0.05 indicating improvement of statistical significance
** indicates a P-value between 0.05 and 0.10 indicating positive improvement*

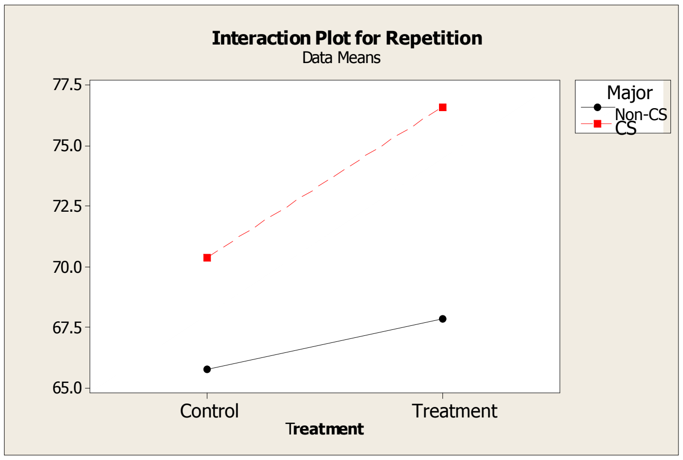


Figure 21. Two-way ANOVA Interaction Plot for Repetition.

The complete results of the repetition checkpoint are shown in Table 5. The biggest improvement on the repetition checkpoint was in the set of questions related to count-controlled looping. Results of the performance on this subsection of the selection checkpoint are very positive overall. Average scores were 10.2 to 18.1 points higher for the treatment group as compared to the control group, indicating significant improvement. The results of the performance on the event-controlled looping subsection and the pseudocode question were also improved, but only by a few points therefore indicating only slight improvement.

Table 5

Comparison of Student Performance on Repetition Checkpoint – Descriptive Statistics

		Entire Class			***Majors***			***Non-Majors***		
		N	**Mean**	**Std. Dev.**	**N**	**Mean**	**Std. Dev.**	**N**	**Mean**	**St. Dev.**
Question 1	Control	65	84.6	36.4	38	78.9	41.3	27	92.6	26.7
	Treatment	68	94.1	23.7	54	94.4	23.1	14	92.9	26.7
Question 2	Control	65	63.1	48.6	38	63.2	48.9	27	63.0	49.2
	Treatment	68	82.4	38.4	54	85.2	35.9	14	71.4	46.9
Question 3	Control	65	50.8	50.4	38	57.9	50.0	27	40.7	50.1
	Treatment	68	54.4	50.2	54	57.4	49.9	14	42.9	51.4
Question 6	Control	65	80.0	40.3	38	73.7	44.6	27	88.9	32.0
	Treatment	68	88.2	32.5	54	88.9	31.7	14	85.7	36.3
Count Controlled Subtopic	Control	65	**69.6**	26.7	38	**68.4**	28.3	27	**71.3**	24.7
	Treatment	68	**79.8**	20.8	54	**81.5**	19.5	14	**73.2**	24.9
Question 4	Control	65	81.5	39.1	38	84.2	37.0	27	77.8	42.4
	Treatment	68	82.4	38.4	54	85.2	35.9	14	71.4	46.9
Question 5	Control	65	56.9	49.9	38	60.5	49.5	27	51.9	50.9
	Treatment	68	69.1	46.5	54	66.7	47.6	14	78.6	42.6
Question 7	Control	65	69.2	46.5	38	78.9	41.3	27	55.6	50.6
	Treatment	68	61.8	39.0	54	66.7	47.6	14	42.9	51.4
Event Controlled Subtopic	Control	65	**69.2**	30.8	38	**74.6**	23.8	27	**61.7**	37.8
	Treatment	68	**71.1**	30.4	54	**72.8**	30.4	14	**64.3**	30.6
Question 10	Control	65	61.5	49.0	38	65.8	48.1	27	55.6	50.6
	Treatment	68	66.2	47.7	54	68.5	46.9	14	57.1	51.4
Overall Performance	Control	65	**68.5**	24.6	38	**70.39**	23.9	27	**65.7**	25.8
	Treatment	68	**74.8**	20.4	54	**76.6**	19.7	14	**67.9**	22.3

Objects and Classes

The objects and classes checkpoint was designed to test students’ comprehension of basic object-oriented concepts in programming. Questions not directly related to declaring objects of classes or invoking class methods were not included in the comparisons. As a result, the set of questions 7, 8, 9, and 10 comprise the objects and classes checkpoint as it relates to the comparisons being performed.

As can be seen in Table 6, the results of the performance on the objects and classes checkpoint showed statistically significant improvement. When comparing the treatment and control groups in their entirety with a *t*-test, the difference was significant at the 0.05 level. The average score for the treatment group was 11 points higher than the control group. Then, a two-way analysis of variance was performed to analyze the method of teaching and major simultaneously. The treatment group performed better on average than the control group for both majors and non-majors, showing statistically significant improvement. The results of the Two-way ANOVA, as shown in the Interaction Plot for Objects/Classes in Figure 22, indicate statistical significance in the improvement in the mean score for the treatment group when compared to the control group, regardless of major. The graph also indicates that there is a higher statistically significant improvement for non-majors than majors.

Table 6

Comparison of Student Performance on Objects/Classes Topic – Statistical Analysis

Using a *t*-test to compare performance between the Control Group and Treatment Group					
		N	*Mean*	*Std. Dev.*	*P-Value*
Entire Class	Control Group	62	64.5	32.2	0.034 ✦
	Treatment Group	53	75.5	31.6	
Means by Treatment and Major					
Control	Major	35	65.0	33.3	
	Non-Major	27	63.9	31.3	
Treatment	Major	46	73.9	33.3	
	Non-Major	7	85.7	13.4	
Two-Way ANOVA to analyze simultaneously Treatment and Major					
Factor	*Treatment*		*Major*	*Interaction*	
P-Value	0.048 ✦		0.489	0.403	

✦ indicates a P-value < 0.05 indicating improvement of statistical significance
** indicates a P-value between 0.05 and 0.10 indicating positive improvement*

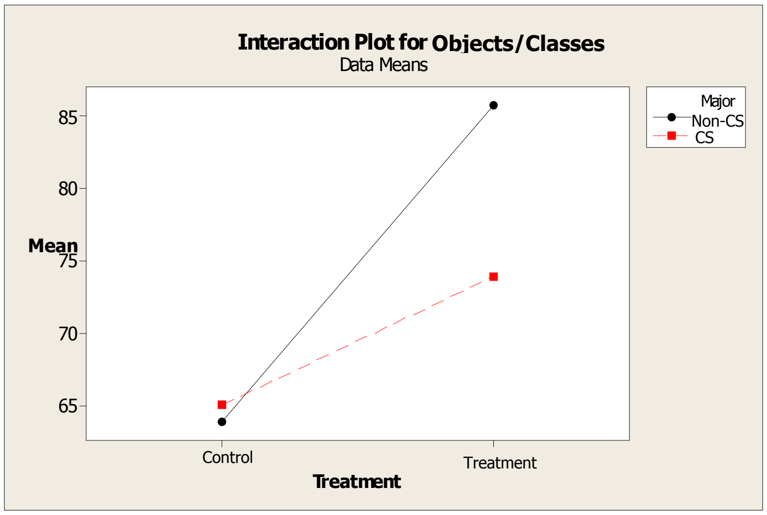


Figure 22. Two-way ANOVA Interaction Plot for Objects/Classes.

The complete results of the objects/classes checkpoint are shown in Table 7. While the results of the objects/classes checkpoint were statistically significant among the entire class, majors and non-majors, the biggest improvement on the objects/classes checkpoint was among non-majors. For non-majors, average scores were 22 points higher for the treatment group as compared to the control group.

Table 7

Comparison of Student Performance on Objects/Classes Checkpoint – Descriptive Statistics

		Entire Class			*Majors*			*Non-Majors*		
		N	**Mean**	**Std. Dev.**	**N**	**Mean**	**Std. Dev.**	**N**	**Mean**	**St. Dev.**
Question 7	Control	62	38.5	49.0	35	36.8	48.9	27	40.7	50.1
	Treatment	53	77.4	42.3	46	76.1	43.1	7	85.7	37.8
Question 8	Control	62	81.5	39.1	35	78.9	41.3	27	85.2	36.2
	Treatment	53	69.8	46.3	46	69.6	46.5	7	71.4	48.8
Question 9	Control	62	69.2	46.5	35	71.1	46.0	27	66.7	48.0
	Treatment	53	71.7	45.5	46	69.6	46.5	7	85.7	37.8
Question 10	Control	62	66.2	47.7	35	68.4	47.1	27	63.0	49.2
	Treatment	53	83.0	37.9	46	80.4	40.1	7	100.0	0.0
Overall Performance	Control	62	**64.5**	32.2	35	**65.0**	33.3	27	**63.9**	31.3
	Treatment	53	**75.5**	31.6	46	**73.9**	33.3	7	**85.7**	13.4

Final calculated grade for the course

The results of the performance in the entire course, as measured by the final calculated grade in the course, are presented in Table 8. When comparing the treatment and control groups in their entirety with a *t*-test, positive improvement was observed for the entire class, although not statistically significant at the 0.05 level. With an average grade of 73.7, the treatment group performed higher than the control group which had an average grade of 68.6. Then, a two-way analysis of variance was performed to analyze the method of teaching and major simultaneously. Majors performed better than non-majors in both groups (treatment and control) showing statistical significance. The treatment group also performed better on average than the control group for both majors and non-majors although not statistically significant. The results of the Two-way ANOVA can also be seen in the Interaction Plot for Final Grade in Figure 23. This visually demonstrates the fact that the mean score for the treatment group on the final calculated grade is higher than that for the control group, regardless of major. It also indicates that there is a significant difference between the average final calculated grade for majors versus non-majors.

Table 8

Comparison of Student Performance on Final Calculated Grade – Statistical Analysis

Using a *t*-test to compare performance between the Control Group and Treatment Group					
		N	*Mean*	*Std. Dev.*	*P-Value*
Entire Class	Control Group	81	68.6	26.6	0.093*
	Treatment Group	77	73.7	21.5	
Means by Treatment and Major					
Control	Major	45	72.3	27.0	
	Non-Major	36	64.0	25.8	
Treatment	Major	60	75.5	19.8	
	Non-Major	17	67.6	26.5	

Two-Way ANOVA to analyze simultaneously Treatment and Major			
Factor	*Treatment*	*Major*	*Interaction*
P-Value	0.431	0.061*	0.962

✦ indicates a P-value < 0.05 indicating improvement of statistical significance
** indicates a P-value between 0.05 and 0.10 indicating positive improvement*

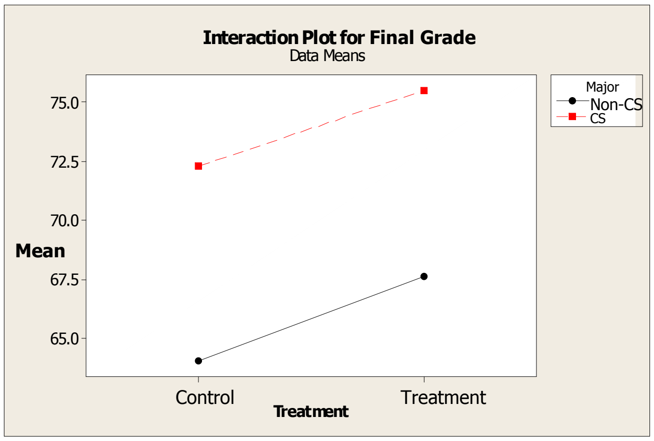


Figure 23. Two-way ANOVA Interaction Plot for Final Grade for the Course.

Student Retention and Persistence in the major

To determine if there was a statistically significant improvement in student retention in the treatment group as compared to the control group, the Fisher's exact test was performed on the data comparing the rates of dropout among majors during the semester. Because retention in the major does not apply to non-majors, this test was only applied to participants in each group who were CS majors. The Fisher's exact test was used, instead of the chi-squared test, because the values for the number of students who dropped out were so low. As with the *t*-test, a resulting P-value of 0.05 or less indicates statistically significant improvement between the performance of the control group and the treatment group. A P-value between 0.05 and 0.10, although not considered statistically significant, indicates improvement between the performances of the two groups.

Among majors, the control group saw a 2% dropout rate with 1 major out of 47 dropping the course before the end of the semester. The treatment group saw a 3% dropout rate with 2 majors out of 62 dropping the course before the end of the semester. The resulting P-value of 1 indicates that these results were statistically the same.

A chi-squared test was used for comparison of persistence rates in the major between the control group and treatment group. Persistence in the major is defined as successful completion of CS1 and subsequent enrollment in CS2 or re-enrollment in CS1 if the student received a failing grade on the first attempt at the course. Because persistence in the major does not apply to non-majors, this test was only applied to participants in each group who were CS majors.

Among majors, the control group saw a 72% persistence rate with 34 out of 47 declared majors continuing in the CS major. The treatment group saw an increase in the persistence rate with 84%, 52 of 62 declared majors, continuing in the CS major. However, despite the fact that the persistence rate increased by 12% as shown in Table 9, this was not a statistically significant change.

Table 9

Comparison of Student Retention in CS1 and Persistence in the Major

		Enrolled the entire semester	Dropped mid-semester	P-Value
Retention in CS1	Control Group	46 (98%)	1 (2%)	1
	Treatment Group	60 (97%)	2 (3%)	
		Continued in the major	**Dropped the major**	**P-Value**
Persistence in the CS major	Control Group	34 (72%)	13 (28%)	0.144
	Treatment Group	52 (84%)	10 (16%)	

✦ indicates a P-value < 0.05 indicating improvement of statistical significance
** indicates a P-value between 0.05 and 0.10 indicating positive improvement*

The analysis of the data collected indicates some level of improvement as a result of incorporating visual technologies in CS1 at ETSU. Conclusions and implications regarding the impact of the use of these visual technologies are presented in Chapter 5.

Chapter 5

Conclusions, Implications, Recommendations and Summary

In this chapter, conclusions are drawn regarding the use of visual technologies in the introductory programming course, CS1, as the research questions are answered. Implications and recommendations are made regarding the impact of the findings as they apply to CS education. Ideas for future research regarding the use of visual technologies in CS education will also be presented.

Conclusions

Research Question 1: What are the factors attributable to poor performance and low retention rates and what solutions have been reported?

The factors most commonly reported as attributable to poor performance and low retention rates in the CS1 course are student misconceptions about the CS field, poorly designed introductory programming courses, students being under-prepared for an introductory programming course, number and complexity of topics being introduced, use of industry-strength programming languages, and teaching and delivery styles that do not relate to today's visual learners (the majority of CS majors).

To address the problem of student misconceptions about the CS field, many universities have included a course in their CS curriculum that introduces students to the field and the types of career opportunities that exist in the field. This type of course focuses on what type of education is necessary for certain careers in the field and assists the students in selecting from the different concentrations offered such as computer science, information systems and information technology. ETSU has such a course; however, it also focuses on helping students to be successful in the university setting, practice time management skills and develop study skills. Therefore, this ETSU course is restricted to incoming freshman who have declared CS as their major. Students who transfer from other schools or other majors are not allowed to take this course and do not reap the benefits of taking the course.

Introductory programming courses that are poorly designed affect the retention of students in the CS1 course and the major. Three factors causing poorly designed CS1 courses are that the nature of introductory programming courses have become quite complex, that CS is a constantly evolving field and that CS educators generally lack exposure to educational theories and practices. These factors have made it hard for CS educators to design CS1 courses that adequately cover the large number of topics necessary to learn programming while also being accommodating to visual learners. The number and complexity of topics introduced in the CS1 course is further complicated by the trend for CS departments to want to provide students with exposure to and depth of experience with industry-strength programming languages. This has prompted a change from using languages designed for teaching programming concepts to using languages that make it more difficult to teach programming concepts to novice programmers. The use of industry-strength programming languages has also prompted a switch in the programming paradigm from procedural to object-oriented. This change in programming paradigm at the introductory level has contributed not only to the number but more importantly to the complexity of the topics introduced.

To address the problem of poorly designed introductory programming courses, the Department of Computing at ETSU has encouraged its faculty to participate in faculty development workshops related to improving teaching, course development and student learning. Over the past several years, ETSU has invited guest lecturers to offer faculty development workshops on improving courses and teaching. In 2011, Tom Angelo offered a series of faculty development workshops presenting research-based strategies for improving teaching, assessment and learning. In 2013, Harvey Brightman presented a series of teaching workshops designed to help faculty identify critical factors that affect student performance and demonstrate how to develop a course framework and course presentations designed to stimulate student interest in a subject. These types of faculty development workshops offer CS faculty the opportunity to supplement their CS knowledge and background with the tools necessary to develop courses that can effectively present subject matter to students in an engaging manner.

Students who enter the CS major are often under-prepared for the major. These students lack the problem solving skills and the pre-college preparation required to succeed in a program that is very complex in its nature. To address this problem, many CS departments have incorporated a CS0 course as a pre-requisite to the typical introductory CS1 programming course. This type of course is generally designed to improve students' problem solving skills and introduce them to introductory programming concepts without the complexity of industry-strength programming languages. This is commonly accomplished through the use of simpler IDEs and visual technologies that make it easier for the student to create solutions to programming problems and to visualize the concepts being presented.

The visual technologies that are used in CS0 courses not only simplify the environment in which the student is learning basic programming constructs, but they also accommodate the learning styles of the majority of today's CS students by visually presenting concepts and giving the students visual environments in which to develop solutions and implement those solutions in code. In fact, visual technologies have been developed to help address the majority of the factors identified as attributing to poor performance and retention rates in introductory programming courses. However, these technologies have not widely been adopted in CS1 courses. Reasons for the reluctance to use visual technologies include the fact that CS faculty are unaware of the benefits of using visual technologies to present introductory programming concepts, CS departments are reluctant to replace the programming languages and development environments currently being used with simpler, more visual languages and development environments, and it may be difficult to cover all CS1 concepts if visual technologies are included in CS1.

To address the problem of students being under-prepared for the major, the Department of Computing at ETSU offers a CS0 course. However, due to the fact that the department is under-staffed and cannot offer the number of sections necessary to implement it as a requirement for all students, it is an elective course for CS majors. Students who enter the major with low ACT math scores, have not performed well in high school or college math courses, or have had no experience with programming are encouraged to take the CS0 course. Because it is not a requirement for the major, only a fraction of students who enter the major actually take the CS0 course.

Research Question 2: How can the introductory course, CS1, be redeveloped and implemented to incorporate visual technologies?

Using the ADDIE approach to instructional design, the CS1 course at ETSU was redesigned and redeveloped to incorporate visual technologies. During the analysis phase, the learning style of today's students, the visual technologies appropriate for use in introductory courses, and the instructional goals and objectives for the course were identified for the CS1 course at ETSU.

Unlike many CS courses that adopt one visual technology to be used in the course, it was decided that the CS1 course at ETSU would be redeveloped to incorporate two visual technologies, one to support algorithm development and one to support object-oriented programming development. The two visual technologies selected for inclusion in the CS1 course at ETSU were RAPTOR which supports algorithm development, and Alice which supports object-oriented program development. RAPTOR was used to introduce the topics of selection and repetition. In addition to demonstrating the development and testing of algorithmic solutions with pseudocode, RAPTOR was used to visually demonstrate to the students how the solution would work if executed. This also helped in testing the solution and determining if there were any errors before translating the solution into Java code. RAPTOR was also used by the students throughout the semester as an interactive tool for creating and testing their own algorithmic solutions to the programming problems that they were assigned.

Alice was used to visually introduce the basic concepts of object-oriented programming and to give the students some exposure to objects and classes in a visual environment. Rather than replace the use of the Java programming language in CS1 with Alice, the course was supplemented with Alice. The students were given exercises to complete using Alice to become familiar with using objects and classes and to specifically help them draw a parallel between objects and classes used in Alice and similar objects and classes to be used or developed in the Java programming language.

The design phase involved the integration of the selected visual technologies into the course content and instructional delivery methods. The assessment techniques were also designed during this phase as well as when they would be administered throughout the semester.

During the development phase, course lecture notes and in-class exercises were developed to incorporate the use of RAPTOR for the topics of selection and repetition and the use of Alice to present the topics of objects and classes. The checkpoints for assessment, to be used by all sections of CS1, were also developed during this phase.

The redesigned CS1 course was reviewed and approved by the Introductory Programming Committee at ETSU and subsequently distributed to all faculty teaching sections of CS1. Faculty members teaching the revised version of CS1 were also trained on the visual technologies to be used in the course.

The implementation phase is the phase during which the CS1 course was taught in its original form during the first semester of the study and in its revised form during the second semester of the study. Once the revised course had been taught, analysis was performed on the data collected during both semesters of the study. The analysis of the data collected was performed during the final phase of the ADDIE approach, the evaluation phase.

Research Question 3: What are the outcomes of teaching the redesigned course?

During the evaluation phase of the ADDIE approach to the redevelopment and redesign of the CS1 course at ETSU, assessment was performed using the Post-Test Only Nonequivalent Groups Design approach. The control group, consisting of 93 students (47 CS majors and 46 non-CS majors), was the collection of students who took the original version of the CS1 course (instruction without the use of visual technologies) at ETSU during the first semester of the study. The treatment group, consisting of 80 students (62 CS majors and 18 non-CS majors), was the collection of students enrolled in the CS1 course during the second semester of the study, in which the revised curriculum (instruction with the use of visual technologies) was implemented.

Of the 93 students who participated in the control group, 10 dropped the course, 52 passed and 31 failed. Of the students who participated in the control group, 47 were majors. Among the majors, 1 student dropped during the semester. Upon the completion of the CS1 course, 34 (72%) of the majors persisted in the major while 13 (28%) of the majors did not.

Of the 80 students who elected to participate in the treatment group, 3 dropped, 53 passed and 24 failed. Of the students who participated in the treatment group, 62 were majors. Among the majors, 2 students dropped during the semester. Upon the completion of the CS1 course, 52 (84%) of the majors persisted in the major while 10 (16%) of the majors did not.

These results indicate that, while not a statistically significant improvement at the 0.05 level, the persistence of students in the major at the introductory courses level was improved. The number of students dropping the course throughout the semester was small for both the treatment and control group and was statistically the same, therefore showing no improvement.

Among the groups as a whole, statistically significant improvement was shown for the treatment group on the topic of selection in general as well as selection subtopics of if/else statements and nested if/else statements, the repetition subtopic of count-controlled loops and the topic of objects and classes. Evidence of positive improvement, although not statistically significant, was also observed for the topic of repetition in general and the final grade assigned for the course.

Among both CS majors and non-majors, positive improvement was shown for the treatment group on the topics of selection, repetition and objects and classes. Positive improvement was also shown on the selection subtopics of if/else statements and nested if/else statements and the repetition subtopic of count-controlled loops.

Informal feedback from students indicated that they were positive about the use of RAPTOR and Alice in the CS1 course. Several students made positive comments regarding the software to the instructors throughout the semester and a few students gave written feedback regarding the use of the software on student evaluations of the courses. Students indicated that the use of RAPTOR helped them to visualize the concepts thus understanding them better and that they preferred to develop their solutions in a more visual, rather than textual, manner. Students also liked the fact that they could more easily test their solutions before implementing them in the Java programming language.

The professors involved in teaching the courses were also positively impacted by the use of the visual technologies in the CS1 course. They are now aware of the need to incorporate more visual teaching tools and techniques into the programming courses. One professor had already used flowcharts to represent solutions to problems in a visual manner, although he had not used flowcharting software. The other professor had never used flowcharts at all. Both, however, realized the impact that using visual representations can have on student understanding of the concepts and how the use of flowcharting software encouraged the students to both develop and test solutions prior to implementing them in the Java programming language. The professors involved in the study are committed to using more visual-based teaching tools in the future.

Research Question 4: What conclusions may be drawn regarding the value of the new curricula in terms of student performance and retention?

Analysis performed on the data collected from both the control group and treatment group indicates that statistically significant improvement was observed for the treatment group among several different areas of the CS1 course. The use of RAPTOR in a CS1 course can have a significant impact on student performance regarding if/else statements and nested if/else statements for decision constructs and regarding count-controlled loops for repetition constructs. RAPTOR, however, did not show significant improvement on event-controlled loops for repetition constructs. Overall, students showed a better understanding of selection and repetition constructs from using RAPTOR. The use of Alice in a CS1 course can have a significant impact on student performance in defining and using objects and classes. Finally, the use of visual technologies proved to have a positive impact on student performance for the overall grade for the course. Although not statistically significant, final grades for students in the treatment group improved 1 to 8 points.

Implications and Recommendations

In the greater scheme of things, student performance and retention in CS1 courses are critical to the success of the CS department. Much research has been done in efforts to address these two problems. Solutions that include technologies showing improvement in student performance are being sought by CS faculty. In this research, it is shown that the visual technologies of RAPTOR and Alice, when incorporated in a CS1 course, contributed to the improvement of CS1 student performance.

Before the close of the Spring 2013 semester, a summary document will be presented to the Introductory Programming Committee at ETSU summarizing the results of this research along with recommendations for modifications to the CS1 course at ETSU. Because the results in this research were extremely positive in the use of RAPTOR, it will be recommended that RAPTOR be incorporated in CS1 on a permanent basis for teaching selection and repetition constructs. More precisely, it will be recommended that RAPTOR be added to the course curriculum and be required by students in the development and testing of all programming assignments.

Although Alice was shown to be helpful to students in the understanding of objects and classes, it will be recommended that it only be a supplemental tool in CS1. Alice seems better suited for visually introducing the concepts of objects and classes rather than including it in the course curriculum or requiring students to use it. In other words, it will be suggested that faculty only use Alice in lectures to introduce and illustrate the concepts of objects and classes. Remembering that one of the problems with CS1 is the number of topics included in the curriculum, this approach will allow students to see objects and classes visually but will not overburden the course and thus the students with another technology to be taught and learned.

The results of this research have opened up several possibilities for future lines of research. CS departments are continually addressing the need to attract and retain women and minorities in the field of computer science. Expanding the analysis to examine student performance and retention among the women and minorities in the CS1 courses at ETSU is a future goal of this research.

This research has also illuminated issues with the sequencing of the course topics in CS1. For example, is it better to focus on repetition and selection first or to focus on objects and classes first? Additionally, does more time and effort need to be spent on repetition and selection in CS1 before introducing objects and classes? Finally, would the visual technologies also be effective if incorporated into the CS2 curriculum? The researcher and other members of the Introductory Programming Committee at ETSU will continue to investigate these questions prior to the 2013-2014 academic year, and the results will be presented in a follow up summary document to the Introductory Programming Committee at ETSU for possible modification of the introductory programming sequence.

Although it was not implemented in this study, a follow-up survey to students during the treatment semester would be beneficial to obtain more formal feedback from the students regarding the use of the visual technologies. This would provide feedback from the student point of view regarding the benefits of using the software as well as any challenges that may have been encountered using the software.

Due to the fact that such a high percentage of today's CS students are visual learners, it is possible that other disciplines are also experiencing this same trend. Therefore, it may be beneficial for other disciplines to explore the use of visual technologies in their curricula as well. For example, any discipline in which problem solving is a key component could potentially benefit from using visual flowcharting software such as RAPTOR in its courses.

Summary

To a CS educator, the problems that the CS academic community is facing are very real and very troubling. The statistics regarding student success and retention in the freshman year of the CS major are troubling and even at times alarming. This combined with declining enrollment in the CS major and increased pressure among universities to base department funding on student retention are causes for concern among CS educators. So, what is the solution? How can we recruit more students into the discipline and more importantly improve student retention and success so as to keep the existing majors and help them to successfully complete a difficult and challenging program? A significant amount of time has been dedicated to researching the answers to these questions. This research expands upon that research to find answers to those questions as they relate to the CS program at ETSU.

The first phase of the research was to fully explore the problems that the CS academic community is facing, identify what factors are attributable to the problems of poor performance and low retention rates specifically within the freshman level introductory programming courses and what solutions have been reported upon to address these issues. As a result, the factors identified regarding the introductory level programming courses included student misconceptions of the field, poorly designed courses, students being under-prepared, number and complexity of topics being introduced, use of industry-strength programming languages and teaching and delivery styles that do not relate to today's visual learners, a category into which many CS majors fall. The solutions presented to address many of these issues include: the addition of a course to educate students about the field, the different career opportunities within the field and the type of education necessary for those career choices; faculty development and educational training to improve the design of the introductory level courses; the inclusion of a prerequisite to the CS1 course, commonly referred to as CS0, to improve students' problem solving and algorithmic thinking skills at a lower level without the complexity of development environments and languages that are typically used in introductory level programming courses; and the use of visual technologies to simplify the environment and the language used to introduce programming and solve problems.

Like many other universities, ETSU has embraced several of these solutions in an attempt to improve the learning experience and success of the students in the introductory programming courses. ETSU offers a course that familiarizes the student with the various aspects of the CS field and the education requirements necessary to succeed and pursue a CS related career. ETSU also offers a CS0 course that serves as a prerequisite to the CS1 course and gives students more exposure to problem solving and algorithmic development skills using technologies that simplify the problem solving and programming process. These courses, however, are not required of all CS majors. ETSU has also provided the CS faculty with many opportunities to improve teaching and course development skills through faculty development workshops and educational training. However, these solutions have not shown dramatic improvement in student success and retention in the introductory programming courses. Therefore, other solutions needed to be explored.

The next phase of the research was to explore the use of visual technologies as a solution to the problems of student success and retention in the introductory programming courses. After researching the different types of visual technologies appropriate for use in a CS1 course, the Introductory Programming Committee at ETSU selected two visual technologies to be incorporated into the CS1 course at ETSU. The technologies selected were RAPTOR, to introduce the topics of decisions and repetition in programming, and Alice, to introduce the topics of objects and classes in object-oriented programming.

The following phase of the research was to redesign and redevelop the CS1 course at ETSU to incorporate the use of RAPTOR and Alice into the course. Course materials were developed to include RAPTOR and Alice where appropriate in the course. Evaluation tools were also developed to measure the effect of the use of the visual technologies on student performance and retention.

After redesigning the course, the modified course was taught and data was collected among the treatment group of students. The treatment group of students consisted of students enrolled in CS1 during the second semester of the study who elected to participate in the study. Data collected from the treatment group was to be compared to data collected the previous semester from the control group. The control group consisted of students enrolled in CS1 prior to the redesign of the CS1 course to include the use of visual technologies.

Upon completion of the data collection, the data were analyzed to determine the effect of the use of visual technologies on student performance and retention in CS1 at ETSU. The results were largely positive indicating that the use of RAPTOR and Alice did have a positive impact on student performance and retention. Some areas of the course were more positively impacted than others, but all areas of the course that were selected for inclusion in the study showed an indication of improvement whether statistically significant or not.

The results indicated that the use of visual technologies in the CS1 course can have a positive impact on student performance in the course and retention in the major. As a result, it has been proposed that the Department of Computing at ETSU consider adoption of the revised CS1 course into the curriculum and potentially explore ways in which the visual technologies may be incorporated into the CS2 course as well for further exposure and possible additional benefits to the students.

Appendix A

Data Collection Instruments

CSCI-1250

Student Questionnaire

Student Name:____

Are you a major in the Computer & Information Sciences Department? ❑ Yes ❑ No

If Yes, which concentration are you?

❑ CS (Computer Science)

❑ IS (Information Systems Science)

❑ IT (Information Technology)

❑ I don’t know

If No, what is your major? ____

Have you previously attempted CSCI-1250 at ETSU? ❑ Yes ❑ No

Are you at least 18 years of age? ❑ Yes ❑ No

CS1 Checkpoints

The following are checkpoints that have been created to assess student comprehension of the major programming concepts presented in CS1.

Checkpoints have been created for each of the following programming concepts:

- Selection (decisions)
- Repetition (looping)
- Classes/Objects

The checkpoints have been developed as an assessment in D2L so they can be distributed to all students in all sections of CS1 after the corresponding topic has been covered. The reporting feature will make it easy to collect data on the student responses.

CS1 – **Selection** Checkpoint

Question 1.

```
grade = 60;
if (grade > 70)
        System.out.println(“PASS”);
else if (grade < 70)
        System.out.println(“FAIL”);
```

Given the code above, what would the output be?

a. PASS
b. FAIL
c. Nothing at all

Question 2.

```
grade = 70;
if (grade > 70)
        System.out.println(“PASS”);
else if (grade < 70)
        System.out.println(“FAIL”);
```

Given the code above, what would the output be?

a. PASS
b. FAIL
c. Nothing at all

Questions 3-6.

```
int ACT;
double GPA;
      :
//code here to get ACT & GPA from user
      :
      :
if ((ACT < 18) && (GPA < 2.00))
      System.out.println("REJECT");
else
{
      if ((ACT > 18) && (GPA > 2.00))
            System.out.println("ACCEPT");
      else
            if ((ACT > 27) || (GPA > 3.85))
                  System.out.println("ACCEPT UNCONDITIONALLY");
}
```

Given the code above, what is the output if ACT = 19 and GPA = 3.98?

[✓ a. ACCEPT]
b. REJECT
c. ACCEPT UNCONDITIONALLY
d. No output will be displayed

Given the code above, what is the output if ACT = 15 and GPA = 1.98?

a. ACCEPT
[✓ b. REJECT]
c. ACCEPT UNCONDITIONALLY
d. No output will be displayed

Given the code above, what is the output if ACT = 28 and GPA = 3.9?

[✓ a. ACCEPT]
b. REJECT
c. ACCEPT UNCONDITIONALLY
d. No output will be displayed

Given the code above, what is the output if ACT = 18 and GPA = 3.85?

a. ACCEPT
b. REJECT
c. ACCEPT UNCONDITIONALLY
[✓ d. No output will be displayed]

Question 7.

```
rating = 7;
switch (rating)
{     case 1 :    System.out.println("Best");
                  break;
      case 3 :
      case 5 :    System.out.println("Better");
                  break;
      case 7 :
      case 9 :    System.out.println("Good");
                  break;
      default:    System.out.println("Incorrect!");
}
```

Given the code above, what would the output be?

a. Best
b. Better
c. Good
d. Incorrect
e. Best
   Better
   Good
   Incorrect
f. No output would be displayed

Question 8.

```
rating = 6;
switch (rating)
{     case 1 :    System.out.println("Best");
                  break;
      case 3 :
      case 5 :    System.out.println("Better");
                  break;
      case 7 :
      case 9 :    System.out.println("Good");
                  break;
      default:    System.out.println("Incorrect!");
}
```

Given the code above, what would the output be?

a. Best
b. Better
c. Good
d. Incorrect
e. Best
   Better
   Good
   Incorrect
f. No output would be displayed

Question 9.

```
rating = 5;
switch (rating)
{     case 1 :    System.out.println("Best");
      case 3 :
      case 5 :    System.out.println("Better");
      case 7 :
      case 9 :    System.out.println("Good");
      default:    System.out.println("Incorrect!");
}
```

Given the code above, what would the output be?

a. Best
b. Better
c. Good
d. Incorrect
e. Best
   Better
   Good
   Incorrect
f. Better
   Good
   Incorrect
g. No output would be displayed

Question 10.

```
int a = 1;
int b = 10;

if (a < 1)
   a = 10;
if (b > 5)
   a = 30;
else
   a = 40;
System.out.println(a + "  " + b);
```

Given the code above, what would the output be?

a. 1 10
b. 10 10
[✓ c. 30 10]
d. 40 10

Question 11.

```
int a = 1;
int b = 10;
if (a < 1)
   a = 10;
if (b > 5)
   a = 30;
if (a <= 30)
   a = 40;
System.out.println(a + "  " + b);
```

Given the code above, what would the output be?

a. 1 10
b. 10 10
c. 30 10
[✓ d. 40 10]

Question 12.

```
int a = 1;
int b = 5;
if (a <=  b)
{
   b = 15;
   System.out.println(a + "  " + b);
}
else
    b = 20;
   System.out.println(a + "  " + b);
```

Given the code above, what would the output be?

a. 1 15
b. 1 20
c. 1 15
   1 15
d. 1 20
   1 20

Question 13.

Will the following evaluate to True or False?

10 < 12 && 5 > 5

a. True
b. False

Question 14.

Will the following evaluate to True or False?

12 < 10 || 15 > 5

a. True
b. False

Question 15.

Will the following evaluate to True or False?

((5 > 0 || 10 < 10) && 6 == 6)

a. True
b. False

Question 16.

Which of the following will check for a **valid** value for gender, M or F, regardless of upper or lower case?

```
a. if (gender == 'M'&& gender == 'm' && gender == 'F' && gender == 'f')
b. if (gender == 'M'|| gender == 'm' || gender == 'F' || gender == 'f')
c. if (gender != 'M'&& gender != 'm' && gender != 'F' && gender != 'f')
d. if (gender != 'M'|| gender != 'm' || gender != 'F' || gender != 'f')
```

CS1 – **Repetition** Checkpoint

Note: *the questions in this checkpoint will only be presented to the user one at a time because of the nature and progression of some of the questions.*
(One per page, one page at a time, without the ability to return to previously answered questions)

Questions 1-3.

```
sum = 0;
count = 0;
while (count <= 5)
{
     sum = sum + count;
     count++;
}
System.out.print(sum + “ “);
```

What kind of loop is this?

[✓ a. Count controlled]
b. Event controlled

How many times will the loop in the code above execute?

a. 0
b. 1
c. 5
[✓ d. 6]
e. infinite

When the code above is executed, what will the output be?

a. 0
[✓ b. 15]
c. 0 1 2 3 4 5
d. 0 1 3 6 10 15

Question 4.

```
String input=" ";
char stop = 'Y';

while (stop != 'N')
{
  System.out.println("Hi");
  System.out.print("Continue? (Y or N)");
  input=keyboard.nextLine();
  stop = input.charAt(0);
}
```

How many times will the loop above execute?

a. Not at all
b. Once
c. At least once, possibly more
d. Definitely more than once
e. Infinitely

Question 5.

```
String input=" ";
char stop = 'N';

do
{
  System.out.println("Hi");
  System.out.print("Continue? (Y or N)");
  input=keyboard.nextLine();
  stop = input.charAt(0);
} while (stop != 'N');
```

How many times will the loop above execute?

a. Not at all
b. Once
c. At least once, possibly more
d. Definitely more than once
e. Infinitely

Question 6.

```
for (int count=0; count <= 10; count++)
{
  System.out.println("Hi");
}
```

How many times will the loop above execute?

a. Not at all
b. Ten times
[✓ c. Eleven times]
d. Infinitely

Question 7.

```
String input=" ";
char stop = 'N';

while (stop != 'N')
{
  System.out.println("Hi");
  System.out.print("Continue? (Y or N)");
  input=keyboard.nextLine();
  stop = input.charAt(0);
}
```

How many times will the loop above execute?

[✓ a. Not at all]
b. Once
c. At least once, possibly more
d. Definitely more than once
e. Infinitely

Question 8.

```
char  response;
Scanner keyboard = new Scanner(System.in);
      :
System.out.print ("Would you like extra credit? (Y or N) ");
response = keyboard.nextLine().charAt(0);
```

Given the code above, which of the following would be the appropriate code to do input validation on the user's response? (it should allow for both capital or lowercase letters)

```
a. while (response == 'Y' && response == 'y' && response == 'N' && response ==
   'n')
   {
      System.out.println("Error: you must enter a Y or an N.");
      System.out.print ("Would you like extra credit? (Y or N) ");
      response = keyboard.nextLine().charAt(0);
   }

b. while (response == 'Y' || response == 'y' || response == 'N' || response ==
   'n')
   {
      System.out.println("Error: you must enter a Y or an N.");
      System.out.print ("Would you like extra credit? (Y or N) ");
      response = keyboard.nextLine().charAt(0);
   }

c. while (response != 'Y' && response != 'y' && response != 'N' && response !=
   'n')
   {
      System.out.println("Error: you must enter a Y or an N.");
      System.out.print ("Would you like extra credit? (Y or N) ");
      response = keyboard.nextLine().charAt(0);
   }

d. while (response != 'Y' || response != 'y' || response != 'N' || response !=
   'n')
   {
      System.out.println("Error: you must enter a Y or an N.");
      System.out.print ("Would you like extra credit? (Y or N) ");
      response = keyboard.nextLine().charAt(0);
   }
```

Question 9.

What kind of loop is an input validation loop?

a. Count controlled
b. Event controlled

Question 10.

Which of the following is an appropriate algorithm for a count controlled loop that should loop 25 times, printing the current count to the screen each time?

```
a.  Initialize the counter to 0
    While the counter < = 25
        Print the counter to the screen
        Add 1 to the counter
```

```
b.  Initialize the counter to 1
    While the counter < 25
        Print the counter to the screen
        Add 1 to the counter
```

```
c.  Initialize the counter to 0
    While the counter < 25
        Print the counter to the screen
```

```
d.  Initialize the counter to 1
    While the counter < = 25
        Print the counter to the screen
        Add 1 to the counter
```

CS1 – **Classes/Objects** Checkpoint

Note: *the questions in this checkpoint will only be presented to the user one at a time because of the nature and progression of some of the questions.*
(One per page, one page at a time, without the ability to return to previously answered questions)

Use the Employee class given as a handout when necessary to answer the following questions.

Question 1.

What is the purpose of a "copy constructor"?

a. To create an object of the class with the same values as some other object of that class
b. To create an object of the class that will set class attributes to default values of 0, null, or blanks according to the appropriate variable type
c. To create an object of the class but assign no values leaving the driver program to call the set methods to set the appropriate values

Question 2.

What would the parameter(s) for a copy constructor need to be if you were to create one for the Employee class?

a. A name, an employee id, a pay rate, the number of hours worked
b. An employee object
c. No parameters would be needed

Question 3.

What is the purpose of an "equals method"?

a. To determine if two objects of the same class are equivalent
b. To determine if two objects of different classes are equivalent
c. To determine if multiple parameters passed through the method's parameter list are equivalent to the objects attributes
d. To set two objects to be equal to each other

Question 4.

What would the parameter(s) for an equals method need to be if you were to create one for the Employee class?

a. A name, an employee id, a pay rate, the number of hours worked
b. An employee object
c. No parameters would be needed

Question 5.

What return type should an equals method have?

a. void (nothing will be returned)
b. A String stating whether or not the two objects are equivalent
c. A boolean indicating whether or not the two objects are equivalent
d. An object of the same type (in this case an Employee object)

Question 6.

What is the purpose of a toString method?

a. To convert the object to a String object
b. To format and display the contents of the object to the command line window
c. To format and display the contents of the object to a message box
d. To prepare and return a String containing the contents of the object

Question 7.

Which of the following would be an appropriate no-arg constructor for the Employee class?

```
public Employee()
     {
          setName(empName);
          setId(empId);
          setPayRate(empPayRate);
          setHoursWorked(0);
     }

public Employee()
     {
          setName();
          setId();
          setPayRate();
          setHoursWorked();
     }

public Employee()
     {

     }

public Employee()
     {
          setName(" ");
          setId(" ");
          setPayRate(0);
          setHoursWorked(0);
     }
```

Question 8.

Which of the following would be the correct way to create an object of the Employee class whose name is Jane Doe, payrate is 12.00 an hour and id is 123456?

a. Employee janeDoe = new Employee();
b. Employee janeDoe = new Employee(Jane Doe, 12.00, 123456);
c. Employee janeDoe = new Employee("Jane Doe", 123456, 12.00);
d. Employee janeDoe = new Employee("Jane Doe","123456",12.00);

Question 9.

Which of the following would be the correct way to record the fact that Jane Doe worked 30 hours this week?

a. Employee.setHoursWorked = 30;
b. janeDoe.setHoursWorked = 30;
c. Employee.setHoursWorked(30);
d. janeDoe.setHoursWorked(30);

Question 10.

Which of the following would be the correct way to display how much Jane Doe will get paid this week?

a. System.out.println(calcWages());
b. System.out.println(janeDoe.calcWages());
c. System.out.println(Employee.calcWages());
d. System.out.println(janeDoe.calcWages(hours, payRate));
e. System.out.println(Employee.calcWages(hours, payRate));

Employee class for Classes/Objects checkpoint

```
public class Employee
{
        private String name;
        private String id;
        private double payRate;
        private double hoursWorked;

        public Employee(String empName, String empId, double empPayRate)
        {
                setName(empName);
                setId(empId);
                setPayRate(empPayRate);
                setHoursWorked(0);
        }

        public void setName(String empName)
        {
                name = empName;
        }

        public void setId(String empId)
        {
                id = empId;
        }

        public void setPayRate(double empPayRate)
        {
                payRate = empPayRate;
        }

        public void setHoursWorked(double empHoursWorked)
        {
                hoursWorked = empHoursWorked;
        }

        public String getName()
        {
                return name;
        }

        public String getId()
        {
                return id;
        }

        public double getPayRate()
        {
                return payRate;
        }

        public double getHoursWorked()
        {
                return hoursWorked;
        }

        public double calcWages()
        {
                return hoursWorked * payRate;
        }

}//End employee class
```

Appendix B

Course Materials for CS1 at ETSU

CSCI-1250 (CS1) Learning Outcomes

At the conclusion of the course, a student will be able to:

- Explain the software development life cycle: requirements analysis and specification, design, implementation, testing, and maintenance (Student Outcome 5a*)
- Develop an object-oriented design (Student Outcomes 1c*, 4b*, 5c*)
- Program in Java, an object-oriented programming language (Student Outcomes CS-2*, IS-1b*)
- Describe the qualities of good programming style and use good programming style in his or her programs (Student Outcome 1c, 4b, 5c, CS-2, IS-1b)
- Understand and discuss ethical and professional issues in the use of computers and the impact of computers on society (Student Outcome 2a).

CSCI-1250 (CS1) Course Calendar

CSCI-1250 Course Schedule

Updated: 8/12/12

		Location	Reading for Class	Topics / Unit	Quiz/ Graded Lab	In-Class Activity	Project Assigned	Project Due
Week 1				* If you are not in class the day of a quiz or lab exercise, you will not be able to receive credit for it *				
1	27-Aug	490		Course Introduction		Prelim Quiz/Activity		
2	29-Aug	110	Ch. 1	Introduction to Computers, Problem Solving				
Week 2								
3	3-Sep	490		**Labor Day Holiday**				
4	5-Sep	110	Ch. 2	Java Fundamentals				
Week 3								
5	10-Sep	490		Java Fundamentals	Q1 (Ch1)	LabEx		
6	12-Sep	110		Java Fundamentals			P1	
Week 4								
7	17-Sep	490		Java Fundamentals	Q2 (Ch 2)	LabEx		
8	19-Sep	110	Ch. 4.1-4.10	Decision Structures				
Week 5								
9	24-Sep	490		Decision Structures	Q3 (Ch 2/4)	LabEx		
10	26-Sep	110		Decision Structures, Methods			P2	**P1 - Due**
Week 6								
11	1-Oct	490		Decision Structures, Methods	Q4 (Ch 4)	LabEx		
12	3-Oct	110		Methods	**Q5 (Comprehensive Makeup)**			
Week 7								
13	8-Oct	490		Methods	**Test 1 Ch. 1,2,4 *(Selection Checkpoint)**			
14	10-Oct	110	Ch. 3	Classes/Objects/Methods				**P2 - Due**
Week 8								
15	15-Oct	490		**Fall Break**				
16	17-Oct	110		Classes/Objects/Methods			P3	
Week 9								
17	22-Oct	490		Classes/Objects/Methods	Q6 (Methods)	LabEx		
18	24-Oct	110		Classes/Objects/Methods				
Week 10								
19	29-Oct	490		Classes/Objects/Methods	Q7 (Ch 3)	LabEx		
20	31-Oct	110	Ch. 5.1-5.9	Repetition			P4	**P3 - Due**
Week 11								
21	5-Nov	490		Repetition	Q8 (Ch 3)	LabEx		
22	7-Nov	110		Repetition				
Week 12								
23	12-Nov	490		Repetition	Q9 (Ch 5)	LabEx		
24	14-Nov	110	Ch. 6.1-6.8, 6.10	More Classes/Objects/Methods			P5	**P4 - Due**
Week 13								
25	19-Nov	490		More Classes/Objects/Methods	Q10 (Ch 5/6) ***(Repetition Checkpoint)**	LabEx		**P5 - UML Due**
26	21-Nov	110		More Classes/Objects/Methods	**Q11 (Comprehensive Makeup)**			
Week 14								
27	26-Nov	490		More Classes/Objects/Methods	**Test 2 Ch. 5,3,6**			
28	28-Nov	110		More Classes/Objects/Methods				**P5 - Class Due**
Week 15								
29	3-Dec	490		Review		LabEx		
30	5-Dec	110		Review				**P5 - Driver Due**
Week 16								
31	10-Dec	110		**Final Exam (Comprehensive) Monday, 12/10 8am-10am**	***(Objects/Classes Checkpoint)**			

CSCI-1250 (CS1)
Problem Solving and Programming Handout

Problem Solving and Programming

We have to solve problems every day…some small, some large, some simple and some complex. Sometimes we even use technology to help us solve our problems. However, we can think and solve problems on our own, but computers can't. I know, SHOCKER isn't it?!?! Seriously, a computer cannot think, therefore it cannot solve problems. So, how is it that you can plug in 3 numbers and it can tell you the average with the click of a button??? The reason is that a programmer has written some instructions in a computer language that tell the computer how to solve the problem. So, in case you haven't noticed, the programmer is the key here to solving problems!!! No pressure, right?! ☺

As a future programmer (very near future, as in a week or so from now) you must realize that in order to tell a computer how to solve a problem, you must first know how to solve the problem yourself. Seriously! If you don't know how to solve the problem, you will never in a million years be able to tell the computer how to do it.

Do you know what this means?!?! It means that programming really isn't all about programming (i.e., writing code)…it is mostly about problem solving and then being able to write your solution in a language that a computer can actually understand.

Let's look at a problem…

Problem: I want to find the average of 3 numbers

Solution: Add the numbers together and divide by 2.

That was easy, right?!

Wait just a minute…let's look at that more closely. Do you have any questions about the solution? Does it look like it will work? I have a few questions…What numbers am I supposed to add together? Where did they come from? Do I get to see the answer? Is it calculated correctly?

When we solve a problem, we want to write out the steps as detailed as possible. In the programming world, we call this an **algorithm**. In the real world, you might call this a solution, plan of action, list of steps, etc. The thing about a solution in the real world is that even if you leave out minute details, it's easy enough to still know what to do. Some of the details can be inferred. However, that is definitely not the case with a computer. Every little detail must be included. Things that you do without thinking, like walking down a step…you would have to tell a computer (or robot in this case) every little detail about how to do it. So, as a programmer you will need to learn to pay attention to detail and to include as much detail in a solution to a problem as possible.

Once you come up with a solution, do you assume it works? That may be a safe assumption, but would you want to ride a bike down a gigantic hill assuming that the brakes will work when you need them??? I don't think so. We must test our solutions to make sure they will work before putting them into action. And we don't just want to test it once and if it worked assume that it will always work. We want to test it out in every circumstance imaginable to make sure that it will work EVERY time without fail.

Going back to our problem, **let's test our solution** (after we add a little more detail to address the question of where did the numbers come from).

> **Problem**: I want to find the average of 3 numbers
>
> **Revised solution**: Ask for 3 numbers, Add the numbers together and divide by 2, display the result.
>
> Let's say that the 3 numbers entered were 0, 0, 0. The average should be 0. Add them up, divide by 2 and you get 0. It looks like it works, right?!
>
> Let's test it again just to be sure. Let's say that this time we enter 3, 4, 5. The answer should be 4. Add them up and divide by 2 and you get 6. That doesn't sound right!
>
> What went wrong? Let's try one more. Let's enter 100, 100, 100 this time. The answer should be 100. Add them up and divide by 2 and you get 150. How can you get an average of 150 if the highest number you entered was 100???

So, this testing has let us know that something is wrong with our solution. Now we just need to see what. If we go back and look closely, we will discover that our error is in the math. To get the average you don't divide by 2, you divide by the number of items you are averaging.

> **Problem**: I want to find the average of 3 numbers
>
> **Newly revised solution**: Ask for 3 numbers, Add the numbers together and divide by 3, display the result.

Let's test each of our numbers from before…

> 0, 0, 0 → add and divide by 3 → answer is 0 (as it should be)
>
> 3, 4, 5 → add and divide by 3 → answer is 4 (as it should be)
>
> 100, 100, 100 → add and divide by 3 → answer is 100 (as it should be)

Congratulations, you now have a working solution to the problem!

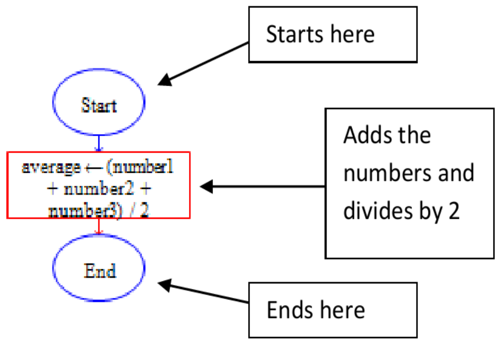
Now, some of you are what we would probably call visual learners, meaning that you don't learn things by reading lots of text or looking at examples. You learn better by seeing things visually and watching things happen. So, when we solve problems this semester, we are going to use some software to help us be able to "see" what is happening as we test our solutions. This may help us to identify problems more easily. We are going to use some software called RAPTOR that uses flowcharts to show solutions to a problem. The cool thing is that it is able to animate your solutions and allow you to test them! We tested our other solution on paper. This time we can test it interactively!

Let's look at our problem again…

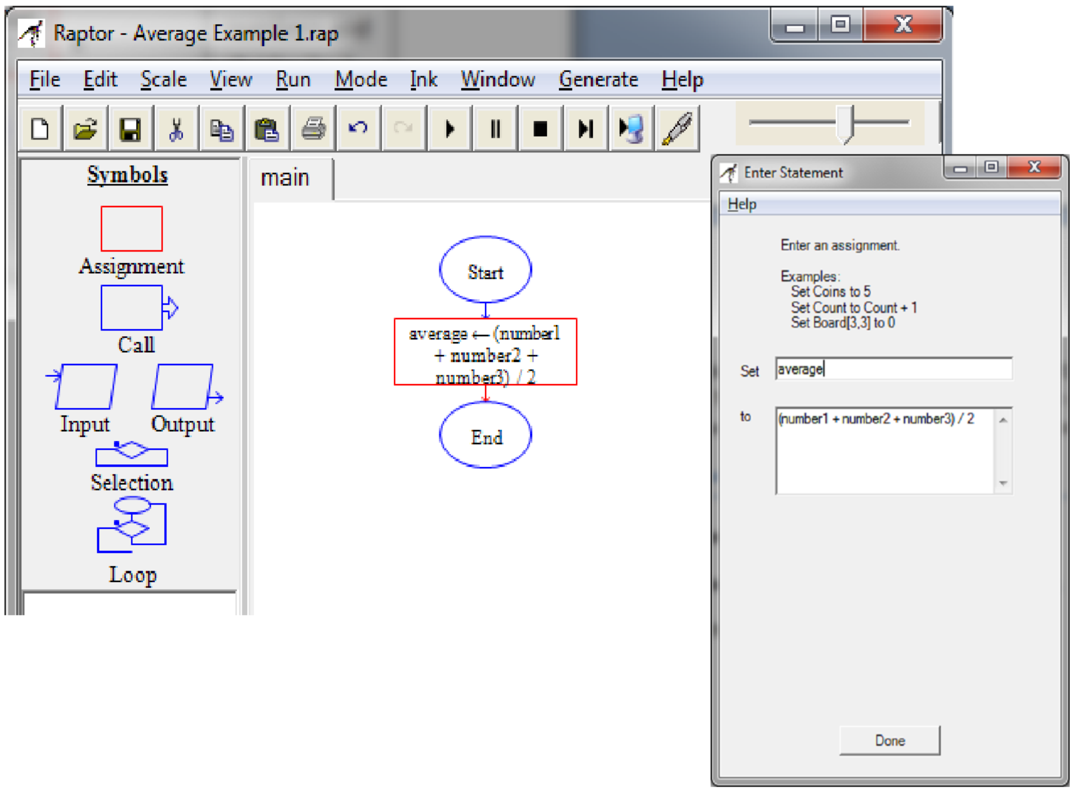
Problem: I want to find the average of 3 numbers

Solution: Add the numbers together and divide by 2.

We can show the solution visually like this…


Starts here
Start
average ← (number1 + number2 + number3) / 2
Adds the numbers and divides by 2
End
Ends here

Using RAPTOR, I am able to use the flowcharting tools to implement my solution. It prompts me for information and then creates the **flowchart**. A flowchart is a visual way to show the steps in the solution to a problem.

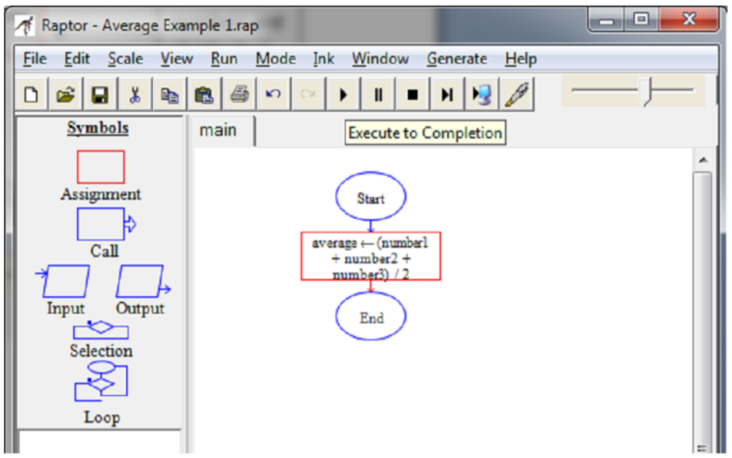

Raptor - Average Example 1.rap
File Edit Scale View Run Mode Ink Window Generate Help
Symbols
Assignment
Call
Input
Output
Selection
Loop
main
Start
average ← (number1 + number2 + number3) / 2
End
Enter Statement
Help
Done

Enter an assignment.

Examples:
Set Coins to 5
Set Count to Count + 1
Set Board[3,3] to 0

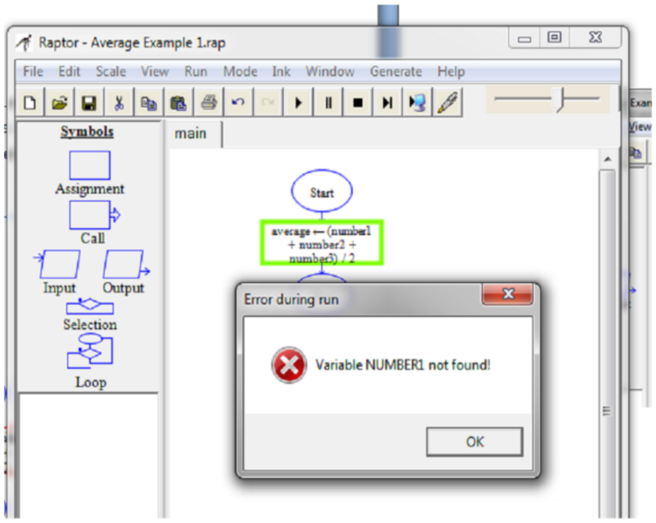
Set average

to (number1 + number2 + number3) / 2


Raptor - Average Example 1.rap
File Edit Scale View Run Mode Ink Window Generate Help
Symbols
Assignment
Call
Input
Output
Selection
Loop
main
Execute to Completion
Start
End

average ← (number1 + number2 + number3) / 2

I can now click the button to "execute" my solution.

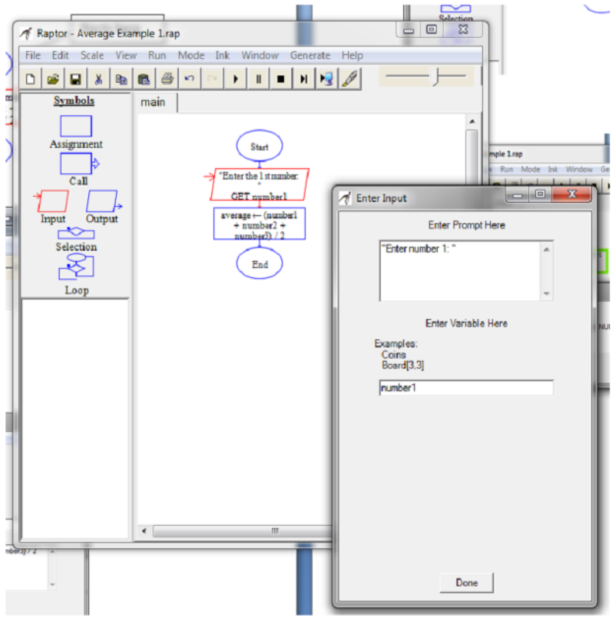

Raptor - Average Example 1.rap
File Edit Scale View Run Mode Ink Window Generate Help
Symbols
Assignment
Call
Input
Output
Selection
Loop
main
Start
Error during run
OK

average ← (number1 + number2 + number3) / 2

Variable NUMBER1 not found!

When I execute it, it highlights the steps in green as it encounters each one. If there is an error in the steps it will give me an error message. In this case it says it doesn't know what number1 is. That's because I don't have the numbers yet, remember?

So, RAPTOR helped us to see what was happening and also pointed out our first error in our solution.



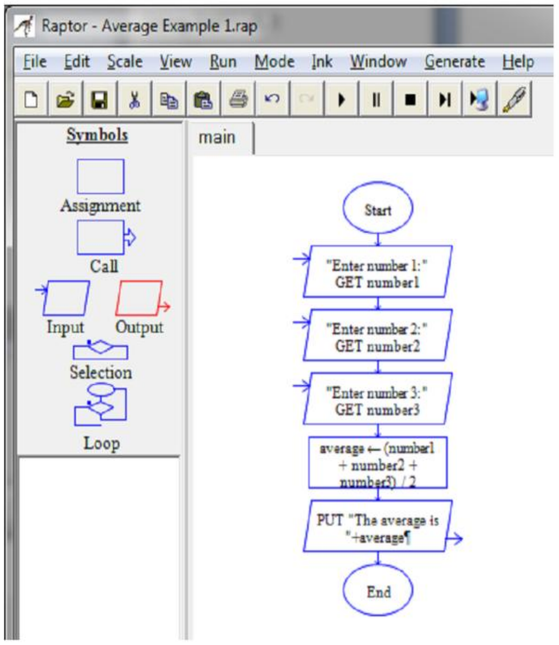
"Enter the 1st number:"
GET number1

average ← (number1 + number2 + number3) / 2

"Enter number 1: "

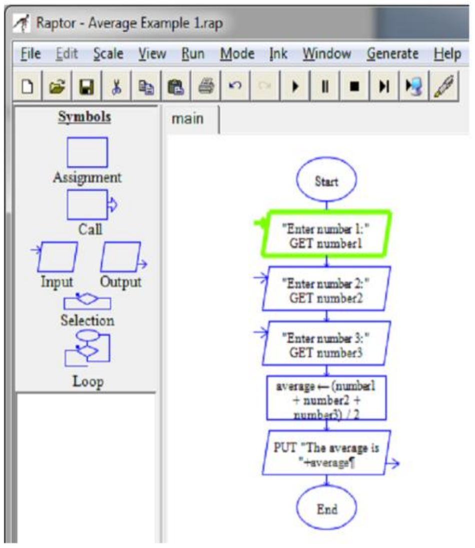
number1

Now I can add in the step to get the numbers, one at a time. It asks me for a prompt (what to ask the user) and then for a variable name to store the number they enter in (a variable is the name of a place to temporarily hold some information). I will do this for all 3 numbers that I'm going to ask the user for. I will also add the step of displaying the average after it has been calculated.


Raptor - Average Example 1.rap
File Edit Scale View Run Mode Ink Window Generate Help
Symbols
Assignment
Call
Input
Output
Selection
Loop
main

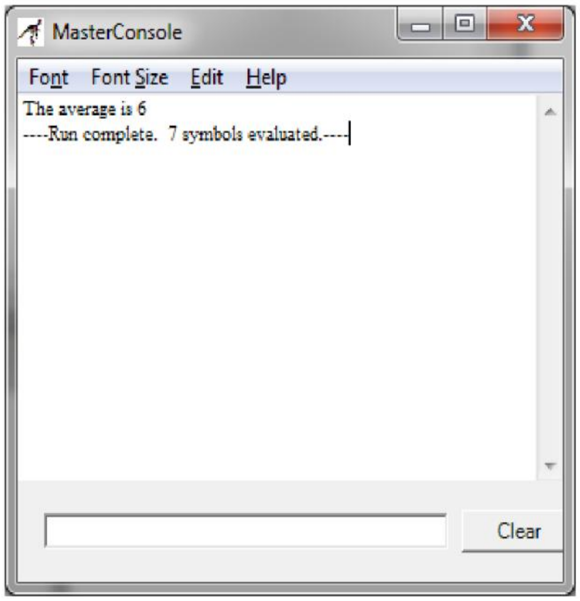
```
Start
"Enter number 1:"
GET number1
"Enter number 2:"
GET number2
"Enter number 3:"
GET number3
average ← (number1 + number2 + number3) / 2
PUT "The average is "+average¶
End
```

This is what my revised solution now looks like.

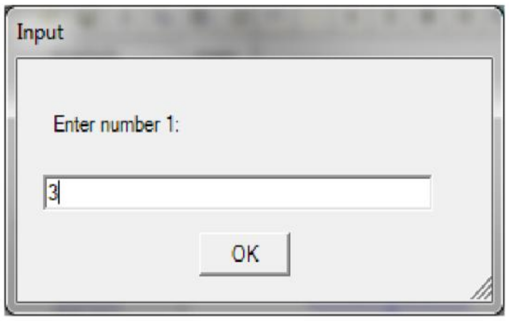

Raptor - Average Example 1.rap
File Edit Scale View Run Mode Ink Window Generate Help
Symbols
Assignment
Call
Input
Output
Selection
Loop
main

```
Start
"Enter number 1:"
GET number1
"Enter number 2:"
GET number2
"Enter number 3:"
GET number3
average ← (number1 + number2 + number3) / 2
PUT "The average is "+average¶
End
```

I can now click ▶ to step through the solution again and test it. This time it will prompt me for each of the numbers using a message box as shown and then it will show me the results in the output window.



```
The average is 6
----Run complete.  7 symbols evaluated.----
```



Enter number 1:

3

This semester we will be using algorithms (solutions written out in a step-by-step format in English), flowcharts (the visual representation of your algorithm) and other visual tools such as UML diagrams to help you solve your programming problems and prepare your solutions BEFORE you ever write any code! As you will soon see, the more time you spend solving your problem before you start coding, the quicker you will be able to complete your code. I promise!

Appendix C

IRB Approvals

East Tennessee State University
Office for the Protection of Human Research Subjects • Box 70565 • Johnson City, Tennessee 37614-1707
Phone: (423) 439-6053 Fax: (423) 439-6060

IRB APPROVAL – Initial Expedited Review

September 7, 2012

Kellie Price

Re: Using Visual Technologies in the Introductory Programming Courses for Computer Science Majors
IRB#: c0812.1s
ORSPA #:

The following items were reviewed and approved by an expedited process:

- form 103, narrative (8/16/2012), Potential Conflict of Interest Form, Student Assurance Statement, Student Questionnaire (CSCI-1250), CS1 Checkpoints, final exam, CV, ICD* Ver. 8/20/12

The item(s) with an asterisk(*) above noted changes requested by the expedited reviewers.

On **September 7, 2012**, a final approval was granted for a period not to exceed 12 months and will expire on **September 6, 2013**. The expedited approval of the study *and* requested changes will be reported to the convened board on the next agenda.

The following **enclosed stamped, approved Informed Consent Documents** have been stamped with the approval and expiration date and these documents must be copied and provided to each participant prior to participant enrollment:

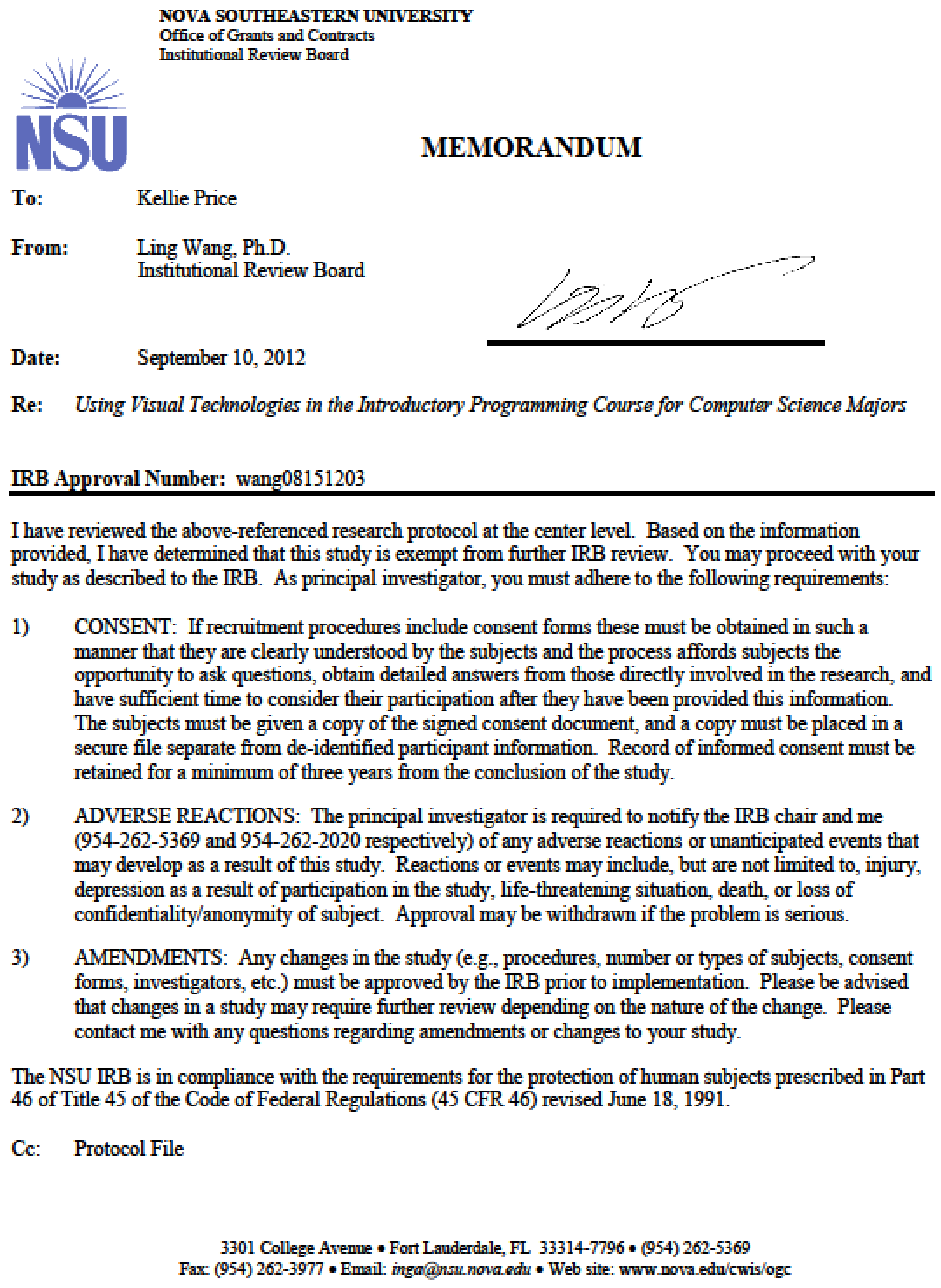
- Informed Consent Document (9/6/12)

Federal regulations require that the original copy of the participant's consent be maintained in the principal investigator's files and that a copy is given to the subject at the time of consent.

Projects involving Mountain States Health Alliance must also be approved by MSHA following IRB approval prior to initiating the study.

Unanticipated Problems Involving Risks to Subjects or Others must be reported to the IRB (and VA R&D if applicable) within 10 working days.

Accredited Since December 2005



NOVA SOUTHEASTERN UNIVERSITY
Office of Grants and Contracts
Institutional Review Board

NSU

MEMORANDUM

To: Kellie Price

From: Ling Wang, Ph.D.
Institutional Review Board

[illegible]

Date: September 10, 2012

Re: *Using Visual Technologies in the Introductory Programming Course for Computer Science Majors*

IRB Approval Number: wang08151203

I have reviewed the above-referenced research protocol at the center level. Based on the information provided, I have determined that this study is exempt from further IRB review. You may proceed with your study as described to the IRB. As principal investigator, you must adhere to the following requirements:

1) CONSENT: If recruitment procedures include consent forms these must be obtained in such a manner that they are clearly understood by the subjects and the process affords subjects the opportunity to ask questions, obtain detailed answers from those directly involved in the research, and have sufficient time to consider their participation after they have been provided this information. The subjects must be given a copy of the signed consent document, and a copy must be placed in a secure file separate from de-identified participant information. Record of informed consent must be retained for a minimum of three years from the conclusion of the study.

2) ADVERSE REACTIONS: The principal investigator is required to notify the IRB chair and me (954-262-5369 and 954-262-2020 respectively) of any adverse reactions or unanticipated events that may develop as a result of this study. Reactions or events may include, but are not limited to, injury, depression as a result of participation in the study, life-threatening situation, death, or loss of confidentiality/anonymity of subject. Approval may be withdrawn if the problem is serious.

3) AMENDMENTS: Any changes in the study (e.g., procedures, number or types of subjects, consent forms, investigators, etc.) must be approved by the IRB prior to implementation. Please be advised that changes in a study may require further review depending on the nature of the change. Please contact me with any questions regarding amendments or changes to your study.

The NSU IRB is in compliance with the requirements for the protection of human subjects prescribed in Part 46 of Title 45 of the Code of Federal Regulations (45 CFR 46) revised June 18, 1991.

Cc: Protocol File

3301 College Avenue • Fort Lauderdale, FL 33314-7796 • (954) 262-5369
Fax: (954) 262-3977 • Email: *inga@nsu.nova.edu* • Web site: www.nova.edu/cwis/ogc

References

Adams, J. (2007). *Alice in action: computing through animation*. Boston, MA: Course Technology.

Ali, A. (2009). A conception model for learning to program in introductory programming courses. *Issues in Informing Science and Information Technology, 6*, 517-529.

Anewalt, K. (2007). Making CS0 fun: An active learning approach using toys, games and Alice. *Journal of Computing Sciences in Colleges, 23(3)*, 98-105.

Barnes, D., & Kölling, M. (2012). *Objects first with Java: A practical introduction using BlueJ* (5th ed.). Upper Saddle River, NJ: Prentice Hall.

Beaubouef, T., & Mason, J. (2005, June). Why the high attrition rate for computer science students: some thoughts and observations. *Inroads – SIGCSE Bulletin, 37(2)*, 103-106.

Becerra-Fernandez, I., Elam, J., & Clemmons, S. (2010, February). Reversing the landslide in computer-related degree programs. *Communications of the ACM, 53(2)*, 127-133.

Bhuta, S. (2007). *First encounter with Java including Bluej*. Maharashtra. India: Shroff Publishers & Distributors Pvt. Ltd.

Biggers, M., Brauer, A. & Yilmaz, T. (2008). Student perceptions of computer science: a retention student comparing graduating seniors vs. cs leavers. *Proceedings of the 39th SIGCSE technical symposium on computer science education*, 402-406.

Blake, J. (2011). Language considerations in the first year CS curriculum . *Journal of Computing Sciences in Colleges, 26(6)*, 124-129.

Boyer, K., Dwight, R., Miller, C., Raubenheimer, C.D., Stallman, M., & Vouk, M. (2007). A case for smaller class size with integrated lab for introductory computer science. *Proceedings of the 38th SIGCSE technical symposium on computer science education*, 341-345.

Brown, P. (2008). Some field experience with Alice. *Journal of Computing Sciences in Colleges, 24(2)*, 213-219.

Brown, W. (2012, June). Introduction to programming with RAPTOR. Retrieved June 1, 2012, from http://raptor.martincarlisle.com/

Browne, M., Lowe, S., Wells, S., & Berry, M.W. (2006). An assessment of computer science animations: a case study. *Journal of Computing Sciences in Colleges, 22(2)*, 162-168.

Bruce, C., Buckingham, L., Hynd, J., McMahon, C., Roggenkamp, M., & Stoodly, I. (2004). Ways of experiencing the act of learning to program: a phenomenographic study of introductory programming students at university. *Journal of Information Technology Education, 3*, 143-160.

Burgess, G., & Hanshaw, C. (2006, February). Application of learning styles and approaches in computing sciences classes. *Journal of Computing Sciences in Colleges ,21(3)*, 60-68.

Cardellini, L. (2002). An interview with Richard M. Felder. *Journal of Science Education, 3(2)*, 62-65.

Carlisle, M. (2009). RAPTOR: A visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges, 24(4)*, 275-281.

Carlisle, M., Wilson, T., Humphries, J., & Hadfield, S. (2004, April). RAPTOR: introducing programming to non-majors with flowcharts. *Journal of Computing Sciences in Colleges, 19(4)*, 52-60.

Chamillard, A.T., & Karolick, D. (1999). Using learning style data in an introductory computer science course. *Proceedings of the 30th SIGCSE technical symposium on computer science education*, 291-295.

Chamillard, A.T., & Sward, R.E. (2005). Learning styles across the curriculum. P*roceedings of the 10th annual conference on innovation and technology in computer science education, 241*-245.

Chen, L., & Lin, J.M. (2011). Learning styles and student performance in java programming courses. *Proceedings of the 2011 international conference on frontiers in education: CS and CE*, 53-58.

Chen, S., & Morris, S. (2005). Iconic programming for flowcharts, java, turing, etc. *Proceedings of the 10th annual SIGCSE conference on innovation and technology in computer science education*, 104-107.

Cliburn, D. (2006). A CS0 course for the liberal arts. *Proceedings of the 37th SIGCSE technical symposium on computer science education*, 77-81.

Dann, W., Cooper, S. & Pausch, R. (2011). *Learning to program with Alice* (3rd ed.). Upper Saddle River, NJ: Pearson Prentice Hall.

Davies, S., Polack-Wahl, J.A., & Anewalt, K. (2011). A snapshot of current practices in teaching the introductory programming sequence. *Proceedings of the 42nd ACM technical symposium on computer science education*, 625-629.

Dierbach, C., Taylor, B., Zhou, H., & Zimand, I. (2005). Experiences with a CS0 course targeted for CS1 success. *Proceedings of the 36th SIGCSE technical symposium on computer science education*, 317-320.

Dillon, E., Anderson, M., & Brown, M. (2012). Comparing feature assistance between programming environments and their "effect" on novice programmers. *Journal of Computing Sciences in Colleges, 27(5)*, 69-77.

Drake, P., & Sung, K. (2011). Teaching introductory programming with popular board games. *Proceedings of the 42nd ACM technical symposium on computer science education*, 619-624.

Eckerdal, A., Thuné, M., & Berglund, A. (2005). What does it take to learn 'programming thinking'? *Proceedings of the first international workshop on computing education research*, 135-142.

Forte, A., & Guzdial, M. (2004). Computers for communication, not calculation: media as a motivation and context for learning. *Proceedings of the Hawaii International Conference on System Sciences*.

Frand, J. (2000, September). The information-age mindset: Changes in students and implications for higher education. *EDUCAUSE review*, 15-24.

Gaddis, T. (2011a). *Starting out with Alice: A visual introduction to programming* (2nd ed.). Boston, MA: Addison-Wesley.

Gaddis, T. (2011b). *Starting out with Java: Objects early* (4th ed.). Boston, MA: Addison-Wesley.

Gal-Ezer, J., & Harel, D. (1998, September). What (else) should CS educators know? *Communications of the ACM, 41(9)*, 77-84.

Garlick, R., & Cankaya, E.C. (2010). Using Alice in CS1 – a quantitative experiment. P*roceedings of the 15th annual conference on innovation and technology in computer science education,* 165-168.

Gay, L.R., & Airasian, P.W. (2003). *Educational Research: Competencies for analysis and applications* (7th ed.). Upper Saddle River, NJ: Merrill/Prentice Hall.

Gomes, A., Carmo, L., Bigotte, E., & Mendes, A.J. (2006). Mathematics and programming problem solving. *Proceedings of the 3rd E-Learning conference – computer science education, (*CD-ROM).

Gomes, A., & Mendes, A.J. (2010). A study on student performance in first year CS courses. P*roceedings of the 15th annual conference on innovation and technology in computer science education,* 113-117.

Gomes, A., & Mendes, A.J. (2008). A study on student's characteristics and programming learning. *Proceedings of the ED-MEDIA '08 – world conference on educational multimedia, hypermedia, & telecommunications,* 2895-2904.

Gomes, A., & Mendes, A.J. (2007). Learning to program – difficulties and solutions. *Proceedings of the ICEE '07 – international conference on engineering education,* (CD-ROM).

Grant, N. (2003). A study on critical thinking, cognitive learning style, and gender in various information science programming classes. *Proceedings of the 4th conference on information technology curriculum*, 96-99.

Gribbons, B., & Herman, J. (1997). True and quisi-experimental designs. *Practical Assessment, Research and Evaluation, 5(14)*. Retrieved October 28, 2011, from http://pareonline.net/getvn.asp?v=5&n=14

Gross, P., & Powers, K. (2005). Evaluating assessments of novice programming environments. *Proceedings of the first international workshop on computing education research*, 99-110.

Gudmundsen, D., Olivieri, L., & Sarawagi, N. (2011, June). Using Visual Logic©: three different approaches in different courses – general education, CS0 and CS1. *Journal of Computing Sciences in Colleges, 26(6)*, 23-29.

Guthrie, R., Yakura, E., & Soe, L. (2011). How did mathematics and accounting get so many women majors? What can IT disciplines learn? *Proceedings of the 2011 conference on Information Technology education*, 15-20.

Herbert, C. (2011). *An introduction to programming using Alice 2.2* (2nd ed.). Boston, MA: Course Technology.

Herrmann, N., Popyack, J., Char, B., Zoski, P., Cera, C., Lass, R., & Nanjappa, A. (2003). Redesigning introductory computer programming using multi-level online modules for a mixed audience. *Proceedings of the 34th SIGCSE technical symposium on computer science education*, 196-200.

Howles, T. (2007). Preliminary results of a longitudinal study of computer science student trends, behaviors and preferences. *Journal of Computing Sciences in Colleges, 22(6)*, 18-26.

Hughes, J., & Peiris, D.R. (2006). ASSISTing CS1 students to learn: learning approaches and object-oriented programming. P*roceedings of the 11th annual conference on innovation and technology in computer science education,* 275-279.

Kölling, M. (2010). *Introduction to programming with Greenfoot: Object-oriented programming in Java with Games and Simulations*. Upper Saddle River, NJ: Prentice Hall.

Kouznetsova, S. (2007, April). Using BlueJ and Blackjack to teach object-oriented design concepts in CS1. *Journal of Computing Sciences in Colleges, 22(4)*, 49-55.

Kuri, N.P., & Truzzi, O.M.S. (2002). Learning styles of freshman engineering students. *Proceedings of the international conference on engineering education,* 18-22.

Lewis, J., & DePasquale, P. (2008). *Programming with Alice and Java*. Boston, MA: Addison-Wesley.

Manaris, B. (2007). Dropping CS enrollments: Or the emperor's new clothes? *Inroads – SIGCSE Bulletin, 39(4)*, 6-10.

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program. *Computer Science Education, 16(3)*, 211-227.

McCauley, R. & Manaris, B. (2002, May). Comprehensive Report on the 2001 Survey of Departments Offering CAC-Accredited Degree Programs. Retrieved May 14, 2011, from http://www.cs.cofc.edu/~mccauley/survey

Miller, L.D., Soh, L., Samal, A., Nugent, G., Kupzyk, K., & Masmaliyeva, L. (2011). Evaluating the use of learning objects in CS1. *Proceedings of the 42nd ACM technical symposium on computer science education*, 57-62.

Mitchell, W. (2001). Another look at CS0. *Journal of Computing Sciences in Colleges, 17(1)*, 194-205.

Molenda, M. (2003). ADDIE Model. In *Educational Technology: An Encyclopedia*. Santa Barbara, CA: ABC-Clio.

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *Proceedings of the 35th SIGCSE technical symposium on computer science education*, 75-79.

Mullins, P., Whitfield, D., & Conlon, M. (2009, January). Using Alice 2.0 as a first language. *Journal of Computing Sciences in Colleges, 24(3)*, 194-205.

Naps, T., Robling, G., Anderson, J., Cooper, S., Dann, W., Fleischer, R.,…Ross, R. (2003, December). Evaluating the educational impact of visualization. *ACM SIGCSE Bulletin, 35(4)*, 124-136.

Nesbit, T. (2009). The teaching of introductory programming: issues of context. *Proceedings of the 22nd national advisory committee on computing qualifications*, 71-78.

Oblinger, D. (2003, July). Boomers, Gen-xers & Millennials: Understanding the new students. *EDUCAUSE review,* 37-47.

Parker, S. (2002). *McGraw-Hill dictionary of scientific and technical terms* (6th ed.). New York, NY: McGraw-Hill.

Pearce, J., & Nakazawa, M. (2008). The funnel that grew our CIS major in the CS desert. *Proceedings of the 39th SIGCSE technical symposium on computer science education*, 503-507.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., & Paterson, J. (2007, December). A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin, 39(4)*, 204-223.

Pillay, N., & Jugoo, V. (2005, December). An investigation into student characteristics affecting novice programming performance. *Inroads – SIGCSE Bulletin, 37(4)*, 107-110.

Riley, D. (2003). *The object of Java: introduction to programming using software engineering principles BlueJ edition*. Boston, MA: Addison-Wesley.

Ruslanov, A., & Yolevich, A. (2010, April). College student views of computer science: opinion survey. *Journal of Computing Sciences in Colleges, 25(4)*, 142-148.

Shelly, G., Cashman, T., & Herbert, C. (2007). *Alice 2.0: introductory concepts and techniques*. Boston, MA: Course Technology.

Sigle, J. (2008). Teaching an introductory programming course in a virtual world. *Proceedings of the 2008 international conference on frontiers in education: computer science*, 236-240.

Sloan, R., & Troy, P. (2008). CS 0.5: a better approach to introductory computer science for majors. *Proceedings of the 39th ACM technical symposium on computer science education*, 271-275.

Soe, L., Guthrie, R., Yakura, E., & Hwang, D. (2011). Designing an introductory CIS course to attract and retain female (and male) students. *2011 ISECON Proceedings*, *28(1642)*, 1-10.

Sprankle, M., & Hubbard, J. (2009). *Problem solving & programming concepts* (8th ed.). Upper Saddle River, NJ: Pearson Prentice Hall.

Stamey, J., & Sheel, S. (2010). A boot camp approach to learning programming in a CS0 course. *Journal of Computing Sciences in Colleges, 25(5)*, 34-40.

Stolee, K.T., & Fristoe, T. (2011). Expressing computer science concepts through Kodu Game Lab. *Proceedings of the 42nd ACM technical symposium on computer science education*, 99-104.

Stone, J.A., & Clark, T.A. (2011). The impact of problem-oriented animated learning modules in a CS1-style course. *Proceedings of the 42nd ACM technical symposium on computer science education*, 51-56.

Talton, J., Peterson, D., Kamin, S., Israel, D., & Al-Muhtadi, J. (2006). Scavenger hunt: computer science retention through orientation. *Proceedings of the 37th SIGCSE technical symposium on computer science education*, 443-447.

Thomas, L., Ratcliffe, M., Woodbury, J., & Jarman, E. (2002). Learning styles and performance in the introductory programming sequence. *Proceedings of the 33rd SIGCSE technical symposium on computer science education*, 33-37.

Trochim, W.M.K, & Donnelly, J.P. (2008). *The Research Methods Knowledge Base* (3rd ed.). Mason, OH: Cengage Learning.

Urness, T., & Manley, E. (2011, May). Building a thriving CS program at a small liberal arts college. *Journal of Computing Sciences in Colleges, 26(5)*, 268-274.

Vegso, J. (2008). Enrollments and degree production at US CS departments drop further in 2006-07. *Computing Research News, 20(2).*

Venit, S., & Drake, E. (2011). *Prelude to programming: concepts and design.* (5th ed.). Boston, MA: Addison-Wesley.

Wellman, B.L., Davis, J., & Anderson, M. (2009). Alice and robotics in introductory CS courses. *Proceedings of the Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations*, 98-102.

Yadin, A. (2011, December). Reducing the dropout rate in an introductory programming course. *ACM Inroads, 2(4)*, 71-76.

Yim, K., Garcia, D.D., & Ahn, S. (2010). Computer Science illustrated: Engaging visual aids for Computer Science education. *Proceedings of the 41st ACM technical symposium on computer science education*, 465-469.

Yoo, J., Yoo, S., Seo, S., & Pettey, C. (2011). Can AlgoTutor change attitudes toward algorithms? *Proceedings of the 42nd ACM technical symposium on computer science education*, 311-316.

Zweben, S. (2008). Computing degree and enrollment trends from the 2007-2008 CRA Taulbee Survey. *CRA.org*. Retrieved May 20, 2011, from http://archive.cra.org/taulbee/CRATaulbeeReport-StudentEnrollment-07-08.pdf

Zweben, S. (2011). Undergraduate CS degree production rises from the 2009-2010 CRA Taulbee Survey. *CRA.org*. Retrieved December 9, 2011, from http://www.cra.org/uploads/documents/resources/taulbee/CRA_Taulbee_2009-2010_Results.pdf