

Introduction

On average, at least one half of college students majoring in computer science withdraw from the field. [32] While attrition occurs throughout the 4 years, the majority of students drop out during the first year. [26] Thus, there is only a small window in which changes can be made that will be effective in reducing attrition. Though attrition is a tricky phenomenon influenced by multiple factors, it is necessary to understand its causes before effective changes can be made.

In the 1970's and 80's, researchers investigated the causes for this attrition with the goal of developing measures that could be used to predict student success or failure. The idea was that there were certain characteristics a student could have that increase the probability of success in a computer science curriculum. Attracting students with those characteristics would therefore be a key goal. The characteristics identified include an ability to think algorithmically, to apply problem-solving techniques, and to perform logical reasoning. [7, 30, 22]

Work in the 1990's focused on the attrition rate of minorities, especially of women. Suggested reasons for attrition included a lack of experience with computers [27], mathematical ability/preparation [8], self-efficacy [6], socialization [26], and the university environment [9]. Based on the recommendations of these researchers, many colleges and universities enacted programs to provide a better social and support environment.

Of late, there has been a shift in viewpoint in studies of attrition rates. Recent studies [13, 17] have shown students without prior programming experience are at a decided disadvantage in being able to complete a computer science degree. This disagrees with the work of many earlier researchers who observed little to no correlation between prior programming experience and success in a computer science program. We believe that this shift in explanation may be related to the change by most computer science departments from imperative languages (such as C and Pascal) to object-oriented (OO) languages (such as C++ and Java). OO languages require the

teaching and learning of all of the material for an imperative language (assignment, decisions, functions, procedures, repetition, arrays, etc.). They then add the concepts of class, object, information hiding, inheritance, and polymorphism. (Though polymorphism is not uniquely an OO concept, most imperative languages do not support it.) Many computer science programs also teach event-driven programming which adds another programming paradigm students must master. As the time spent in college classrooms has not increased, students with no prior programming experience are likely to be overwhelmed by the breadth and depth of material. We speculate this leads to the observations by [13, 17]. The volume of concepts and information now required in CS1 (introductory computer science) exacerbates an already troublesome situation.

In conclusion, there appear to be two major issues in determining success in the first year of a computer science program. The first is the set of skills (problem solving, logical reasoning, etc.) needed to succeed, and the second is the quantity of programming concepts that students must learn. The challenge is to determine how to concentrate on developing the fundamental skills while covering a more diverse range of material.

Visualization: Part of the difficulty in teaching the necessary problem solving and logical reasoning skills is that students are simultaneously learning the syntax of a programming language, the design of algorithms, and the connection between the two. Soloway [28] notes that the real difficulty for novice programmers lies in “putting the pieces together”, i.e. in figuring out what constructs to use and how to coordinate those constructs. Beginning programmers not only need to learn how to design an algorithm for solving a problem but also how to translate the steps of the algorithm into specific programming constructs. But, making the connection between an algorithmic step and a specific programming construct requires some concept of how the computer executes that programming construct at runtime. We have observed that many students have difficulty visualizing the steps of the execution of a program, or the current state of the

program at any given time. As a result, students have trouble figuring out what went wrong when things do not work. A source of confusion in building and debugging programs may be an inadequate understanding of program state during program execution.

The use of graphics and visualization is recognized as an effective teaching tool in computer science to help students put the pieces together [21]. Several visualization tools have been developed and used over the last 20 years. Papert's LOGO [23] uses turtle graphics. Papert discovered two problems for beginning programmers in using LOGO: variables and debugging. Many programming language packages now include a debugging utility to show the program state (the values of the variables) any time during program execution. But, novice programmers tend to struggle with debuggers. They do not know where to set breakpoints or which variables to examine, particularly when objects are involved.

GROK [12], designed by one of the PIs, is a data visualization tool to help students more easily view their data transformation during program execution. GROK is only designed to work for Pascal programs and does not handle sophisticated data structures. Karel, a robot world simulation, was used in the early 1980's to introduce structured programming concepts, similar to Pascal [24]. In the mid-1990's, Karel was adapted for use as an introduction to OO concepts, similar to those used in C++ [3]. But Karel++ introduced a more complicated code structure, and was more difficult to program (being less intuitive for students). Specifically, Karel++ moved from using one robot (and the actions it could perform) to a robot model. Inheritance makes the robot models far more complex, as multiple robots can have multiple capabilities.

Algorithm animation tools, such as XTANGO [29] and BALSAM [4], were developed to incorporate visualization into the learning process for algorithm analysis courses. The instructor programs the algorithm animation and the student runs the program while observing the animation of the algorithm using different inputs. Thus, algorithm animation packages generally

do not help the students in the visualization of *their own* programs. Students need a methodology for testing animation programs that *they* have developed.

In summary, visualization seems to be a key tool to help students resolve many problem solving and reasoning difficulties. We believe teaching fundamental concepts in a *preliminary* course using visualization in an OO environment, will provide sufficient "prior programming experience" (as recommended by [13, 17]). However, none of the current tools provide program visualization for the novice within the confines of an OO world in a way that we need.

Goals and Objectives

The purpose of this project is to create a textbook and develop curriculum materials for teaching and learning fundamental programming concepts using simulation and visualization in a 3-dimensional (3-D), interactive, OO, animation environment. We plan to use an animation tool called Alice [31] (see page 6) to provide that environment. The approach used in this proof of concept will take advantage of a high-level of interest in graphics, animation and storytelling (commonly found among students who have grown up in a multimedia world). However, the major emphasis is the use of visualization to teach and learn a strong core of fundamental programming concepts and problem-solving techniques in an OO, interactive environment.

The goals of this proof of concept project are to:

1. Produce a complete draft of a textbook.
2. Develop curricular materials, including laboratory exercises and a reference for using the software. The curriculum materials can be used to supplement the material in a traditional computer science introductory programming course or can be used together with the textbook in a separate course that precedes the traditional CS1.
3. Apply and integrate the text and curricular materials at two institutions.
4. Assess the effectiveness of goals 2 and 3.

Timeline: In the spring and summer of 2002, we will work on developing a complete draft of the textbook. We have a foundation of (rough-draft) chapters that have been developed with an early version of the software. A new version of the animation software (JAlice) is scheduled for

release in fall 2001. The new version will be substantially different from the current version (Alice). Our objective is to redesign and reconstruct the current materials to work with the new edition of the software. A second objective is to develop additional topics and projects.

In the summer of 2002, we will also pursue our second goal: developing curriculum materials to supplement the text, assisting the instructor. The objective is to develop additional PowerPoint lecture slides, labs, exercises, a teacher's manual and a separate software reference.

The fall semester of 2002 will provide an opportunity to pilot test the complete text and curricular materials in our courses. Courses scheduled for the fall 2002 semester are *Alice Animation* at Ithaca College (IC) and *Building Virtual Worlds: An Introduction to Computing Concepts* at Saint Joseph's University (SJU). An objective is to establish a solid framework for assessment of our approach. A second objective is to examine characteristics such as class performance and gender with the students' experiences and the students' perception of benefits using 3-D animation/visualization in learning fundamental programming concepts.

Evaluation of the courses and of students' attrition during the first year will occur in both the spring and summer of 2003. Please see the evaluation section for more details.

Project Description

We propose to develop curricular materials for use in teaching fundamental programming concepts using a 3-D, interactive, OO, animated tool. The goal is to better prepare students for success as a computer science major. We will use an animation environment to help students visualize program execution. As opposed to traditional, text-oriented programming languages, a 3-D virtual world has the advantage that most of the program state (like object position and color) is intrinsic in a "natural" way. Students **see** the program state and how it changes over time. Also, 3-D worlds are more realistic than their 2-D counterparts and seem to encourage exploration in creating variants of existing worlds and making novel worlds of their own.

Current situation: One of the PIs for this proof of concept project is currently teaching at IC and the other PI is at SJU. In both institutions, the CS1 courses follow the ACM Curriculum 91 guidelines. Both courses make use of weekly in-class (closed) laboratory exercises, as well as open laboratories where students can work with teaching assistants to obtain extra help.

At SJU, there are approximately 30 new computer science majors each year. The attrition rate is slightly over 1/3 for the first year. Students with little or no programming background but with adequate mathematical background (defined as those students who are ready for calculus I) have an attrition rate that is slightly higher. However, students with little or no programming background and insufficient math background have close to 100% attrition.

At IC, there are approximately 35-40 computer science majors among the entering freshman class each year. The attrition rate for computer science majors during the first year is about 30 percent. Of the students who change their major during the first year, approximately 90 percent have had no previous programming experience and about 40 percent begin their freshman year at the pre-calculus level in mathematics.

3-D Animation Technology: The software to be used in this work is Alice [31]. Alice is a freely available 3-D Interactive Graphics Programming Environment for Windows, developed at Carnegie Mellon University (CMU) under the direction of Randy Pausch. (See [25], [10], or [11] for more details.) A new version, available in fall 2001 will run on any platform. The goal of the Alice project is to make it easy for novices to develop interesting 3-D animations and to explore the new medium of interactive 3-D graphics.

3-D models of objects (e.g., animals and vehicles) populate a virtual world in Alice. Alice has a strong OO flavor. By writing simple scripts, Alice users can control object appearance and behavior via mouse and keyboard input. Alice serves as a good programming environment for the novice. Students are immediately able to see how their animated programs run, affording an

easy relationship of the program construct to the animation action. Figure 1 displays a typical virtual world in Alice, with three objects: a rabbit, a butterfly, and a butterfly net.



Figure 1. A 3-D Virtual World in Alice

LOGO and Karel, previously discussed, have been wonderful tools and have gained widespread acceptance for use with novice programmers. However, we believe that Alice provides certain advantages for today's students:

- ◆ Alice is more OO than functional. This will lead to a smoother transition of programming concepts to CS1 where students learn to program in an OO language.
- ◆ For the true novice, minimal programming or keyboarding is required. Alice provides a “smart” editor, so the user can simply select which actions the animated object will perform. It is not possible to make syntax errors, as no typing is required! Instead of seeing many syntax errors caught by the compiler, the novice sees an error when a particular object performs the “wrong” action. It is clear what went wrong. Entire programs may be written simply by clicking and dragging with the mouse, indicating which objects are to be animated and how, and specifying the sequence of the animation.
- ◆ Alice has a high interest level. Certainly, this has been true with LOGO and Karel, but Alice provides 3-D objects and realistic animation. We have found students to be extremely enthusiastic about the animated worlds they can easily create. Students invest an extraordinary amount of time and effort into building their virtual worlds! Anecdotally, we have seen their additional effort leading to higher achievement.
- ◆ The added complexity of control structures really gives added functionality. For example, the use of recursion allows students to create animated worlds that contain "chase scenes". This is a key component to being able to create a type of computer game.
- ◆ It is quite easy to write programs in Alice. A single statement can be tested individually or several statements can be tested as a small collection of statements. It is not necessary to write an entire program first before trying it out.

Creation of a textbook: There are currently no textbooks that support using Alice as an animation tool to teach novices programming concepts. The Alice web site contains a tutorial on how to use Alice, but this is not sufficient for educational use. We have a rough draft of a text [2], based on our previous experiences with teaching high school students in special summer programs and college students in a trial course at IC. (See the next section for more details.) Significant revision is necessary, as is the addition of a number of important topics. While three publishers have already indicated an interest in our text, much rework and additional development are required. Specifically, the following changes are necessary:

- “Translation” of the entire text to use the new version of Alice.
- Addition of an introductory chapter on building interactive worlds. And, incorporation of interaction sections to many of the early chapters.
- Creation of several experimental project chapters that illustrate the content principles described in the earlier chapters.
- Addition of numerous examples and problems.
- Adding a user’s guide to the end of the text.
- Addition of “advanced” chapters, covering such topics as collision detection, and dynamic creation and destruction of objects. While these topics are not essential from a pedagogic perspective, they are topics in which students have expressed an interest.

When completed, the text will contain chapters of content (to develop the conceptual ideas and illustrate the problem-solving techniques) and chapters of experiments/projects (to apply these concepts in particular animated scenarios and simulations). A brief discussion of the topics for the text is included here. See Appendix A for a detailed list. The text begins with an introduction to the Alice environment, explaining virtual worlds and the 3-D nature of objects. Then, object interaction in a virtual world is explained (movement and interaction with other objects), and how objects are composed of parts. (For example, a rabbit has sub-parts head, torso, tail, arms, and legs. Many sub-parts themselves have sub-parts.) In this way, students are introduced to the OO nature of a virtual world environment. After introducing objects, the creation of scripts or programs is illustrated. (It is important to note that the smart editor allows the novice programmer to use mouse click-and-drag to build programs.) Next, students learn to write procedures, essentially combining sequences of motion instructions into a named command.

After presenting the basics, more sophisticated animations are described and illustrated. Students learn about decision statements, as well as repetition in animations. Of course, this leads to a discussion of writing functions. And, a form of recursion becomes an invaluable tool.

Interaction (or “event driven-ness”) will be presented throughout the text. Later chapters will focus on interactive animations for web pages. However, the text will be designed so that it is possible to skip those sections without impact for the instructor who does not wish to cover this programming paradigm. Finally, there will be a chapter (or possibly two) on advanced animation concepts such as collision detection. The text will conclude with an Alice user’s guide.

It is important to note that the text does not introduce/use mutable variables. This avoids the problem (of students trying to use variables) observed by Papert when his students learned LOGO. The only “variables” are the objects themselves and students see these objects as they interact in their virtual world.

Creation of curricular materials: Current materials include a few PowerPoint slides for lectures and some laboratory exercises. See Appendix B for a sample laboratory exercise. Existing slides must be modified based on errors, omissions, and the new version of the Alice software. Our aim is to prepare a minimum of two laboratory exercises for each topic. Currently, most topics have either one or no laboratory exercise. Several end-of-section problems will be written to expand the current minimal set. An Alice user’s guide will be created. The user’s guide will also provide help on the most common errors students and teachers will encounter, and how to correct them. And, the initial set up of the Alice environment will be explained in much greater detail.

The current status: Some materials have been developed and used in special summer sessions and one trial semester course. Based on anecdotal evidence collected during these sessions, the computer science departments at IC and SJU have agreed to offer courses for learning computer programming using 3-D animation, scheduled for Fall 2002, as stated previously.

Integration at SJU: We plan to use the materials and text for inclusion in a one semester 4-credit course. We anticipate that students taking this class will be first-year computer science students with no previous programming experience and a weak mathematics background. Also, some non-majors who are interested in learning about computer programming, but who are not ready for the traditional CS1 class, will take the class. The summer before they start college, targeted computer science students will be sent a letter, recommending that they enroll for this class before taking CS1. It is expected that the materials we have developed will make up approximately the first 2/3 of the class. We will use JKarel for the last 1/3 of the course. JKarel is a version of Karel the robot written in Java. (Since the students will be using Java in CS1, JKarel will allow them to see the syntax of Java.) Computer science students will be expected to enroll in CS1 the following semester. We hope some non-majors will enroll in CS1 as well.

Integration at IC: We plan to use the materials and text in a course offered during the first half of the fall 2002 semester. Anticipated enrollment includes first year computer science majors with little or no previous programming experience and non-majors who want to take a course involving animation. Computer science majors who take the course will simultaneously enroll in CS1. The instructors of this course and the CS1 course will collaborate to ensure that the concepts being studied in this course will support and enhance the concepts learned in CS1.

Testing with human subjects: As students will be used to help evaluate the materials (see the evaluation section for more detail), all appropriate human subject procedures will be followed throughout the investigation. This includes receiving approval from Internal Review Boards (of human subjects) at both schools, obtaining all necessary written consents, and the maintenance of strict confidentiality throughout the study.

Deliverables

The deliverables will consist of instructional materials, a textbook, and a report of the results of implementing the materials in two different scenarios at IC and at SJU. The instructional materials will consist of lecture notes, lab exercises covering all of the topics (with solutions), and sample programming project assignments. They will be provided online. The textbook will include a CD containing a copy of the Alice software, and all of the worked examples from the text. The textbook will be submitted to a publisher. The PIs have already been in contact with MIT Press, O'Reilly, and Wiley.

Future plans

If this proof of concept is successful, a full CCLI proposal will be submitted to NSF. We are in touch with faculty at other schools (US and abroad) who are interested in the materials. We expect that they will use the materials in other ways and are interested in studying the effectiveness. Another possible avenue is involvement with high schools teachers to modify materials for use in high school introductory programming classes.

Other future plans include using the materials we have developed as a means of encouraging women and African-Americans to pursue careers in computer science. There are strong connections between female learning and arts and graphics [20, 14] and between storytelling and African-American learning/culture [16, 18, 5]. It will certainly be possible to focus on the arts/graphics aspects of the 3-D animation software or on the storytelling aspects.

Impact

The problem of attrition from the computer science major, particularly during the first year, is a serious one. We hope to lower the rate, especially among the most vulnerable students (those with poor mathematics and/or programming background). We also hope that, through animation, students will become more immersed in their computer studies and will increase their overall effort. One other benefit may be the attraction of non-majors into the computer science major.

Experience/capability of PIs

We believe that we are uniquely qualified to run this investigation. Dr. Dann has twelve years experience in teaching introductory computer science. Her expertise spans applied research in program visualization, curriculum development, and workshop presentations for event-driven programming. Dr. Cooper has been teaching introductory computer science for ten years. He also has 10 years of industry experience, thus having good knowledge about what industry “needs” in the way of computer science graduates.

Both PIs have been working with Alice for 3 years. Our experiences with using and teaching computing concepts with Alice enable us to complete this project. We are in close contact with the developers of Alice at CMU and make biannual visits to meet with the development team to discuss the design of Alice and how to better tailor it to the needs of novice programmers. The team has been quite willing to implement our suggestions.

We already have a draft of a book [2], with publisher interest. The CS education community is quite interested in our Alice work, as they have accepted several of our papers for conferences. (See the PIs’ biographical sketches for a complete list.)

Dr. Moskal, an expert in educational assessment, will oversee the implementation of the assessment process throughout this project. Dr. Moskal has extensive experience in assessment and evaluation and has worked as an assessment consultant on two other NSF-funded computer science curriculum development projects. Dr. Lurie, who has extensive statistical analysis experience, will perform all data analysis. (See biographical sketches.)

Evaluation and Assessment

Formative evaluations will be used to improve project-related activities. Summative evaluations will be used to determine whether project related objectives are being reached.

Formative: Qualitative research techniques will be used for formative evaluation purposes. Qualitative techniques have the advantage of providing detailed descriptive information and this type of information is useful for project improvement purposes.

Peer Review of Curriculum Materials. This project will result in the development of a textbook and curricular materials. At least three external experts in the fields of computer science and computer science education will review all materials and revisions will be made based on their recommendations. One of these experts is expected to be Randy Pausch, developer of the Alice tool. Peer review is a widely accepted technique for examining the content, construct and criterion validity of instructional materials [1].

Student Interviews. The instructional materials will be pilot-tested in two classrooms in the fall of 2002, at IC and at SJU. At the end of the semester, five randomly selected students in each pilot classroom will be interviewed. This interview will focus upon their experiences with the instructional materials and how these experiences influenced their future educational goals and decisions. The individual interviews will allow the investigators to acquire a detailed understanding of students' experiences with the instructional materials.

Student Focus Groups. Students not selected for individual interviews will participate in student focus group activities. Questions asked in the focus groups will be the same as those asked during the individual interviews. The focus group activity will ensure every student has an opportunity to express his/her viewpoint.

Summative: The summative evaluation will primarily use quantitative research techniques to support the statistical analysis of our efforts. At both locations, control classrooms will also be selected. The control classrooms will consist of first year computer science majors who are completing their first college computer science course. In pilot and control classrooms, a survey will be given to determine the students' background in mathematics, computers and computer

science. The data collected by the readiness survey will be used to examine relationships between math and computer science preparedness, student performances and rates of attrition.

Attitudes Survey. At the beginning and end of the semester, students in both the pilot and control classrooms will also complete a computer science attitude survey. This survey will help to determine whether students' participation in their first computer science course has further stimulated their interest in pursuing a degree in computer science. Two surveys have been identified as potentially being appropriate to our purposes ([15] and [19]). As part of this project, a more extensive search will be completed to locate the most current and appropriate instrument.

Statistical comparisons will be made between the pilot and the control classroom, providing evidence as to the influence of the newly designed course on students' computer science interests. Special attention will be given to the relationship of the students' entering level of mathematics and computer science readiness and their continued interest in a computer science degree. Comparisons will also be made between female and minority students that are in the pilot and control classrooms. We do not expect these numbers to be large enough for statistical inference. But we will use descriptive statistics as a method to compare these subgroups. Though the information acquired from these comparisons cannot be generalized beyond the current population, it will be used in the development of a full proposal.

Pretest/Posttest: Through a collaborative effort between the investigators and the assessment consultant, a pretest/posttest assessment instrument will be developed. The researchers will identify fundamental concepts in computer science that are typically taught in introductory computer science courses. The resultant instrument will be included in the expert review that was described in the Formative section of this proposal. Statistical comparisons will be made between the pilot and the control classrooms' performances on this instrument.

Again, attention will be given to students' entering readiness and their performance on the pre and post instrument, and comparisons will be made between female and minority students in the pilot and control classrooms. The gender and ethnicity comparisons will be descriptive in nature and can only be used to make speculations for the development of the full proposal.

Retention Statistics: Statistical comparisons will also be made between the number of students in the pilot and control classrooms that select to continue to pursue a computer science degree. This data will suggest whether the newly developed course contributes to student retention in computer science. Readiness will be examined to determine whether it was a factor in outcomes of these statistics. Descriptive comparisons between female and minority students in the pilot and control classrooms will also be made. As previously stated, these comparisons can only be used to make speculations that will guide the development of the full proposal.

Dissemination of Results

Dissemination activities will begin immediately through a project web site. This web site will eventually contain all of the curricular materials we will develop. Presentations will be made at local, national, and international conferences (e.g., CCSCNE, SIGCSE, ITiCSE). We are expecting at least two conference papers -- one focussing on our methodology and the other on our results. Intra-school workshops will be held to familiarize other computer science faculty members with our work. Papers will also be submitted to computer science educational journals, such as Computer Science Education and JERIC. We expect at least one journal article, arguing for more pre-CS1 preparatory material and examining different ways of using our materials. A textbook (with a CD containing Alice) will be produced, and (publisher permitting) an on-line version made available. Instructional materials including laboratory exercises will also be made available on the project web site.