OOP: the rest of the story

Article *in* Journal of Computing Sciences in Colleges · October 2011

CITATIONS
0

READS
2,177

1 author:

Chuck Allison
Utah Valley University
57 PUBLICATIONS **10** CITATIONS

SEE PROFILE

The Journal of Computing Sciences in Colleges

Volume 27, Number 2 December 2011

The Journal of Computing Sciences in Colleges

Papers of the Twentieth Annual CCSC Rocky Mountain Conference

October 14-15, 2011
Utah Valley University
Orem, Utah

Papers of the Twenty-fifth Annual CCSC Southeastern Conference

November 11-12, 2011
Furman University
Greenville, South Carolina

Volume 27, Number 2 **December 2011**

The Journal of Computing Sciences in Colleges (ISSN 1937-4771 print, 1937-4763 digital) is published at least six times per year and constitutes the refereed papers of regional conferences sponsored by the Consortium for Computing Sciences in Colleges. Printed in the USA. POSTMASTER: Send address changes to Paul Wiedemeier, CCSC Membership Chair, Assistant Professor of Computer Science, The University of Louisiana at Monroe, Computer Science and CIS Department, 700 University Avenue, Administration Building, Room 2-37, Monroe, LA 71209.

TABLE OF CONTENTS

THE CONSORTIUM FOR COMPUTING SCIENCES IN COLLEGES BOARD OF DIRECTORS

Following is a listing of the contact information for the members of the Board of Directors and the Officers of the Consortium for Computing Sciences in Colleges (along with the year of expiration of their terms), as well as members serving the Board:

Bob Neufeld, President (2012), Professor Emeritus of Computer Science, McPherson College, P. O. Box 421, North Newton, KS 67117, 316-283-1187 (H), neufeld@mcpherson.edu.

Laura J. Baker, Vice-President (2012), Professor, Computer Sciences Department, St. Edward's University, Box 910, 3001 S. Congress Ave, Austin, TX 78704, 512-448-8675, laurab@stedwards.edu.

Paul Wiedemeier, Membership Chair (2013), Assistant Professor of Computer Science, The University of Louisiana at Monroe, Computer Science and CIS Department, 700 University Avenue, Administration Building, Room 2-37, Monroe, LA 71209, 318-342-1856 (O), 318-342-1101 (fax), ccscpaul@gmail.com.

Bill Myers, Treasurer (2014), Dept. of Computer Studies, Belmont Abbey College, 100 Belmont-Mount Holly Road, Belmont, NC 28012-1802, 704-461-6823 (O), 704-461-5051, (fax), myers@crusader.bac.edu.

John Meinke, Publications Chair (2012), Collegiate Associate Professor, UMUC Europe, US Post: CMR 420, Box 3368, APO AE 09063; (H) Werderstr 8, D-68723 Oftersheim, Germany, 011-49-6202-5 77 79 16 (H), meinkej@acm.org.

Colleen Lewis, Southwestern Representative (2014), University of California at Berkeley, 4421 Gilbert St. Apt 113, Oakland, CA 94720, 510-388-7215, colleenl@berkeley.edu

Elizabeth S. Adams, Eastern Representative (2014), Associate Professor Emeritus, James Madison University - Mail Stop 4103, CISAT - Department of Computer Science, Harrisonburg, VA 22807, 540-568-2745 (fax), adamses@jmu.edu.

Jeff Lehman, Midwestern Representative (2014), Professor of Computer Science, Mathematics and Computer Science Department, Huntington University, 2303 College Avenue, Huntington, IN 46750, 260-359-4209, jlehman@huntington.edu.

Scott Sigman, Central Plains Representative (2014), Associate Professor of Computer Science, Drury University, 1273 E. 345th Road, Flemington, MO 65650, 417-873-6831, ssigman@drury.edu.

Kevin Treu, Southeastern Representative (2012), Furman University, Dept of Computer Science, Greenville, SC 29613, 864-294-3220 (O), kevin.treu@furman.edu.

Timothy J. McGuire, South Central Representative (2012), Sam Houston State University, Department of Computer Science, Huntsville, Texas 77341-2090, 936-294-1571, mcguire@shsu.edu.

Brent Wilson, Northwestern Representative (2012), George Fox University, 414 N. Meridian St, Newberg, OR 97132, 503-554-2722 (O), 503-554-3884 (fax), bwilson@georgefox.edu.

Pat Ormond, Rocky Mountain Representative (2013), Professor, Information Systems and Technology, Utah Valley University, 800 West University Parkway, Orem, UT 84058, 801-863-8328 (O), 801-225-9454 (cell), ormondpa@uvu.edu.

Lawrence D'Antonio, Northeastern Representative (2013), Ramapo College of New Jersey, Computer Science Dept., Mahwah, NJ 07430, 201-684-7714, ldant@ramapo.edu.

Linda Sherrell, MidSouth Representative (2013), Associate Professor, Computer Science, University of Memphis, 209 Dunn Hall, Memphis, TN 38152, 901-678-5465 (O), linda.sherrell@memphis.edu.

Serving the Board: The following CCSC members are serving in positions as indicated that support the Board:

Will Mitchell, Conference Coordinator, 1455 S Greenview Ct, Shelbyville, IN 46176-9248, 317- 392-3038 (H), willmitchell@acm.org.

George Benjamin, Associate Editor, Muhlenberg College, Mathematical Sciences Dept., Allentown, PA 18104, 484-664-3357 (O), 610-433-8899 (H), benjamin@muhlenberg.edu.

Susan Dean, Associate Editor, Collegiate Professor, UMUC Europe, US Post: CMR 420, Box 3369, APO AE 09063; Werderstr 8, D-68723 Oftersheim, Germany. 011-49-6202-5 77 82 14 (H), sdean@faculty.ed.umuc.edu.

Robert Bryant, Comptroller, Professor & Information Tech. Program Director, MSC 2615, Gonzaga University, Spokane, WA 99258, 509-313-3906, bryant@gonzaga.edu.

Mark Goadrich, National Partners Chair, Broyles Eminent Scholars Chair of Computational Mathematics, Assistant Professor of Computer Science, Centenary College of Louisiana, 2901 Centenary Boulevard, Shreveport, LA 71104, 318-869-5194, mgoadric@centenary.edu.

Brent Wilson, Database Administrator, George Fox University, 414 N. Meridian St, Newberg, OR 97132, 503-554-2722 (O), 503-554-3884 (fax), bwilson@georgefox.edu.

Myles McNally, Webmaster, Professor of Computer Science, Alma College, 614 W. Superior St., Alma, MI48801, 989-463-7163 (O), 989-463-7079 (fax), mcnally@alma.edu.

CCSC NATIONAL PARTNERS

The Consortium is very happy to have the following as National Partners. If you have the opportunity please thank them for their support of computing in teaching institutions. As National Partners they are invited to participate in our regional conferences. Visit with their representatives there.

National Science Foundation

Panasonic Solutions Company

Turing's Craft

FOREWORD

It is a pleasure to present the papers of two excellent conferences. I commend both conferences to you, the reader and the participant. While the ideal is to participate in the conferences themselves, the *Journal* offers those of us unable to attend the opportunity to share in the wealth that is presented at the conferences.

I would like to take a moment here to mention our reviewing process. Each conference individually accomplishes its own reviewing which results in a submission being either accepted, rejected, or recommended acceptance with modifications. It is a double blind peer reviewing process which means that the reviewer does not know who submitted the paper nor does the submitter know who reviewed the submission. Each submission is typically reviewed by at least three reviewers. This process maintains the quality of the presentations at the individual conferences. The welcome statement at the beginning of each conference gives more specific information on the reviewing results for that conference.

Let me also remind you that the contents of the *Journal* also appear in the ACM Digital Library.

There are many who deserve thanks as we look at any of these conferences and the individual sets of papers. I have so often been concerned about properly thanking those involved with both the production of the conference which results in the *Journal* and those involved in production of the *Journal*. This time I would like to just express general thanks to all those hard workers but would like to also thank both the conference attendees as well as the *Journal* readership. Without the attendees there would be no conference which would in turn result in no *Journal*. I most definitely find the *Journal* a valuable addition to my professional library plus I benefit majorly from attendance at at least one regional conference per year.

John Meinke
University of Maryland University College Europe
CCSC Publications Chair

Papers
of the
Twentieth Annual
Consortium for Computing
Sciences in Colleges
Rocky Mountain Conference

October 14-15, 2011
Utah Valley University
Orem, Utah

WELCOME — 2011 CCSC: ROCKY MOUNTAIN CONFERENCE

Welcome to the 20th annual meeting of the Consortium for Computing Sciences in Colleges: Rocky Mountain Region October 14 - 15, 2011, in beautiful Orem, Utah, at Utah Valley University. I hope you will take some time and look around at the beautiful scenery.

I wish to express my gratitude to the conference steering committee, all of whom completed their assigned duties. Thanks also to our presenters, keynote speaker, and banquet speaker, who made our conference possible. They and I appreciate all the hard work done including the reviews of the submitted papers and workshops. We had 19 abstracts submitted. The reviewers accepted 63% of the submissions. The accepted papers are printed in this issue of the *Journal of Computing Sciences in Colleges*. As you will see, we have papers covering a wide range of significant and timely topics. Also essential to the success of our conference are all of you attendees. Thank you for coming and participating in our sessions. A special thanks to our National Partners: the National Science Foundation, Panasonic Solutions Company, and Turing's Craft. We thank them for supporting the activities of the Consortium.

I hope you enjoy the conference, the attractions and scenery in Orem. Please consider submitting a paper, workshop, or poster session for a future conference or serving as a reviewer, keynote speaker, or banquet speaker. You are invited to become a more active member of our organization by joining the Conference Steering Committee including volunteering to host one of our annual meetings.

Victoria Eisele, Conference Chair
Front Range Community College

2011 STEERING COMMITTEE — RM CCSC CONFERENCE

Pat Ormond, Board Representative. . . . . Utah Valley University, Orem, UT
Victoria Eisele, Conference Chair. Front Range Community College, Westminster, CO
Tim Reeves, Past Conference Chair. . . . . San Juan College, Farmington, NM
Resa Sanati Mehrizy, Site Co-Chair. . . . . Utah Valley University, Orem, UT
Afsaneh Minaie, Site Co-Chair. . . . . Utah Valley University, Orem, UT
Henry Zwick, Registrar. . . . . College of Eastern Utah, Price, UT
Pat Ormond, Treasurer.. . . . . Utah Valley University, Orem, UT
Victoria Eisele, Publicity Chair. . . Front Range Community College, Westminster, CO
Aaron Gordon , Webmaster. . . . . Fort Lewis College, Durango, CO
Jerry Shultz, Program Chair.. . . . . Metro State College of Denver, Denver, CO
Karina Assiter, Papers/Panels/Tutorials Chair.. . . . . .
. . . . . Wentworth Institute of Technology, Boston, MA
Jean Johnson, Proceedings Editor. . . . . Black Hills State University, Spearfish, SD
Resa Sanati Mehrizy, Moderators Co-Chair. . . . . Utah Valley University, Orem, UT
Afsaneh Minaie, Moderators Co-Chair. . . . . Utah Valley University, Orem, UT

REVIEWERS — 2011 CCSC ROCKY MOUNTAIN CONFERENCE

Rajan Alex. . . . . West Texas A&M University, Canyon, TX
Chuck Allison. . . . . Utah Valley University, Orem, UT
Allison Thompson Brown. . . . . University of Colorado at Boulder
Daniel Bryce.. . . . . Utah State University, Logan, UT
Renee Bryce. . . . . Utah State University, Logan, UT
Roger deBry. . . . . Utah Valley University, Orem, UT
Reva Freedman. . . . . Northern Illinois University, DeKalb, IL
Russell Jones. . . . . Arkansas State University, State University, AR
Marcos Pinto. . . . . NYC College of Technology, Brooklyn, NY
Curtis Welborn.. . . . . Utah Valley University, Orem, UT

A LEARNING SPACE FOR BEGINNING PROGRAMMING STUDENTS*

Roger deBry
Department of Computing/Network Sciences
Utah Valley University
Orem, UT
debryro@uvu.edu

ABSTRACT

Traditional programming courses would seem to be active learning courses by their very nature. After all, students must be actively engaged in practicing their craft if they are to be successful. But, does a course that depends on lectures and a handful of programming assignments really engage students in a meaningful way? The concept of a learning space, as proposed by David and Alice Kolb, describes a number of learning concepts that are intended to enhance experiential learning by promoting growth producing experiences for learners. This paper outlines some of the key aspects of a learning space as described by Kolb, and describes the changes made to one section of our beginning programming course to create an active learning space. Results are compared with a similar section taught during previous semesters.

1. INTRODUCTION

In this paper we explore the application of Kolb's concept of Learning Spaces to an introductory programming course. Our discussion begins with a description of the traditional model of a programming course as taught at our institution. Our concern is that a course that depends largely on lectures and a handful of programming assignments may not provide the kind of learning experience that many students require. This conclusion is supported by looking at the number of students who fail this introductory course.

In order to provide a better learning experience, we then look at some of the important properties of a learning space, as proposed by Alice and David Kolb and

---

describe changes made to one section of our beginning programming course to create a learning space that is based on many of these properties. Finally, the results of teaching in this environment are compared with the outcomes of previous semesters.

2. LEARNING SPACES

The word traditional in this context refers to the way that the introductory programming courses have been taught at our institution for many years. The course material is presented using PowerPoint slides and supplemented with programming examples done in class. During a 15 week semester students may have to write ten or twelve programs and take three exams. Grading of assignments is done by student graders who probably took the same course just a few semesters earlier. While many students are successful in this environment, a large number are not. This does not seem to be uncommon in computer science programs across the country [1]. As an example, in the on-line section of our beginning programming course in spring 2009, the grade distribution looked like this:

A/ A-	31%
B+/ B/ B-	5%
C+/ C/ C-	11%
D+/ D/ D-	11%
E	42%

There is obviously room here for improvement! With the notion of a learning space in mind, several small experimental steps have been taken over the past few semesters to improve the way that this beginning programming course is taught.

3. LEARNING SPACES

The notion of a learning space was introduced by Alice and David Kolb [2] as a way of enhancing experiential learning. In a nutshell, experiential learning theory defines learning as the process of learning by doing. Recent studies have suggested that this process of experiential learning is connected with the way that the brain works [3, 4]. Kolb suggests that experiential learning can be enhanced through the use of learning spaces. While the concept of a learning space is fairly new, it seems to embody many principles that are becoming prominent in higher education circles today [5]. These principles are summarized in the following paragraphs.

A Space Where Students Feel Respected

A learning space should provide an environment where students feel like they are valued members of a learning community. Students want and need the respect of their faculty and peers. Kolb calls this a space "*where everybody knows your name*". Many faculty routinely teach hundreds of students every semester, and where this is the case it may not be possible to remember everyone's name. However, many names can be remembered, every student can be treated with respect, and every student's experiences

can be valued. When students feel alone, unrecognized, or undervalued, it often becomes difficult for them to learn.

A Space Where Students Feel Supported

Most students entering their first computer science class are going to feel a bit uncomfortable. They are most certainly going to be outside of their comfort zone that first day of class when they learn that a third of the class has already had some programming experience, but they don't even know what a compiler is. This can be a threatening experience for a new computer science student. It has been said that "*people grow best where they continuously experience an ingenious blend of challenge and support*" [5]. The new computer science student is certainly going to be challenged. That's the easy part. Often the more difficult task for the instructor is to be sure that the student feels supported in their learning efforts. Students need to know that the faculty are there to support them, not flunk them.

A Space That is Open to Conversation

Humans interact through conversation, and it is through conversations where much learning takes place. When students are subjected to long lectures with no opportunity to ask questions or express opinions, learning is stifled. When students are engaged in conversation about a topic they have an opportunity to create knowledge by reflecting on their experiences. Talking and listening are both important elements of conversation, so a teacher must be a good listener. Conversations don't just happen between a teacher and a student. Sometimes the most meaningful conversations take place between students, and these kinds of conversations should be encouraged.

A Space for Developing Expertise

Programming is a discipline that requires a deep understanding of the principles and ideas that are taught in class. Students cannot get by with just a superficial knowledge of programming. They need real knowledge of the whys and the hows of programming so that they can organize and apply that knowledge to the creation of new solutions to difficult problems. Listening to lectures and reading textbooks doesn't begin to build this kind of knowledge. It is not until a student puts pencil to paper (or fingers to keyboard) that understanding begins to happen. The "aha" moments come when a student is struggling to get a program to work, not while listening to a lecture, no matter how entertaining the lecture may be. The more students are challenged to write good code, the more aha moments they will have. Expertise comes from sustained, recursive practice.

A Space Where Students Take Ownership

Many students come into the classroom with the attitude that they are going to be passive recipients of whatever it is that is being taught. They are very happy to sit and be lectured to, hoping that enough will sink in that they can complete their assignments and

get a passing grade. In a programming course this often leads to *trial and error* programming, where students try endless combinations of things that they have seen in class, without a true understanding of how they work or how they fit together. A learning space should motivate students to take ownership for their education. When a student takes control of their learning they are more willing to experiment, ask questions, and practice what they learn.

4. DEVELOPING A LEARNING SPACE

Creating an environment that exhibits the properties that have just been described is not something that happens overnight. It involves a great deal of introspection into one's own teaching philosophies and recognition of the changes that must be made in one's teaching style. Change comes hard! It is most certainly an iterative process, and one that invites constant improvement. For some of us it is a life-long process.

The following sections describe some of the pieces that have been experimentally added to one section of our beginning programming course to create a such a learning space.

4.1 Lab Assignments

In the spring of 2009 our beginning programming students had to complete twelve programming assignments and a handful of labs. Unless a student chose to write code on their own, that was the extent of the practice that they got in programming. If a student understood what went on in class they could finish an assignment in a few hours. If not, they would generally work by trial and error for many, many more hours. Some students reported working for ten or twelve hours on a programming assignment without success.

In the spring of 2010 we experimented by adding additional lab assignments to one section of our beginning programming course. Students were required to do three labs a week, in addition to their regular programming assignments. Most labs included some additional study material on the topic of concern and usually required a small program to be written to examine the ideas covered in the study material. The objective of these labs was to help students understand the topics of concern in more depth, and to give them many more hours of programming practice. Our hope was that students were writing code almost every day. While students generally did better in this section of the course, there were serious complaints from students about the number of hours spent doing homework.

In the summer of 2010, the class was only taught on-line and the number of labs was reduced to two a week. This seemed to be a reasonable trade-off between the time required to do the homework for the course and the value to be gained by writing more programs. Over the course of the semester students were required to complete twenty-six labs and twelve programming projects. In the student evaluations for this class there were no complaints about the amount of time required to do homework. On the contrary, a number of students commented on how much they appreciated the labs because they helped them prepare for the programming assignment later in the week. In his student evaluation of the course one student commented "*Having small, frequent assignments helps keep you engaged. It's a little like practicing the piano.*" Labs covered topics such

as algorithm development, writing pseudo-code, formatting output, validating user input, debugging, and class diagrams.

4.2 The Grading of Assignments

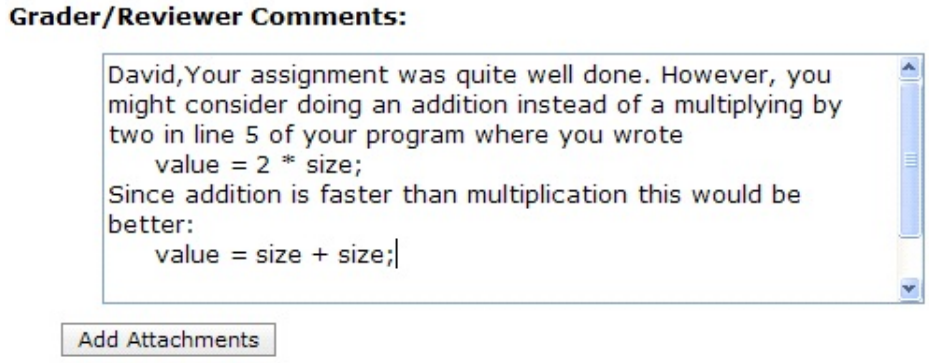
Historically we have used student graders in all of our courses. Most faculty members prefer to use student graders because the beginning course sections are quite large and grading is tedious work, taking time away from other scholarly pursuits. However, in my mind there are a number of concerns with the use of student graders.

First of all, we have no graduate students at our institution, and the job market in our region is such that many 3rd and 4th year students can find jobs in the local software industry. As a result our student graders are often not always the brightest and the best. In many cases they only completed their beginning courses a few semesters ago. While these graders are certainly capable of following the grading rubrics supplied with each assignment, they simply do not have the depth of experience to provide meaningful feedback to students when there are errors in the code, when the code has style issues, or when the code could have been written in a better way.

Another problem that we have suffered with when using student graders is that they are not always dependable. Oftentimes graders have gotten two or three weeks behind in their grading, so any feedback that students do get is so late that it is not useful. In some cases, graders have just stopped doing their work without telling anyone, and it is only when students complain that their instructors discover that there is a problem. When this happens it is difficult to find a replacement mid-semester, and the instructor ends up having to pick up the pieces.

I decided several semesters ago that it was not worth the trouble of using a student grader. I felt that it was important that I grade all of my student's assignments This was a major step towards creating the kind of learning space that I wanted. This simple step has helped to build a space where I can get to know students better because I see their work several times a week. With appropriate words of praise and/or encouragement it helps me give students a space where they feel supported. When grading is done quickly students feel respected, and with meaningful feedback it helps students to reflect on what they are learning. When grading on Blackboard, a student grader might judge that the program that the student submitted works correctly and fill out the graders review/comment box like this:

Now, when I grade, I always try to (1) mention the student by name, give some praise or encouragement, as appropriate, and (3) give suggestions to help the student enhance their programming skills. Compare the following figure to the one above:



4.3 Resubmitting Assignments

In the on-line CS1 course for summer 2010, I took what I felt was another important step in building a learning space for my students. Up to this point in time students would in turn assignments, they would be scored, and that was it. I felt that we were missing an important teaching opportunity by just giving a student a grade and as shown above some feedback on their assignment. To me, finding problems in an assignment and telling the student how they could be fixed was really a teaching moment, but it was only by allowing the student to reflect on the error, fix it, and resubmit the assignment that learning really took place.

So, I announced to my on-line class that I would grade their assignments as soon as I noticed that they had been turned in on Blackboard. I also told them if there were any

problems with their code, I would give them suggestions for fixing their code, return their assignment to them, and let them re-submit their code before the due date for full credit.

Student feedback has been very enthusiastic. In his class assessment one student said "*I Loved how quickly assignments were corrected and really appreciated the opportunity to fix mistakes.*"

4.4 Discussion Board

A final piece of the learning space has been the addition of a discussion board for students. We have actually used this for some time, but it fits nicely into our over-all objective by providing another place for conversations to take place. I have found that many students are a bit shy about engaging in a face to face discussion, but they have no problem when using the discussion board. I suppose that this comes from our student's culture of texting and using social networks. Blackboard does have a discussion board feature that many faculty members on campus use. The problem is that access to a discussion board is limited to students in a particular section of a course. This is certainly okay, but if all of the students in a course had access to the same discussion board, it would be much better.

Several years ago, I experimented with a number of open source forums and discussion boards. I finally picked phpBB [7] as our discussion board software. The discussion board for our CS1 course is open to every student in every section of the course, and we often have students who have moved on to our CS2 course who remain active on the CS1 board, helping newer students. Of course, the great advantage to this is that there are lots and lots of students looking at the board every day and there a lot of valuable conversations going on that would not happen otherwise. I think the biggest benefactors of this are the students in the on-line section of the course, because they don't have the benefit of sitting with their peers in the classroom and starting a conversation.

5. RESULTS

While I have been experimenting with bits and pieces of this idea for some time, it only began to come together during the summer semester of 2010. Based on student feedback alone, the results have been well worth the effort. Of course we not only want happy and satisfied students, we want students who have indeed had a successful learning experience. The following table bears out the fact that the application of the ideas presented has been successful on that account as well.

The table below summarizes the grade distribution for students from three semesters. To provide a consistent measurement, these were all from on-line sections of our beginning programming course. The changes in grade distribution are dramatic.

	Spring 2009	Spring 2010	Summer2010
A, A-	31%	35%	58%
B+, B, B-	5%	15%	19%
C+, C, C-	11%	12%	5%
D+. D, D-	11%	0%	4%

E	42%	38%	14%

6. CONCLUSIONS

In his commencement address in 2003, Harvard president Lawrence Summers related the story of a top science student who had written a letter to him containing the following statement, "I am in the eighth semester of college, and there is not a single science professor here who could identify me by name"[8]. This letter motivated a major review of undergraduate teaching at Harvard. Our goal in this paper has been to convince you that creating an environment where students are known, respected, and supported is important, and that it does make a significant difference in how well students learn. We believe that the results speak for themselves. The changes in our CS1 course have made a measurable and significant difference in student grades.

Clearly, the implementation of most of these ideas requires that the instructor spend a lot of time grading and commenting on student work. This can take time away from other scholarly pursuits and when classes are very large it may not even be feasible. Finally, we have not studied the impact of these ideas when taken one at a time. We suggest further research to clarify and expand on these issues.

REFERENCES

[1] Beauboef, T. and Mason, J. 2005. Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations. SIGSCE Bulletin (Inroads). 103-106

[2] Kolb, Alice Y., and Kolb, David A. 2005. Learning Styles and Learning Spaces: Enhancing Experiential Learning in Higher Education. Academy of Management Learning & Education, Vol. 4, No. 2, 193-212

[3] Zull, J. E. 2002. *The Art of Changing the Brain*. Sterling Virginia. Stylus Publishing, LLC

[4] DeBry, R.K. 2004. Learning Exercises for the Rest of the Brain. Journal of Computing Sciences in Colleges. Vol. 20. Issue 1. 291-296

[5] Bransford, J. D., Brown, A., and Cocking, R.R. 2000. *How People Learn*. Washington, D.C. National Academy Press

[6] Kegan, R. 1994, In Over Our Heads: The Mental Demands of Modern Life. Cambridge MA. Harvard University Press. 42.

[7] http://www.phpbb.com/

[8] Summers, L.H., On undergraduate education, *Harvard Magazine*, 63-65, July-August 2003

UNDERSTANDING A DBMS FROM THE INSIDE OUT*

Bradley Marshall
Silverlode Interactive
Lindon, Utah 84058
801 400-1682
bmarshall@playsaga.com

Curtis Welborn
Department of Computer Science
Utah Valley University
Orem, Utah 84058
801 863-7058
Curtis.Welborn@uvu.edu

ABSTRACT

In the 1970's a new file structure was needed to support the storing of tens of millions of records on a hard drive. This new file structure had to support both fast random access and in-order sequential access to records. Additionally, the new structure had to support transaction-based processing without significantly degrading the performance of the system over time. The file structure that was developed to meet these demands was a B+ Tree. While B+ Trees can be implemented as data structures that reside strictly in memory, they come into their own when they are converted to a file structure and stored on disk. B+ Trees are such an elegant solution to this problem that virtually every Relational Database Management System (DBMS) uses a B+ Tree to store information. As elegant as a B+ Tree is for building a DBMS, much of the internal structure of both the DBMS and the B+ Tree is devoted to minimizing the impact of accessing a slow hard drive when compared to accessing main memory. The relative speed difference between accessing a hard drive vs. accessing main memory can easily reach 100,000 times. This paper will outline our experience in developing a small DBMS from scratch. The DBMS is composed of a B+ Tree file structure for storing indexes and tuples, a system catalog for defining the contents of a tuple, a buffer pool for caching tuples, and a relational algebra engine for executing queries to access data store in the B+ Tree.

.

INTRODUCTION

One of the most important concepts that an individual must be aware of when starting to learn about B+ Trees is that there is conflicting and misleading information available about B+ Trees. It is generally understood that a B+ Tree can be constructed from a B-Tree. This is also where the problem begins as there are two distinctly different data structures that exist; both named B-Tree, yet only one of these structures can be converted into a B+ Tree suitable for implementing a DBMS. This paper will not focus on the history of how this problem has come about; rather, we will focus on how a B+ Tree must be constructed to support development of a DBMS.

In the 1970's a new file structure was needed to support the storing of tens of millions of records on a hard drive. This new file structure had to support both fast random access and in-order sequential access to records. Additionally, the new structure had to support transaction-based processing without significantly degrading the performance of the system over time. The file structure that was developed to meet these demands was a B+ Tree. While a B+ Tree can be implemented as a data structure that resides strictly in memory, B+ Trees are specifically designed to be converted into a file structure and stored on disk. B+ Trees are such an elegant solution to this problem that virtually every Relational Database Management System (DBMS) uses a B+ Tree to store information. As elegant as a B+ Tree is for building a DBMS, much of the internal structure of both the DBMS and the B+ Tree is devoted to minimizing the impact of accessing a slow hard drive when compared to accessing main memory. The relative speed difference between accessing a hard drive vs. accessing main memory can easily reach 100,000 times. While a commercial quality DBMS is composed of hundreds of thousands if not millions of lines of code and requires many man-years to build, a small yet functional DBMS composed of a B+ Tree file structure for storing indexes and tuples (e.g., database records), a system catalog for defining the contents of a tuple, a buffer pool for caching tuples, and a relational algebra engine for executing queries can be created in two semesters by a single student if he possesses the correct prerequisites.

B+ Tree Overview

Before diving into the details of a B+ Tree works, let us explore the basic structure of a B+ Tree. A B+ Tree is a type of height balancing n-ary tree; in practice the arty of the tree may be 101 or more (i.e., 101-ary tree), though in a classroom setting a 5-ary B+Tree is sufficient. B+ Trees are composed of data nodes and index nodes. Data nodes or leaf nodes can only exist in the leaves of a B+ Tree and are used to store the tuples of a database. Tuples within a leaf node should be kept in sorted order (ascending) based upon the primary key of the tuple. Index nodes contain a list of keys in sorted order; they never contain a tuple with user information, only keys and some type of node reference (leaf or index). An index node that contains 4 keys will contain 5 references; an index node with 100 keys will contain 101 references. An index node with 4 keys can be considered to have the following structure:

(ref_0, key_0, ref_1, key_1, ref_2, key_2, ref_3, key_3, ref_4)

where ref_n is a reference to either another leaf or index node and key_n is a primary or secondary key value. The following constraints/properties must be maintained within the B+ Tree:

1) Keys of an index node must be maintained in ascending sorted order, $key_n < key_{n+1}$.
2) Tuples of a leaf node should be maintained in ascending order by their primary or secondary key value.
3) The subtree referenced by ref_n will only contain keys <= key_n and the subtree referenced by ref_{n+1} will only contain keys > key_n. This constraint establishes a relationship similar to that found in a binary search tree on how keys will be stored in the tree relative to the key found in a parent node.
4) A node, whether index or leaf, has a maxKeys value. The maxKeys value sets the maximum number of keys that can be stored in a node. The value of maxKeys can be different for index and leaf nodes. No node can ever exceed maxKeys. The value of maxKeys should always be set to an even value >= 4. Using an odd value for maxKeys adds an unnecessary complication to the implementation of the B+ Tree.
5) A node, whether index or leaf, has a minKeys value. The minKeys value sets the minimum number of keys and/or tuples that can be stored in a node. The value of minKeys can be different for index and leaf nodes. Only the root of the B+ Tree (leaf or index) can ever have less than minKeys. The value of minKeys is always defined as (maxKeys / 2).
6) Leaf nodes are connected together via a link list. The head of the link list will point at the leftmost leaf node in a tree. A single link list is all that is needed between leaf nodes; there is no reason to implement a doubly linked list between leaf nodes.

B+ TREE INSERT

For the purposes of this paper, leaf (data) nodes will be noted with brackets, and index nodes will be noted with parentheses. An empty B+ tree will consist of only the root node, which is also a leaf node and thus can contain data. An empty B+ tree will be depicted as [], whereas a B+ tree with the values 5, 21, 3, and 7 inserted into the root node as data would be depicted as [3, 5, 7, 21]. Notice that the data is now in sorted order. Each value represents a unique tuple within the B+ tree. Tuples can be added to the leaf node until the number of tuples added reaches maxKeys. For the purposes of this paper, maxKeys will always be 4 for both lead and index nodes.

Inserting another tuple with value 0 into the leaf node once it is at maxKeys will require the node to be split and an index node to be created in order to maintain the structure of the B+ tree. Creating the index node requires using what has been named the minMax. The minMax is the minimum value of a node containing the maximum values once a split has occurred. As the keys are sorted upon insertion into a node, this will always be the leftmost value in the rightmost node after a split has occurred. In the example of adding the value 0, the leaf node [3, 5, 7, 21] will be split into two leaf nodes [3, 5] and [7, 21]. The rightmost node after the split is [7, 21], and the minimum value is 7; thus, the minMax for this split is 7. The minMax is needed to create an index node

of the form (ref1, 7, ref2) to separate the two newly split nodes. Ref1 in the index node will point at the leaf node [3, 5], and ref2 will point at the leaf node [7, 21]. Figure 1 shows the resulting B+ tree. Notice how constraint 3 is maintained by finding the minMax and placing it into the root index node; all keys in the left-subtree are less than 7, and all keys in the right-subtree are greater than 7. Had any of the other existing values been used other than the minMax, then constraint 3 would have been violated. As the 0 was inserted, it was compared to the minMax; being less than 7, it was inserted into the node on the left of the 7.

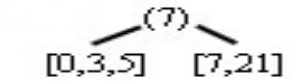


Figure 1. B+ tree after split

If a leaf node splits and there is already an index node above it, then the minMax for the split is simply added to the existing index node. In this manner an index node can grow to contain maxKeys keys before it must be split. Figure 2 shows the result of adding values 2, 6, 19, 13, 56, 12, 11, 15, and 35 to the B+ tree from Figure 1.

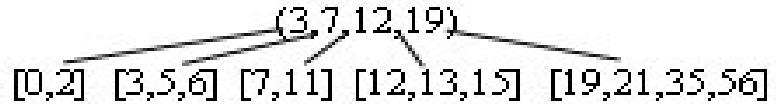


Figure 2. Inserting the values 2, 6, 19, 13, 56, 12, 11, 15, 35.

If 22 were the next value to be inserted, it would be compared against each value in the index node until the index key was greater than 22 or until it ran out of index keys. Here, 22 would be greater than 19 and would need to be inserted into the rightmost leaf node. The rightmost leaf node already contains maxKeys keys and would need to be split, yet adding another index node key would cause the index node to contain more than maxKeys keys, violating constraint 4. Therefore, the index node would also need to be split. The process works from the bottom up. First the data node is split, then the resulting minMax is passed up to the index node, and the index node is split, which then passes up a minMax for another index node to be created as seen in Figure 3.

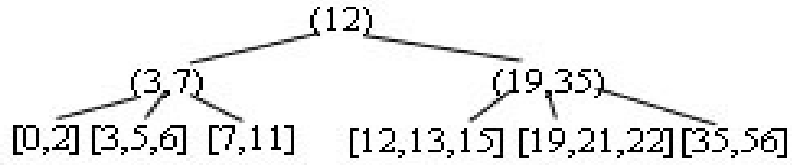


Figure 3. Tree after inserting value '22' and the resulting splits.

A key difference between splitting leaf nodes and splitting index nodes is that the minMax from a leaf node is copied upwards to become an index node key. When an index node splits, the minMax is actually removed and then moved upwards to a higher level, so there are no duplicate index node keys in the tree. Index nodes also must deal with even and uneven splits. This complexity exists in index nodes because the minMax

values are removed when splits occur unlike in leaf nodes where the minMax is only copied. Consider index node (3, 7, 12, 19) from Figure 2 being split into index nodes (3, 7) and (12, 19) respectively with 12 being the minMax. Yet if the minMax of 12 is removed, then index node (19) now has less than minKeys, violating constraint 5. This is fine if the original split occurred in the nodes below index node (12, 19) because a minMax value from the original spilt will be passed up to index node (12, 19) . So even if value 12 is removed from index node (12, 19), the index node gets another value and never drops below minKeys. This type of configuration is called an even split.

To understand an uneven split, consider the same setup as before. Index node (3, 7, 12, 19) is split into index nodes (3, 7) and (12, 19) with 12 being the minMax. However, if the original split occurred in the nodes below (3, 7), then when 12 is propagated upward to form a new index node there is no value to replace it, and the index node (19) will have less than minKeys. This can be fixed by performing an uneven split of index node (3, 7, 12, 19) into index nodes (3) and (7, 12, 19) respectively with 7 being the minMax. We already know that the minMax will be removed and propagated upward to form a new index node; thus, index node (7, 12, 19) will be (12, 19) after propagation. At first glance, index node (3) appears to be a problem because it is below minKeys. However, if the original split occurred in the nodes below (3, 7), then a minMax value is being propagated upward and will be added to the index node bringing the number of keys to minKeys. This splitting and passing up of minMax values to become new index nodes means that the B+ tree is effectively built from the bottom up. New index nodes are only created due to lower level splits occurring. Splits can propagate upward through the tree to create a new root for the tree. B+ trees grow upwards, as opposed to the B tree data structures documented in most books which grow downwards.

B+ TREE DELETE

Note that deleting always relates to removing tuples from a leaf node, just like adding always relates to inserting tuples into a leaf node. The creation and deletion of index nodes is a side effect of changing leaf nodes. When a key is to be deleted, it must first be found utilizing the structure of the B+ tree. Beginning at the root, the tree is traversed until the leaf node containing the key to be deleted is found. In the simplest case of deletion, the current node will have at least 1 more key than minKeys so that deleting the current key will not reduce the number of keys below minKeys, such as removing 10 from leaf node [3, 6, 8, 10] will produce leaf node [3, 6, 8]. If deleting a value from the leaf node would result in the number of keys dropping below minKeys, then an action must be taken to rebalance the leaf nodes unless the leaf node is also the root of the tree. Rebalancing actions require understanding the concepts of anchors, neighbors, and pivot keys.

For any given node (leaf or index) which we will refer to as the current node, the neighbor of the current node is a node at the same level in the tree as the current node and which also sits directly to either the left or right of the current node. It is possible for a node to not have both a left and right neighbor; however, unless the current node is the root, it will have at least one neighbor. A neighbor and a current node will share an anchor. An anchor is a node with the greatest level within a tree for which the current

node and the neighbor are both descendants. For both trees in Figure 4, the node labeled A is the anchor for nodes labeled B and C. The pivot key is the actual index node key which divides two neighbor nodes. In Figure 5, 19 serves as the pivot key between nodes A and B. 35 serves as the pivot key between B and C. Neighbor and anchor nodes must be computed for each node (current node) as the B+ tree is recursively traversed in a depth first search to locate the tuple to delete.

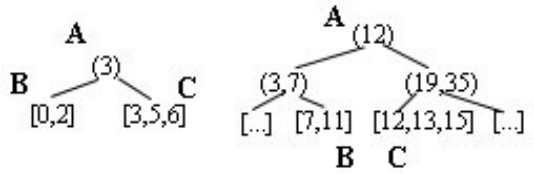


Figure 4. Anchor A.

Figure 5. Pivot Keys

The first rebalancing action that can be taken is a shift. A shift operates on the current node, a neighbor node, and the anchor of the current and neighbor nodes. In order for a shift to occur, the current node must have a neighbor with minKeys + 1 keys. If all neighbors are at minKeys, then a shift cannot occur and a merge will have to be done to rebalance the tree. If both the left and right neighbors are valid for a shift (have minKeys + 1), then one must be arbitrarily chosen. For our purposes, in the case of two valid neighbors, then the left neighbor will be chosen. Shift operations are identified as being either left shifts or right shifts. Data (whether keys, tuples or pointers) is always being shifted from a neighbor into the current node. The shifting of data is always toward the current node because it is only the current node which is having data deleted from it. A left shift occurs when a key from the left neighbor is moved into the current node. A right shift occurs when a key from the right neighbor is moved into the current node. When performing a shift on a leaf node (see Figure 6), a tuple and the keys associated with the tuple are moved from the neighbor leaf node into the current leaf node. In a left shift, the maximum value from the neighbor node is inserted into the current node. In a right shift, the minimum value from the neighbor node is inserted into the current node.

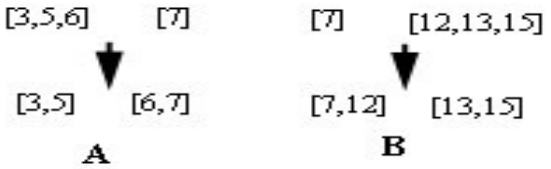


Figure 6. A)Left Shift and B)Right shift.

After a key has shifted from the neighbor into the current node, the pivot key in the anchor node for the current node and neighbor is updated with the minMax of the rightmost node involved in the shift. In the case of a left shift, it will be the current node; in the case of a right shift, it will be the right neighbor. Figure 7 shows the transformations of a B+ tree when value 21 is removed from the tree. Figure 7A is the original tree. Figure 7B shows value 21 removed from the leaf node. Figure 7C shows the result of a right shift to rebalance the tree with the pivot key updated. Figure 7D

shows the result of a left shift to rebalance the tree with the pivot key updated. Only a left or right shift is required to rebalance the tree from Figure 7B.

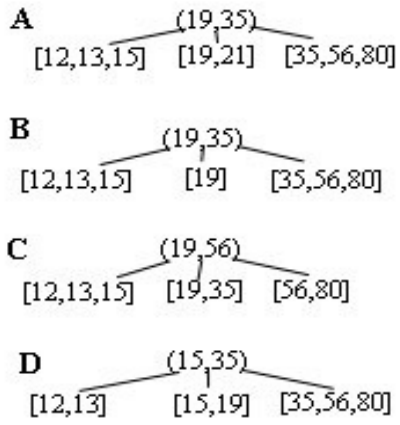


Figure 7. Rebalancing a leaf node.

When performing a shift on index nodes, the keys are rotated rather than simply moved. A right shift on index nodes will move the pivot key to the current node, and the minMax of the right neighbor is rotated to become the new pivot key. Figure 8 shows the result of rebalancing an index node via a right shift. Figure 8A shows the initial configuration of the index nodes with the arrows indicating the direction of rotation for the key being shifted. Figure 8B shows the index nodes after the right shift is completed and the tree is rebalanced.

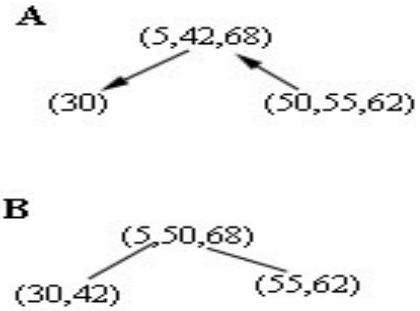


Figure 8. Rebalancing index nodes.

A left shift on index nodes operates much the same with the pivot key becoming the new minMax of the current node, and the maximum key of the left node becoming the new pivotKey. Shift operations can never cause a node to have less then minKeys (underflow) nor more than maxKeys (overflow). A shift operation cannot alter the

number of nodes present in the tree or the height of the tree. The structure of the tree will remain intact throughout a shift and no nodes beyond the anchor node, current node, and a neighbor will be modified. A shift does not propagate any changes up the tree further than the anchor node.

The second balancing action that can be performed is a merge. A merge occurs when a key is removed from a node, the number of keys in that node falls below minKeys (underflow), and a shift cannot be performed due to no neighbors having at least minKeys+1 keys. A merge involves taking all keys in one node and inserting them into another node. A left merge moves all keys from the current node into a left neighbor. A right merge moves all keys from a right neighbor into the current node. One important exception that exists for deletes is the root. The root is the only node allowed to be reduced to fewer than minKeys keys. It is possible for the root to become empty.

SYSTEM CATALOG

For a database to be useful it must be able to handle holding various data types within a tuple. When stored on disk, all information is simply in binary format and the task of properly reading, writing, and utilizing the data held on disk falls to the system catalog. The system catalog is used in a manner similar to a mask, overlaid on the binary data read from disk to parse it into the appropriate attributes of the tuple. The system catalog generally consists of entries in a file that specify to the database, the table/database an entry applies to, the name of the attribute, the data type of the attribute, and its position within a tuple. An example system catalog may look like this:

myDB, key, int, 1
myDB, name,V20,2
myDB, SSN, V20,3
myDB, age, int, 4

The first entry would specify that in the database named 'myDB', there is an attribute named 'key' that is an integer and is the first attribute for this tuple. Further entries specify a 'name' and 'ssn' field that utilize a V20 (a character array of length 20) data type. When the system catalog is read in and interpreted, a data mask is constructed that can be overlaid on data.

BUFFER POOL

In an effort to reduce the number of disk reads required to access data, it is useful to buffer, or cache, frequently accessed tuples so that they do not need to be re-read. An excellent candidate for buffering, for example, is the root node which must be accessed on every lookup performed. The concept of an RID, or record identifier, is important to the use of a buffer pool. In memory, nodes can be linked together through the use of pointers or similar mechanisms depending on the language. When the nodes are written to a file, they are instead linked together through RIDs, which generally represent the absolute offset of that record into the file. For example, an RID of 128 would indicate that the record is the 128th record in the database file.

The buffer manager utilizes a section of memory allocated specifically for caching nodes. As a node is read in from a file, it is inserted into this section of memory. When the database needs to access a node, it requests the node through the buffer manager using the desired nodes RID. The buffer manager first checks to see if it has a node with that RID already in memory, and if so simply returns it. If it does not, the buffer manager has the node read in from file, stores it in memory, and returns it. In this manner, the database system has no working knowledge of the buffer manager or if the nodes it is requesting are already in memory coming from disk, but can always access the node it needs through the appropriate RID. When buffer pool becomes full, it will become necessary to write modified nodes back to disk to make room for more incoming nodes. The means for selecting which buffered nodes are to be replaced is handled by a replacement policy. There are various replacement policies including first in-first out, last accessed, and even random replacement.

With the B+ tree being so efficient at minimizing disk hits, it is worth exploring how effective adding a buffer pool is in terms of performance. Tests were run on read and write speeds using a B+ tree database. A thousand values were inserted into the database with half being inserted initially and the remaining values being inserted with random value lookups performed between each insertion. This test simulates an environment in which the database is primarily utilized to look up existing information with occasionally new data being entered. The control for these tests uses no buffer pool. The first test cases used a first in-first out replacement policy; the second test case used a random replacement policy. Four tests were run for each case using different sizes of memory allocated for the memory pool, as represented by a percentage of the total size of the database: 1%, 5%, 7% and 10%. Figures 9 and 10 show the results of the test cases. Overall, using a buffer pool is always better than using no buffer pool. Read times improve dramatically as the size of the buffer pool is increased, though this has little effect on write times. The difference between replacement policies is seemingly very minor.

REFERENCE

[1] Ramakrishnan, R., Gehrke, J., *Database Management Systems*, New York City, NY : McGraw-Hill, 2002.

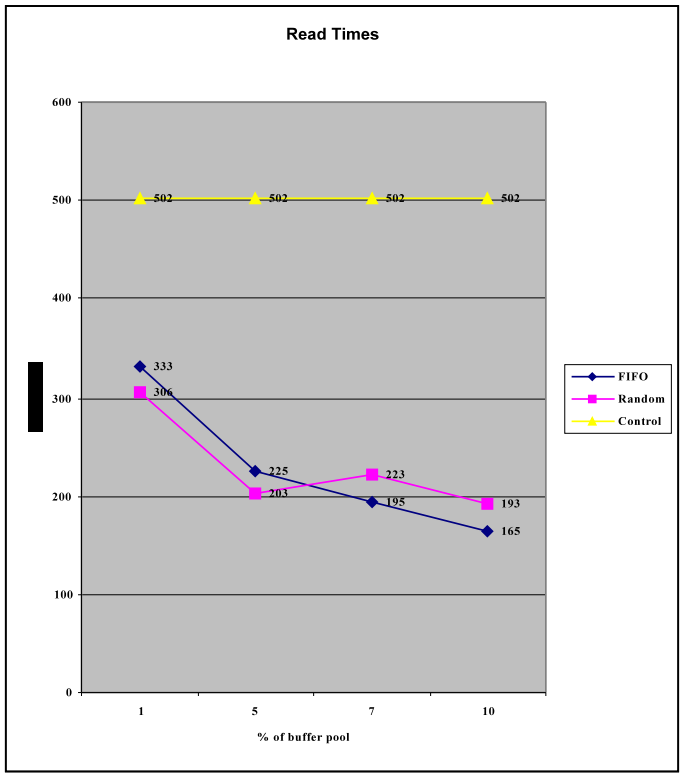


Figure 9. Read time results.

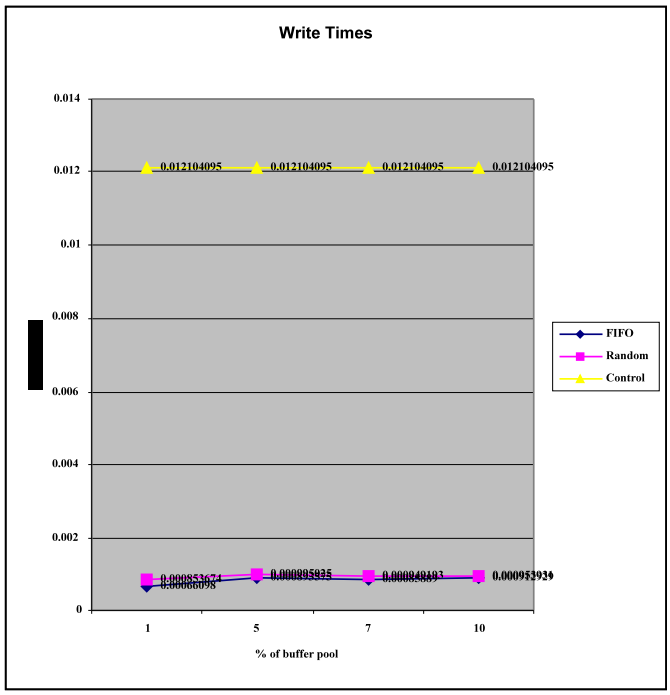


Figure 10. Write time results.

LESSONS LEARNED IN GENERATING STEREOSCOPIC IMAGES*

Don Jordan
Curtis Welborn
Computer Science Department
Utah Valley University
Orem, Utah 84058
801 863-7058
kc7zax@hotmail.com
Curtis.Welborn@uvu.edu

ABSTRACT

An application can generate Stereoscopic 3D images by rendering the same image from two slightly offset viewing angles to produce a left and right eye image. When the scene is displayed, the left and right eye images are interlaced producing what appears to be a fuzzy shaking object to a viewer. Yet, if the viewer is wearing 3D glasses that are synched with the graphics card as it generates the left and right eye images, the 3D glasses will alternately obscure one eye from being able to see. Thus only the right eye will see the right eye image and only the left eye will see the left eye image. The brain will combine the interlaced images making one image which appears to stand out from the screen. Because the human viewer is looking through the 3D glasses at the display, they can see not only the Stereoscopic 3D image but other real objects around them, such as the mouse, keyboard, and their own hands. OpenGL is an Open Source library which allows complex cross platform 3D graphics applications to be developed using C/C++. During the process of converting existing OpenGL application to create Stereoscopic 3D images, it was discovered that GPU vendor NVIDIA only offers limited support for Stereoscopic 3D images when OpenGL is being used. As a result the authors have been forced to convert existing OpenGL applications to Direct3D so as to be able to make these applications support Stereoscopic 3D images. This

---

paper will cover our experience in porting an OpenGL application to Direct3D and the additional features that must be added to support Stereoscopic 3D images.

INTRODUCTION

As evident by the success of such movies as *Avatar*, 3D effects are currently a very important technical innovation. While popular culture refers to these effects as 3D, the engineering term for these affects is Stereoscopic 3D, which helps to distinguish this technology from 3D Computer Graphics. Under normal circumstances, each eye of a human will focus upon an object and capture two separate images of the same object. These images are offset slightly because human eyes are offset slightly in the head. The brain then meshes these two images to form a single Stereoscopic 3D image of the object which allows a person to rather accurately judge distance to an object. This phenomenon can be observed by focusing on objects located at different distances and alternately covering one eye at a time. The object will appear to shift right and left, with the amount of shift being dependent upon the distance of the object. Computer/Camera-based Stereoscopic 3D images are created by tricking the human brain in one of two ways:

1) In a movie, a scene is captured using two slightly offset cameras producing a left and right eye image. When the scene is projected for a human viewer, the left and right eye images are superimposed upon each other, with the light for each image being polarized in a different direction. A viewer looking at the images will see what appears to be a fuzzy out of focus object. Yet, if the viewer is wearing polarized glasses synched with the left and right image polarization, then only the left eye sees the image projected for the left eye, and only the right eye sees the image projected for the right eye. The brain then combines the images into a single clean image that appears to stand out from the flat viewing surface.

2) On a computer, a 3D scene is rendered by a 3D Computer Graphics application using two slightly offset viewing angles producing a left and right eye image. When the scene is displayed to a human viewer, one of two strategies can be used:

    2.1) The left and right eye images can be interlaced (alternated left then right) to produce the display. A viewer looking at the image will see what appears to be a fuzzy shaking object. Yet, if the viewer is wearing 3D glasses that are synched with the graphics card as it generates the left and right eye images, the 3D glasses will alternately obscure one eye from being able to see. Thus only the right eye will see the right eye image and only the left eye will see the left eye image. The brain then combines the interlaced images making one image which appears to stand out from the flat display surface. Because the human viewer is looking through the 3D glasses at the display, they can see not only the Stereoscopic 3D image but other real objects around them, such as the mouse, keyboard, and their own hands. For this strategy to work, the graphics card must be able to interlace the two images at a high rate of speed. Because of this high rate of interlacing, the display monitor must be able to sync and refresh with the graphic card at speeds of 120 Hz. This is in contrast to typical

monitors with refresh rates of only 60 Hz. Additionally, the graphics card must send a signal to the 3D glasses telling them which eye to obscure.

2.2) The viewer must be wearing a pair of Virtual Reality (VR) glasses that completely cover both the left and right eyes. The VR-glasses have a right and left eye display built into the glasses. The right display will only show the right image, and the left display will only show the left image. As before, the brain will combine the images on the two displays to create one image that appears to be right in front of the viewer. Once the VR-glasses are in place, the viewer can only see what images are displayed by the VR-glasses. For a viewer to see objects around them, the objects must be rendered within the 3D scene by the 3D Computer Graphics application. VR-glasses require two tiny display screens with excellent screen resolution; if high quality screens are not used, the viewer will notice the graininess of the images due to the close proximity of the screens to the human eye. High quality VR-glasses can cost in excess of $15,000.

A grant that has been received by the Computer Science Department has allowed the authors to purchase laptops, monitors, and 3D glasses for generating and viewing Stereoscopic 3D images. OpenGL is an Open Source library which allows complex cross platform 3D graphics applications to be developed using C/C++. OpenGL is one of the core technologies used in the department's "Applied 3D Graphics" course. It was the original intention to convert one of the more interesting OpenGL applications created during the course to create Stereoscopic 3D images. During the process of converting the OpenGL application to create Stereoscopic 3D images, it was discovered that the GPU vendor NVIDIA only offers limited support for Stereoscopic 3D images when OpenGL is being used. As a result, any existing OpenGL applications had to be converted to Direct3D so as to be able to make these applications support Stereoscopic 3D images. This paper will cover the experience of porting an OpenGL application to Direct3D and the additional features that must be added to support Stereoscopic 3D images.

THE CONVERSION PROBLEM

Early in the history of computer graphics, computers were too slow to allow effective random access to the main display device, most often a graphics monitor. In order to allow the monitor to display a single continuous image and the program to continue along its path of logic, the practice of storing buffers of display data was developed. These buffers can be handed to the monitor in a single step and then displayed according to the configuration of the monitor while the application continued the logic path writing all new display data to another buffer. Typically, in an OpenGL and/or Direct3D application double buffering or triple buffering is used. For the sake of simplicity and because it is the most common feature among dedicated graphics hardware, everything will be presented in the context of double buffering. As mentioned earlier, the easier of the two methods for presenting Stereoscopic 3D on computer monitors is via glasses with the ability to obscure each eye independently while the images are synced with the monitor's refresh rate. This method is called Active Shutter and is the method used by NVIDIA in their 3D Vision product line and technology. Because the monitor

switches rapidly between the left and right eye images, the monitor requires two buffers of display data for a single frame of video. This necessitates that the application have more than two buffers so that it may continue writing the next frame to additional buffers. A second pair of buffers is required for a total of four buffers. A valid first assumption would be that control over these buffers is required for effective stereoscopy. This control is possible in OpenGL with the GL_LEFT_FRONT_BUFFER, GL_RIGHT_FRONT_BUFFER, GL_LEFT_BACK_BUFFER and GL_RIGHT_BACK_BUFFER states. However, during initial investigation into rendering to the four buffers, it was discovered that the quad buffering feature in OpenGL is not typically provided in their consumer line of hardware. Specialized workstation and computing server hardware is required to provide that functionality. Instead, systems like NVIDIA's 3D Vision system are hooked into Direct3D because it is the most popular 3D graphics system in the entertainment market. Discovering this required a translation of knowledge, process, and code to the Direct3D API from the OpenGL API.

THE CONVERSION PROCESS

When converting from OpenGL to Direct3D, the process is reasonably straightforward as the fundamental aspects of 3D graphics programming remain in effect. The largest changes occur due to the two different paradigms in which OpenGL and Direct3D were architected. The differences occur in two areas of the design: the relationship to graphics rendering hardware and the presentation of the API to the programmer. OpenGL was designed originally as a state-based system that provided a rendering system that manufacturers would implement in their hardware or driver. Although in the more recent versions of the OpenGL specification OpenGL has evolved to be more of an objective-based system, the code being converted was written using OpenGL features present in the first major revision and some minor revisions of the specification. Direct3D enumerations are listed based on the Direct3D9 API. Direct3D, on the other hand, uses the existence of the HAL (Hardware Abstraction Layer)[2] to provide a universal API over the architecture of the graphics hardware itself. This provides a simple abstraction from hardware allowing the programmer to directly utilize the capabilities of graphics hardware without worrying about the hardware itself. The differences in these design decisions also shift the burden of resources as OpenGL hardware manufacturers must maintain 3D hardware resources within their implementations, and Direct3D leaves the managements up to the application. The architecture of the APIs themselves also differs due to the architectural decisions. OpenGL has method calls that alter the state of the system at any given time. Direct3D uses Microsoft's COM interface to present objects to the programmer in a language independent method. Other minor differences between the two systems include their default handedness (right-handed vs. left-handed) for Cartesian coordinate orientation among other things depending on the feature set being used. Once the necessary knowledge of the inherent differences was identified, the conversion process could begin.

The first step was to identify all the state changing calls in the OpenGL code, such as glBegin and glRotate and map them to their appropriate Direct3D calls. In the example of glRotate, Direct3D does not have anything that requires device wide state associated

with the ID3DDevice interface. Direct3D uses matrices directly instead of relying on the internal implementation, whereas the OpenGL code had to perform the necessary matrix multiplication. Because the conversion from OpenGL to Direct3D was a necessity for the purpose of studying Stereoscopic 3D images on the personal computer, D3DX was used to help perform the matrix operations and to keep the matrix data in a format compliant with the requirements of the ID3DDevice object. The resulting code was to call the appropriate D3DXMatrixRotation, Scaling, or Transform method, and then call ID3DDevice::SetTransform with the resulting matrix [1]. Once the system state calls were converted, the geometry itself was converted.

This was not an easy task as neither Direct3D nor D3DX have any convenience primitive functions similar to the GLUT library. Instead, the geometry is generated using trigonometric equations, and then Direct3D is fed the triangles like any other mesh would be. The original OpenGL code utilized display lists with simplistic glBegin and glEnd operations for defining vertices and triangles. Direct3D uses a different method that is similar to one of the newer OpenGL functions called glDrawArrays. Direct3D asks for what is called a Flexible Vertex Format (FVF) that defines the contents of a buffer of data. The FVF tells Direct3D where in the buffer to find vertex data, texture coordinate data, etc. The CreateVertexBuffer method is used to create a buffer according to the FVF. The buffer can then be locked and unlocked to enter/change data in the buffer. The ID3DDevice is told to use the FVF and the buffer is streamed to the graphics hardware. Finally, the DrawPrimitive method is used to tell Direct3D which parts of the buffer the user wishes to use [3]. Adding color and texture to the now converted primitives is simply a matter of utilizing more of the methods and interfaces available in the Direct3D objects created up to this point.

Once the OpenGL code was in a Direct3D project and working, work could proceed toward learning the basics of Stereoscopic rendering using the existing content. NVIDIA has with 3D Vision simplified much of the work to do for Stereoscopic rendering. This is done so that existing games and applications can make some use of the technology without the developers having to release updates for outdated titles. When the developer sends geometry and other information to Direct3D, the NVIDIA graphics driver intercepts the calls before they head to the graphics hardware and adjust the world transform to the left and right eyes and then passes the scene to the hardware and renders the scene twice: once using the left eye world transform, and once using the right eye world transform. This also allows the user to customize the Stereoscopic viewing to his own comfort. NVIDIA provides a user interface wherein the user calibrates the 3D glasses and monitor to the distance between their eyes and whichever refresh rate is optimal. This information is used by the graphics hardware driver to determine the left and right transforms to be sent to the hardware. When the driver is unable to automatically convert aspects of a scene such as a displacement shader, the developer has access to the NVAPI to tweak the 3D Vision system to their application. NVIDIA has also provided the results of internal research so that developers can forego repeating many of the mistakes inherent to developing for stereoscopic 3D [4].

CONCLUSION

Converting existing code from OpenGL to Direct3D is a seemingly straightforward task, especially with newer versions of OpenGL and Direct3D being closer together in design. However, as straightforward as it is, it is a time-consuming task. Finding all the approximations of OpenGL state and matching OpenGL C objects to Direct3D COM objects is daunting and requires research using documentation and tutorials on both sides of the equation. However, once the application is in Direct3D, whether converted or written from scratch, NVIDIA has made the conversion to Stereoscopic 3D a fairly painless process for developers while at the same time making it easier on the end user; they are to be commended. It is rather disappointing from a research standpoint that direct control of the buffers being shown is not provided to consumer level graphics hardware but, such is the marketplace. While it is a good step in the right direction for tweaking the system, the NVAPI that is publicly available is rather lackluster in its capabilities. There is a lower level NVAPI available to those who qualify that may provide more control, but that API was not made available to the research project. Stereoscopic 3D is a technology that keeps coming back with bigger and better ways of doing it and in that regard is something to be considered. People want 3D imagery, but whether the fake 3D of stereoscopy is the answer or just a stepping stone is yet to be seen. Many say that Stereoscopic 3D is more like watching a puppet show where there is depth to the objects but the objects themselves remain only where their material or visual properties can take them. Stereoscopic 3D will really hit a landmark when the technology allows for individual vertices to be separated based on their Z depth and not the depth of the entire portion of the image represented by the mesh. By allowing individual vertices to be offset instead of the mesh as a whole, a greater sense of immersion will occur because it will appear as though the object itself is part of the scene instead of simply popping in or out of the scene. In short, Stereoscopic 3D is not a mature technology. As such, it is hard to say whether or not it will last.

FUTURE WORK

In the future, Stereoscopic 3D would benefit from research into dynamic stereoscopy of elements that make up the generated image. There are various other 3D graphics technologies to explore and study that also correlate with the desire to bring virtual images to life. A particular interest to this author is that of production rendering. Production rendering systems use algorithms that work with billions of polygons and more often than not non-polygonal primitives to bring masterful 3D scenes to life. These systems seem like they would be able to take advantage of the advancements made in the consumer graphics hardware market. NVIDIA and AMD (formerly ATI) graphics cards can consume mass quantities of floating operations simultaneously. Global Illumination is coming to maturity in both the entertainment and gaming industries. Global Illumination typically requires methods not in use by either industry if proper quality and frame rates are to be reserved. Moore's Law is not advancing in such a way as to improve the speed of production rendering. Despite the advancements in computer technology, modern 3D movies take more time per frame than their fifteen-year-old siblings.

REFERENCES

[1] DirectX Tutorial.com - Direct3D Basics Lesson 5: Transforming Vertices, 2009, http://www.directxtutorial.com/Tutorial9/B-Direct3DBasics/dx9B5.aspx, retrieved June 9, 2011.

[2] Direct3D Architecture (Direct3D 9) (Windows), 2009, http://msdn.microsoft.com/en-us/library/bb219679%28v=vs.85%29.aspx, retrieved June 9, 2011.

[3] DirectX Tutorial.com - Direct3D Basics Lesson 4: Drawing a Triangle, 2009, http://www.directxtutorial.com/Tutorial9/B-Direct3DBasics/dx9B4.aspx, retrieved June 9, 2011.

[4] 3D Vision and Surround Technology | NVIDIA Developer Zone, 2009, http://developer.nvidia.com/3d-vision-and-surround-technology, retrieved June 9, 2011.

CONSTRAINED 3D FLOCKING BEHAVIOR*

Greggory Hernandez
Curtis Welborn
Computer Science Department
Utah Valley University
Orem, Utah 84058
801 863-7058
greggory.hz@gmail.com
Curtis.Welborn@uvu.edu

ABSTRACT

Craig Reynolds introduced the concepts of simulated flocking behavior via the use of simple aggregate Boid behaviors which make the Boids appear to exercise independent autonomous actions. The term Boid is used to identify the actors within the flock without attaching any particular animal characteristics with the actor (e.g., fish, bird, mammal). Artificial potential-fields (APF) are often used to implement collision avoidance for robotic and multi-agent systems, making them well suited for implementing flocking behaviors. AFP can also be used to define constraints that direct the motion of a flock while still allowing for simple aggregate Boid behaviors. Flocking behavior is used in computer games or movies where groups of Boids are to move together without appearing to move in lockstep. This paper explores the approach used for directing the flocking behavior of Boids via APF constraints. The APF constraints are intended for an end user (e.g., animator or director) inexperienced with programming or the technical details of implementing flocking behavior or APFs. Our constraint system for controlling the motion of the flock is currently composed of two constraint types:

A fly-by waypoint constraint is used to control the direction of the flock in a manner similar to the scripted flocking defined by Reynolds. A user defines 3D waypoints on the screen to map a sequential path that the flock is to follow.

A boundary constraint is used to control the outward spread of the flock. A user defines a boundary between waypoints that sets the maximum outward spread of the flock by limiting the flock to move within the channel/conduit defined between the waypoints.

INTRODUCTION

Craig Reynolds [2] introduced the concepts of simulated flocking behavior via the use of simple aggregate Boid behaviors which make the Boids appear to exercise independent autonomous actions. The term Boid is used to identify the actors within the flock without attaching any particular animal characteristics with the actor (e.g., fish, bird, mammal). Artificial potential-fields (AFP) are often used to implement collision avoidance for robotic and multi-agent systems, making them well suited for implementing flocking behaviors. AFP can also be used to define constraints that direct the motion of a flock while still allowing for simple aggregate Boid behaviors. Flocking behavior is used in computer games or movies where groups of Boids are to move together without appearing to move in lockstep. This paper explores the approach used for directing the flocking behavior of Boids via APF constraints. The APF constraints are intended for an end user (e.g., animator or director) inexperienced with programming or the technical details of implementing flocking behavior or APFs. Our constraint system for controlling the motion of the flock is currently composed of 2 constraint types:

A fly-by waypoint constraint is used to control the direction of the flock in a manner similar to the scripted flocking defined by Reynolds. A user defines 3D waypoints on the screen to map a sequential path that the flock is to follow.

A boundary constraint is used to control the outward spread of the flock. A user defines a boundary between waypoints that sets the maximum outward spread of the flock by limiting the flock to move within the channel/conduit defined between the waypoints.

ARTIFICIAL POTENTIAL-FIELDS

Now that an introduction has been given involving some math for moving and orienting Boids in 3D space, a methodology is needed for making the Boids avoid collisions while being attracted to a flock. The methodology utilized here is Artificial Potential Fields [1] (aka Pfields). In a limited sense, Pfields is the application of vectors to the problem of robot navigation. In robot navigation every object, including the robot, is associated with one or more vectors. These vectors can be characterized as being either repulsive or attractive.

When looking at an object with a repulsive Pfield (Figure 1), the vectors are always directed away from the object. The magnitude of the vectors associated with the Pfield should become larger when you get closer to the object. As an analogy, consider standing in a room with someone you do not know and who looks a little creepy. When they are on the other side of the room, you just ignore them. If they move within about 5 feet, you will begin eyeing them; when they move within a foot, you might stick out your hand to keep them from bumping into you. As they move closer to you, you use a greater repulsive field to keep them away from you.

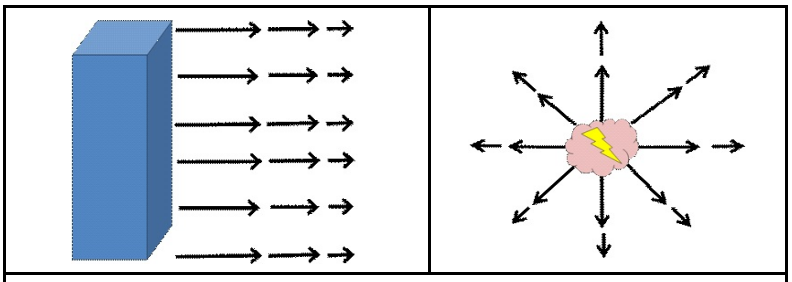


Figure 1. Repulsive Pfields.

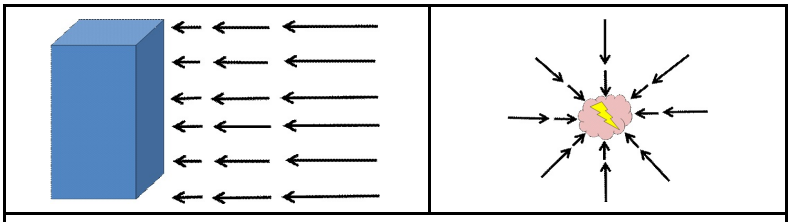


Figure 2. Attractive Pfields.

Attractive Pfields (Figure 2), always point toward the object, with the magnitude of the vectors becoming smaller as you get closer to the object. As an analogy, consider parking your car in your garage. When you turn onto your street, you are compelled to drive your car all the way to your house even if there is traffic on your street. However, as you get closer to the back wall of your garage, your desire to keep getting closer should diminish, unless you want to park inside the house.

Pfields will be associated with every object in the flocking simulation. Boids will have attractive Pfields between each other when they are within a specified distance to encourage their flocking. They will also have repulsive Pfields when they are close together to avoid them colliding with each other. Obstacles in the simulation will have repulsive Pfields that interact with the repulsive Pfields of the Boids to keep the Boids from colliding with the obstacles.

FLOCKING BEHAVIOR

In addition to the attractive and repulsive Pfields associated with every object, Boids also contain a single motion vector. A Boid's motion vector determines its heading and speed. To build upon the driving analogy from above, when you press the accelerator or brake pedal in your car, the car's motion vector will increase and decrease in magnitude, respectively. Turning the steering wheel would alter the direction that the motion vector is pointing. When a Boid enters the range of another Boid's attractive Pfields, the two Boids are pulled together to simulate flocking behavior. The best method to use to pull Boids together is to simply sum the two Boids' motion vectors and gradually apply the (normalized) resultant vector to each Boid.

This method makes the Boids' motion appear natural and smooth when the simulation is run. This works even when numerous Boids are in close proximity to each other since ultimately all the Boids' different motion vectors will converge on what is effectively a single average vector that all the interacting Boids will possess. As new

Boids come within range of the flock, the foreign Boid is assimilated by the motion vectors of the Boids nearest to it.

When the simulation is run, none of these interactions are under the control of the end-user and so the animation is entirely determined by the computer. This may not always be a desired situation. While it is appropriate to let the computer handle the more complex calculations, allowing the user some control can enhance the creative potential immensely.

USER CONSTRAINTS

With the implementation of flocking behavior and the entirety of the Boid simulation, there was always the glaring omission of user control. Up to this point, the Boid simulation that was implemented was entirely under control of the programmer. An end-user had no way of altering the parameters of the animation without being forced to dig into the code and tediously make changes in the hopes that it would be the change that was desired. In an attempt to remedy this, two constraints will now be introduced that the user will have direct control over: fly-by waypoints and boundary constraints.

Fly-by Waypoint

The first constraint available to a user is the Fly-by Waypoint constraint. This allows a user to input an arbitrary set of three-dimensional points (waypoints). When the user starts the simulation, a predefined number of Boids will be created (in a random configuration) around the first waypoint, and then the Boids will move as a flock to each waypoint in the order that the user entered them. This allows a user to create complex paths and more sophisticated animations without being required to have an intimate knowledge of the simulation's inner-workings.

In addition to the base flocking simulation described above, only two main constructs are needed to simulate fly-by waypoints: a list of waypoints, and a method for tracking the current waypoint. A method for allowing the user to input waypoints at runtime and a method for the user to start the simulation once all desired waypoints are entered is also required, but presently the focus will be on the implementation of the actual constraint rather than the details of exposing it to the user. It is much more important that the list of waypoints and tracking of the current waypoint be implemented correctly. Once that is accomplished, the details of a user interface is a trivial matter. All that is currently important to know is that a user is actually able to enter waypoints and can start the simulation. In addition, the simulation has two states: running and not running. The simulation always starts in the "not running" state and only enters the "running" state when instructed to do so by the user.

The list of waypoints is implemented as a circular first-in/first-out list. That is, a queue that goes back to the beginning of the queue upon reaching the final element. Each time the user inputs a new waypoint, that waypoint is added to the end of the queue.

The current waypoint initially refers to the waypoint at the front of the queue. When the Boids reach the current waypoint, that current waypoint will be changed to refer to the

next waypoint in the queue. If the current waypoint is the last waypoint in the queue, the current waypoint will be changed to refer again to the first waypoint in the queue; otherwise, the Boids would remain flocking around the last waypoint ad infinitum. In addition to tracking the current waypoint, each waypoint also has a state of "hit" or "not hit." Initially all waypoints are set to "not hit." Each time a waypoint is reached by the Boids, that waypoint is set to "hit." When the end of the queue is reached, all waypoints are reverted to "not hit." Among other things, this can allow for a visual queue to the user letting them know how far the Boids are along the path.

In this implementation, when the simulation begins, a user is able to click to place a waypoint in the x/y plane. Once it is placed, the user can then adjust the z value for that waypoint. When the user is satisfied, the user sends the "go" instruction to start the simulation.

Boundary Constraint

Depending on the configuration of the waypoints, Boids may end up with a large outward spread as they move between waypoints. They might also exhibit a sudden change in direction as the leading Boid reaches the waypoint first, and the current waypoint transitions to the next waypoint. These may not always be desired effects. To counteract this, a user is given the ability to set a maximum boundary in order to hold the Boids closer to the line created between the source and destination waypoints. With this constraint in place, Boids will move between waypoints without an overly large outward spread or odd behavior when changing directions.

A boundary constraint will be implemented that holds the Boids within a cylinder between the previous and current waypoint. The cylinder will have a user-defined radius. This radius is the maximum distance that a Boid is allowed to be from the line segment created between the two active waypoints. If any Boid reaches a distance from this line that is approximately equal to or greater than the cylinder's radius, it will be pulled back towards the nearest point on the line.

To compute the cylinder, the current waypoint and the previous waypoint are needed. If the current waypoint happens to be the first waypoint in the list, the last waypoint in the list is used as the previous waypoint. Once the previous and current waypoints are obtained, compute the axis-of-the-cylinder--a line that extends infinitely in both directions and passes through the previous waypoint and the current waypoint. After the axis-of-the-cylinder is found, the following process will be applied to each Boid in the simulation.

First compute the distance between a Boid and the axis-of-the-cylinder. To compute the distance, form a right triangle (see Figure 3) with the following points: the position of the current waypoint, the position of a Boid, and the point closest to the Boid on the axis-of-the-cylinder. Since the length of the hypotenuse is known (distance between the current waypoint and the Boid), and the angle (theta) can easily be derived between the hypotenuse and the segment between the current waypoint and the closest point on the axis-of-the-cylinder, the equation: dist = length of hypotenuse * sin(theta) can be used to find the Boid's distance from the line. If the distance is greater than the radius supplied by the user, then the Boid needs to be pulled back towards the center of that radius.

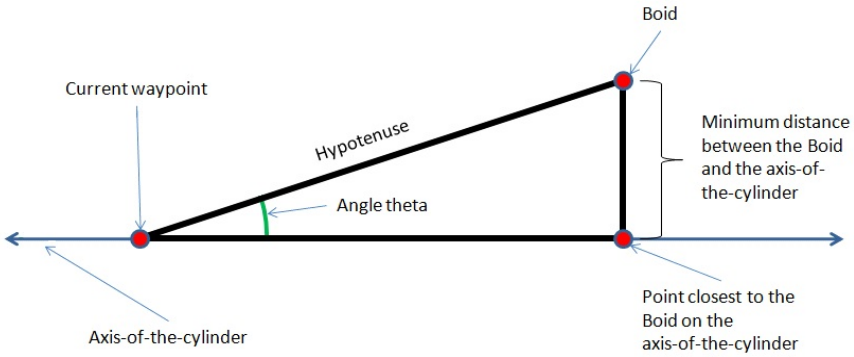


Figure 3. Distance between Boid and axis-of-the-cylinder.

While the Boid's motion can be altered using a vector that points from the Boid to the nearest point on the line, Boid movement is more natural and less abrupt if a vector is used that is a small distance further along the line segment nearer the current waypoint. Once this vector is determined, just sum the vector with the Boid's current motion vector to produce a new trajectory for the Boid that moves it toward the axis-of-the-cylinder.

With this method implemented, all Boids in the simulation will be forced to remain within the boundary constraint as defined by the user. While the primary purpose of this constraint is to allow the user more control over the details of the animation, as a secondary benefit this implementation of the boundary constraint also provides flocking behavior that is more akin to what is seen when a real-world flock turns a corner. The sequence of screenshots shown in Figures 4, 5 and 6 shows a flock of Boids following waypoints (red and blue dots) with a boundary constraint to confine their movement.

Figure 4. Boids moving toward read waypoint.

Figure 5. Boids turn after reaching blue waypoint.

Figure 6. Boids approaching red waypoint.

CONCLUSION

Adding the constraints of waypoints and boundaries adds a new level of sophistication to the original Boid flocking simulation. In addition, the end-user is allowed to have control over the simulation in ways that were not previously possible. With this extra layer of flexibility, higher quality animations become possible more easily by users with less experience.

Both the fly-by waypoints and the boundary constraints provide control without sacrificing the believability of the simulation in most cases. There are some notes to make about maintaining believability. With proper care, waypoints can provide a very

believable animation. However, the user should be aware that a more realistic simulation will usually have more fly-by waypoints in a complex configuration in order to avoid having paths that are predictable or that feel contrived. The user is, of course, free to create waypoints without restrictions (except perhaps those imposed by the computer on which the simulation is being run). With the boundary constraints, making the boundary too small can cause abhorrent behavior. The Boids will basically have nowhere to go so they end up jerking back and forth, often intersecting each other and remaining outside of the boundary. Making the boundary too small should be avoided. Experimentation is the best method to determine what works for any given simulation.

FUTURE WORK

As for the future of this work, active investigation is being done regarding the design and implementation of additional user constraints to control flocking behavior. While there is possibly an endless variation of constraints that could be manufactured, the additional constraints being considered can be roughly grouped into one of two categories of constraint types.

The first are constraints to restrict the shape of a moving flock. The boundary constraint defined in the paper is one example of a shape restricting constraint; additional 3-dimensional shape restrictions are being considered. When restricting the shape of the flock, it is important not to impose an unnatural laminar flow (characterized by smooth parallel changes) or turbulent flow (characterized by chaotic changes) on the flock.

The second are constraints that direct the movement of the flock as a whole. The fly-by waypoint constraint defined in the paper is one example of a flock movement restricting constraint. While the fly-by waypoint constraint allows a user to influence the direction of the flock, new constraints in this category will focus on influencing the speed of the flock.

REFERENCES

[1] Murphy, R.R., *Introduction to AI Robotics*, Cambridge, MA: MIT Press, 1998.

[2] Reynolds, C.W., Flocks, herds and schools: a distributed behavioral model, *Computer Graphics*, 21(4), 25-34, 1987.

CHALLENGES TO NETWORK SECURITY ON COLLEGE CAMPUSES*

Russell Jones
Computer and Information Technology Department
Arkansas State University - Main Campus
State University, AR 72467
(870) 972-3988
rjones@astate.edu

Tamya Jean Stallings
Information Technology Services
Arkansas State University - Newport
Newport, AR

ABSTRACT

With the growth of hacking, denial of service (DoS) attacks, virus/worm propagation, and other modes of computer crime, network security has become a growing challenge in all types of organizations throughout the United States and the rest of the world. With the more open nature of libraries, secondary schools, and colleges/universities, cybercriminals are looking increasingly at these locations as a point from which to launch their attacks.

In response to these increasing attacks, organizations are beginning to look at areas to improve the protection of their Information Technology infrastructure. However, due to current economic conditions and the hierarchy of needs on most higher education campuses, these institutions are falling behind their commercial counterparts in providing, at least, minimal protection for their systems. While most higher education campuses are behind the norm on computer security, two-year campuses are especially vulnerable to network security threats. Their open labs, limited staff, and limited budgetary resources make them a desirable target for the cybercriminal.

This paper focuses on the challenges that these institutions have faced in attempting to protect their data and the network infrastructure from illegal activity. Data was collected from over fifty institutions from seven states.

These states were: Arkansas, Louisiana, Mississippi, Missouri, Oklahoma, Tennessee, and Texas. Results generated from these two-year institutions indicated that, while network security has become a priority over the past five years, there exists a significant discrepancy between the security measures thought to be necessary by the IT administration and the actual safeguards that were currently in place, or even planned for implementation. Research also identified several of the challenges that have caused this discrepancy between network security needs and those already in place. These included lack of staff, lack of funding, and lack of upper management support.

INTRODUCTION

Since 2001, campuses and corporations around the world have been fighting viruses and hackers more than ever. Viruses are sent via email, web sites, or downloaded from the Internet. When a virus hits a computer today, it can spread around the world in a few minutes. On September 18, 2001, the Nimda worm infected more than 2.2 million servers and personal computers within 24 hours which resulted in $539 million in downtime and subsequent cleanup costs (Esche, 2004). As another example, in 2003, SQL Slammer spread to computers all over the world in a matter of ten minutes with a reduction to Internet accessibility by 20%. Seventy-five percent of colleges and universities were affected with this virus and this outbreak cost millions of dollars in staff resources and recovery of data and was more than the technology staff was prepared to handle (Updegrove & Wishon, 2003).

First among the targets in cyberspace are colleges and universities because they possess a vast amount of computer power most of which contain resources with open access. Hackers attack educational institutions due to their perception, which is often accurate, of lax security that is coupled with high capacity computer systems. Research lab servers and workstations as well as student computers in dorms are not typically managed by IT staff thus opening those resources for hackers and viruses (Updegrove & Wilson, 2003). The NSSC has acknowledged these open resources and is in the process of ensuring that higher educational institutions become more secure (National Strategy, 2003).

Top management in higher educational institutions has become aware of the cost and downtime that are associated with network security breaches. The progress being made to improve security by these institutions is partly a result of increased interactions between campus technologies and security officers, conferences sponsored by educational institutions, campus security days, and especially collaborations with the EDUCAUSE/Internet 2 Computer and Network Security Task force (Bruhn & Peterson, 2003). A 5-point Framework for action has been adopted by top university presidents agreeing to make IT security a high priority in their institutions and this commitment includes adoption of the following policies and measures:

1. Make IT security a priority in higher education
2. Revise institutional security policy and improve the use of existing security tools
3. Improve security for future research and education networks
4. Improve collaboration between higher education, industry, and government

5. Integrate work in higher education with the national effort to strengthen critical infrastructure (National Strategy, 2003, p.40)

Four guidelines have been established by the NSSC to help colleges and universities secure their computer networks. These include: (1) dealing with cyber attacks and vulnerabilities with Information Sharing and Analysis Centers; (2) guidelines to empower Chief Information Officers in addressing cyber security; (3) IT security best practices; (4) model user education programs and resources (National Strategy, 2003, p. 41).

Despite the recommendations from NSSC, academia has typically treated Information Security as an after-thought. It is an area that has been left for the Information Technology departments to worry over and pay for out of their already thinly stretched budgets. For years it has been the responsibility of overburdened system administrators or general IT staffers to make sure that the campus was safe and secure (Recor, 2003). IT staff are faced with the daunting task of securing data while allowing an open and convenient environment in academia. The balancing act between convenience, openness and security measures is the biggest challenge for academia. Campus policies and procedures must have the hierarchical support in place for a successful and organized security infrastructure (Recor, 2003). According to Bruhn and Peterson (2003), a particularly unique problem to secondary educational institutions is the fact that tenured faculty view IT security as a hindrance. These faculty also value autonomy and privacy and do not want to have any demands put on their institutional resources or time. The academic arena consisting of faculty, deans, and researchers view proactive security measures as bureaucratic.

Colleges run into a plethora of barriers when it comes to the implementation of network security. A large barrier for any organization would be the lack of financial resources. IT management must take numerous items into consideration for network security which can become very expensive. Some considerations should include physical aspects, training, hardware, and software. Staffing the IT department becomes a barrier after the purchase of the hardware and software. An institution may have hardware and software purchased for network security but the IT staff may not have the skills necessary or the time to install and maintain the resources. When management does approve funding for the equipment, the next barrier would be security training for the installation of equipment to make it work efficiently (Daniels, 2001).

Information Technology Departments should always plan to have a budget for security training for their employees sufficient to keep up with all the security risks. Education is an institutions strongest asset in the defense of a hackers attack (Daniels, 2001). Security training can be very expensive which leads back to the first security barrier - a lack of funding. Another barrier that can cause a problem for the IT department in protecting the campus is lack of support from the administration. If the administration does not place network security as a top priority of an institution then the staff and training will not be funded, which can have disastrous results for a campus. One security breach is all it takes for an institution to realize the importance of investing in network security (Hayes, 2001).

METHODOLOGY

The purpose of this study was to determine if a perceived difference exists between current network security measures and those considered to meet an acceptable minimal level. Also, to determine the major challenges that IT managers on college campuses were facing when attempting to bring network security measures up to what they believed were minimally acceptable standards. To effectuate this purpose, a survey was sent to colleges in seven southern states. The Chief Information Officer, Director of Computer Service, or similar individual was the targeted respondent for this study.

The questionnaire required the respondent to assess the current status of network security on their respective campus in a number of specific areas. They were also asked to respond as to whether or not they believed this status to meet a minimally acceptable level of security. Finally, when a gap between existing security levels and perceived minimums were noted, they were asked to determine the major reason for the difference and whether or not plans were underway to bridge this gap.

RESULTS

Of the 134 colleges surveyed, forty-five usable questionnaires were returned. This rendered a response rate of 34%. When asked "Is your organization where it should be with network security?" 26% of the respondents answered yes. The remaining 74% indicated a gap between what they believed necessary for network security and what was current in place or under development. Most indicated that there were no immediate plans to bridge these gaps at their institution. When asked to respond as to why the respondents believed that their institution did not have adequate network security in each of these areas, the overarching explanations were lack of funding and the fact that upper management did not feel that IT security was a priority issue. While over 70% of respondents reported their Information Technology budget was over $200,000 (Table 1), over 35% indicated that less than 5% of that budget was dedicated to security (Table 2).

Institutions Information Technology Budgets

Dollar Range	Frequency	Percentage
$0-49,999	2	3.9%
$50,000-99,999	4	7.8%
$100,000-149,999	4	7.8%
$150,000-199,999	5	9.8%
$200,000 +	36	70.6%

Table 1

Percentage of Budget Devoted to Network Security

Percentage	Frequency	Percentage
0-4.99	16	36%
5-9.99	11	24%
10-14.99	9	20%
15-19.99	2	5%
20 +	7	15%

Table 2

While most spend less that 5% of their IT budget on security, this research determined, at two-year colleges, 72.5% of Informational Technology staff spends about 0-20% of their workday dealing with security issues. These issues include updating software, monitoring firewalls, monitoring viruses, monitoring network activity and other similar measures. Also, the majority of these institutions (41.2%) do not have anyone who has had at least a year of security training. The lack of training is due to numerous factors such as lack of funding, lack of staff, and lack of senior administrative support.

Administration has to get behind the IT staff and find the resources needed to get the security expertise on their campus. Payne (2003) discussed having a security training tailored for the administrators due to their decision making for allocating resources. Network security should be marketed in business terms to the administrators to show how the institution as a whole can be affected by security breaches (Payne, 2003). Another article states that a computer service department does not have any protection, creditability, or funding for security initiatives if the management does not get behind the department (Hayes, 2001).

Colleges are facing a real crisis in IT security practices. Some administrators are already addressing these issues but at a tragically slow rate. The majority of current administrators have reported a lack of support from other administrative positions and barriers such as excessive time requirements and stress which may prevent them from incorporating policies that would more securely protect their campus network. In the area of funding, Daniels (2001) in his article "The Weakest link…This is not a game" discusses the lack of financial resources as being a large barrier for network security at an institution. With two-year colleges on such tight budgets, steps need to be taken to try and stay on top of the security issues that arise. Some steps might include charging technology fees designated specifically to security issues or seeking government or private grants to aid in security initiatives. Regardless of the steps taken, security must be placed at a higher priority and funded appropriately. If not, campuses will stay a major target and remain one of the weakest links in the computer security chain.

REFERENCES

[1] Bruhn, M., & Peterson, R. (2003, November/December). Planning for improved security. *Educause Review*, 98-99.

[2] Daniels, Jack. (2001). *The weakest link…This is not a game*. Retrieved from http://www.sans.org/readingroom/whitepapers/basics/440.php.

[3] Educause (n.d.b). *About educause*. Retrieved June 24, 2006 from http://www.educause.edu/about.

[4] Esche, H. (2004). The cost of worms? Retrieved from http://www.ucalgary.ca/InfoServe/Vol8.8/worms.html

[5] Hayes, H. (2001, June). Security training checklist. Retrieved from http://www.sans.org/readingroom/whitepapers/basics/440.php

[6] National Strategy to Secure Cyberspace. [www.securecyberspace.gov].2003.

[7] Payne, S. (2003). Campus wide security education and awareness. In M. Luker & R. Peterson (Eds.) Educause leadership strategies: Vol. 8. *Computer and network security in higher education* (pp.31-44). San Francisco, CA: Jossey-Bass.

[8] Recor, J. (2003). Organizing for improved security. In M. Luker & R. Peterson (Eds.) Educause leadership strategies: Vol 8 *Computers and network security in higher education* (pp15). San Francisco, CA:Jossey-Bass.

[9] Updegrove, D. & Wishon, G. (2003). In M. Luker & R. Peterson (Eds.) Educause leadership strategies: Vol. 8. *Computer and network security in higher education* (pp.xi-xiv). San Francisco, CA: Jossey-Bass.

BUG WARS: A COMPETITIVE EXERCISE TO FIND BUGS IN CODE*

Renee Bryce
Utah State University
Computer Science
Renee.Bryce@usu.edu

ABSTRACT

Software bugs are a common problem that students encounter in any Computer Science program. "Bug Wars" is a fun and competitive class exercise for student teams to identify bugs in code. To prepare for the competition, the instructor provides several code examples that contain bugs. Each student team also develops code that has a bug. All of the code examples are placed on a table in the classroom at the beginning of class. The competition then begins by each team taking one problem to solve and checking with the author of the respective code to ask whether they correctly identified the bug. If they solve the bug, we update the score and they swap the problem that they solved for a new problem. The team that identifies the most bugs wins the competition. The majority of students reported that this activity increased their interest in software testing and made them more aware of bugs that they should avoid on future assignments.

1 INTRODUCTION

Students in introductory programming courses often struggle with programming bugs. For instance, we conducted a small survey of students from our Computer Science I course in two separate semesters so that we could understand their frustrations as a CS1 student. A total of 92 students completed the survey. We asked the open-ended question of whether anything in the course was frustrating and if so, what the most frustrating issue was for them. Programming bugs and debugging were the biggest issues reported by 38%

---

of the students. This motivates our work to create fun activities that help students to think about programming bugs so that they can avoid them on future assignments.

Locating bugs is often more difficult than correcting bugs [7, 8]. Software testing is one way to help students with bugs. For instance, Edwards examines the relationship of student performance and how well they test their programs [5]. Another topic in software testing is that of Test Driven Development (see [4] for one example study). Cole et al. propose teaching students to use static analysis to find bugs [2]. Fleming et al. have more advanced students work on corrective maintenance of concurrent software [9]. Williams suggests defect prevention [11]. Bryce et al. collect data about student programming bugs, share the data with students and instructors, and create on-line movies and games about the most common bugs [1]. Elbaum et al. teach students about software testing using an interactive web application called Bug Hunt [6]. Our work here differs as we make software testing a team-based activity in a fun, competitive environment by having students work in teams to (1) create code that contains one bug for others to solve and (2) competitively locate bugs in problems that are created by the instructor and other student teams. Our activities emphasize working in teams since many studies such as pair programming demonstrate that students often enjoy working together. Another example by DeClue et al. reinforces that a sense of community among students is helpful [3].

In the remainder of this paper, Section 2 describes our "Bug Wars" activity, Section 3 provides example problems, Section 4 discusses results of student feedback on the activity, and Section 5 concludes.

2 BUG WARS

The activity spans over two 70 minute class sessions as follows:

Class session 1: In the first class session, students form teams of size 3 to 4 students. Each team then develops code that contains one bug and a solution key. They submit these at the end of the class period. After class, the instructor reviews the problem and solution key from each team to ensure that they are correct and usable for the remainder of the activity. The instructor then creates several additional problems of buggy code that complement those that the students created.

Class session 2: The class session begins by the instructor placing all of the papers with buggy code on a table in the classroom. Each team then takes one problem. Once a team believes that they solved a problem, they check with the author of the code to confirm whether they correctly identified the bug. If they answer correctly, they receive a point and swap their problem for a new one. If they do not answer correctly, they are also allowed to swap for a different problem at any time. The team that locates the most bugs wins.

3 EXAMPLE PROBLEMS

This activity may be used in any programming course, but we used it in a CS3 course. The problems should be tailored to material in the specific course. When we conducted this activity, students were learning about recursion. Therefore, the example problems that the instructor created involved recursion and code that had been reviewed

in class and their textbook, "Data Structures and Other Objects Using Java" [10]. For instance, our first problem in Figure 1 includes a bug in the Mergesort algorithm. This code is modified from the the course textbook [10]. We told the students to assume that the "merge" method works correctly and that the bug is in the "mergesort" method. The bug is on line 11. The starting index for the right half of the array is incorrect.

```
// Assume that the merge method works correctly as we
// reviewed in class
1.      public static void mergesort(int[] data, int first, int n)
2.      {
3.          int firstHalf;
4.          int secondHalf;
5.          if (n > 1)
6.          {
7.              // Compute sizes of the two halves
8.              firstHalf = n / 2;
9.              secondHalf = n - firstHalf;
10.             mergesort(myArray, first, firstHalf);
11.             mergesort(myArray, first, secondHalf);
12.             merge(myArray, first, firstHalf, secondHalf);
13.         }
14.     }
```

Figure 1: Example 1: Bug in the MergeSort Merge algorithm (The bug is on line 11, the starting index for the right half is incorrect)

The second example in Figure 2 is also code that students are familiar with from lecture and their textbook [10]. In this problem, the method takes an integer value and prints each digit of the number on an individual line. We inserted a bug on line 6. The wrong value is passed to the recursive call and can easily be found if the students trace the method.

```
// This code is supposed to take a number and print it vertically.
// For instance, an input of "1234" should result in the output:
// 1
// 2
// 3
// 4
1.      static void writeVertical(int num)
2.      {
3.              if(num10)
4.                      cout << num << endl;
5.              else{
6.                      writeVertical(num%10);
7.              cout << (num%10) << endl;
8.              }
9.      }
```

Figure 2: Example 2: Bug in the formula that computes the value inside of the recursive call (The bug is on line 6, the mod should be divide)

The third example in Figure 3 is supposed to compute the factorial of a number using a recursive method. We inserted a bug on line 6 in which we sum the values instead of multiplying.

```
// This code is supposed to compute the factorial of a number
1.      public static int factorial(int n)
2.      {
3.              if(n == 0) {
4.                      return 1;
5.              }
6.              else {
6.                      return n + factorial(n-1);
7.              }
8.      }
```

Figure 3: Example 3: Bug in the return statement (The bug is on line 6, the '+' should be '*')

Finally, some students in the course were still weak on the recently covered topic of recursion, so we gave an easier problem in order to boost everyone's confidence that they could solve at least one problem. In the example shown in Figure 4, the code should compute the sum from 1 to n. A common bug that was brought to office hours during this particular semester was that students were off-by-one in their loops. The bug in this

problem is on line 4 and includes an off-by-one bug. Further, we gave a bonus point if the students could improve the efficiency of the code. Earlier in the semester, we talked about this example and how a formula could replace the loop.

```
// This code is supposed to calculate the sum from 1 to n.
// Bonus: The code has a bug for you to find. Is there a way to
// implement this more efficiently without a loop?
1.      public static int calculateSum(int n)
2.      {
3.              int totalSum = 0;
4.              for(i=1; i¡n; i++)
5.                      totalSum += i;
6.              return (totalSum);
7.      }
```

Figure 4: Example 4: Bug in a for-loop (The bug is on line 4, the loop is off-by-one)

4 RESULTS

At the conclusion of the activity, we administered a survey to students and then discussed the results of the survey. This section summarizes the results and class discussion.

4.1 Student survey

The survey was optional, but all 26 students completed the survey. Table 1 shows the results. The activity increased interest in software testing for 77% of the students. Approximatley 92% of the students said that the activity increased their awareness of bugs that they should avoid on future assignments. Indeed, this is important as other researchers report that students often have more difficulty locating bugs rather than fixing them [7, 8]. All of the students with the exception of one recommended that we hold this activity again in the future.

Survey Question	No. of students that answered "Yes"(out of 26 students in class)
Did this activity increase your interest in Software Testing?	20
Did this activity increase your awareness of bugs that you should avoid on future assignments?	24
Would you recommend this activity in the future?	25

Table 1: Summary of survey questions

In addition to the three questions above, we asked students to provide any comments that they had. All of the comments were positive and include:

- This was fun! We should do it more often.
- I've made so many bugs that I was able to think of really good ones for this game.
- I like working in teams and class time went by fast.
- We need more time!
- I would have made our bug tougher after seeing some of the other team's bugs.
- Thanks! (This comment was made four times.)
- We should be able to put more than one bug in the problem. Muhaha!
- Can we do this again?

4.2 Class discussion about the survey results

During a class discussion about the activity, we asked the students why the activity increased their interest in software testing. Several students said that they chose this answer because they had fun working with their team members to create the most difficult buggy code that they could for the other teams. They also said that it was interesting to work as a team to find the bugs that other teams created. Students also said that testing their own code was "boring", but that testing code that they did not write on their own was fun! We also asked how the activity increased their awareness of bugs that they should avoid on future assignments. Students volunteered that they experienced many of the bugs already, but that the bugs related to the stopping cases for recursion or the values passed through the recursive methods were tricky. Finally, we asked why all of the students recommended the activity for future. Many students said that it was fun to work in a team. Many agreed that they enjoyed the competition. One student said that it is helpful to see the bugs that you may make before you make them. Many of the students in the class agreed. Another student said that we should do this exercise before every homework assignment so that they could guess the bugs that they would make on the assignment before it was due. Many student enthusiastically supported this. I concluded the discussion by asking the students if the following was a fair summary and they all agreed: "This activity was fun because students enjoyed working in teams. The skills and information that they learned from the exercise is helpful because it strengthens their

ability to avoid programming bugs in the future. Further, they recommend the activity in the future for both their class and other programming classes."

5 CONCLUSIONS

Previous research suggests that students often have more difficulty in locating bugs than fixing them. "Bug Wars" is an activity that makes it fun for students to think about bugs that they could encounter. A brief overview of the activity is as follows: (1) student teams create a problem with code that has exactly one bug and an answer key, (2) the instructor reviews the student code to make sure that there is indeed only one bug, (3) the instructor adds additional problems and buggy code to the pool of problems for students to solve, (4) the instructor shares the problems by placing them on a table in the classroom, (5) each team may take one problem at a time to solve, (6) for each bug that a team identifies, their score increases by one point, and (7) students may swap for a new problem at any time. This activity gives students the opportunity to brainstorm "difficult" bugs when their team creates their problem and code that contains a bug since they want to make it tough for the other teams to solve. It then gives students the opportunity to test code. Our students reported that it is often frustrating finding bugs in their own code, but it was fun to find bugs in code that others wrote. The majority of students reported in a survey that this activity increased their interest in software testing and that it increased their awareness of bugs to avoid on future assignments. All of the students recommended this activity for the future. While we initially meant for this question to ask whether they recommended this activity for future students, the students clarified in a class discussion that they also wanted to repeat this activity for each of their remaining homework assignments so that they could learn about bugs that they should avoid on those assignments as well.

REFERENCES

[1] R. Bryce, A. Cooley, A. Hansen, and N. Hayrapetyan. A one year empirical study of student programming bugs. In *Proc. of Frontiers in Education*, pages F1G-1-7, 2010.

[2] B. Cole, D. Hakim, D. Hovemeyer, R. Lazarus, W. Pugh, and K. Stephens. Improving your software using static analysis to find bugs. In *Proc. Of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 673-674, 2006.

[3] T. DeClue, J. Kimball, B. Lu, and J. Cain. Five focused strategies for increasing retention in computer science 1. *Journal of Computing Sciences in Colleges*, 26(5):252-258, 2011.

[4] V. Desai. Implications of integrating test-driven development into cs1/cs2 curricula. *ACM SIGCSE Bulletin*, 41(1):148-152, 2009.

[5] S. Edwards. Improving student performance by evaluating how well students test their own programs. *Journal on Educational Resources in Computing (JERIC)*, 3(3):1-24, 2003.

[6] S. Elbaum, S. Person, J. Dokulil, and M. Jorde. Bug hunt: Making early software testing lessons engaging and affordable. In *Proc. the International Conference on Software Engineering (ICSE07)*, pages 688-697, 2007.

[7] S. Fitzgerald, G. Lewandowski, R. McCauley, L. Murply, B. Simon, L. Thomas, and C. Zander. Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education Journal*, 18(2):93-116, 2008.

[8] S. Fitzgerald, R. McCauley, B. Hanks, L. Murphy, B. Simon, and C. Zander. Debugging from the student perspective. *IEEE Transactions on Education*, 53(3):390-396, 2008.

[9] S. Fleming, E. Kraemer, R. Stirewalt, S. Xie, and L. Dillon. A study of student strategies for the corrective maintenance of concurrent software. In *Proc. of the International Conference on Software Engineering (ICSE) - Education Track*, 2008.

[10] M. Main. *Data Structures and Other Objects Using Java, 2nd edition*. Addison Wesley, New York, 2002.

[11] L. Williams. Instilling a defect prevention philosophy. In *Proc. of Frontiers in Education*, pages 1308-1312, 1998.

A CLOSER VIEW OF TWO TECHNOLOGIES USED IN e-LEARNING*

Marcos S. Pinto
NYC College of Technology, CUNY
300 Jay St., Brooklyn, NY 11201
mpinto@citytech.cuny.edu

ABSTRACT

The objective of this paper is to evaluate the efficiency of learning by knowledge construction and sharing of experience. Students are taught how to build an e-Learning system by using either an ontology or a topic map. The paper contrasts these two e-Learning technologies to analyze their impact on the development of a pedagogical content under a granular form represented by ontology of the basic concepts of communication networks. During the building of the e-Learning systems, it becomes clear that the role of the ontology is to reduce the overall complexity of the creation process by assuring the necessary comprehension of customized learning requests as well as reusability and sharing for all actors involved.

1. INTRODUCTION

The quick pace of the development in information and communication technology is providing the learner with applications that permit distance education through Internet where asynchronous communication is allowed in space and in time between students and the academic staff. Thus students have maximum flexibility in adapting their studies to their own schedule thereby consolidating a growing dynamic online community [15]. This e-Learning experience has proved to be an effective way of delivering learning materials to previous unreachable students under difficult circumstances and unavailable access and presentation methods.

An online learning system is a "collaborative learning" system in which the actors, experts and students, at various performance levels work together toward a common goal of learning by reusing, sharing experience and building knowledge together within a particular domain [12]. The "collaborative learning" is only possible when the actors use a common terminology such that a given word or expression have the same meaning for everyone [13]. It is one of the reasons that e-Learning systems are often based on ontologies. Ontology development is critical to the creation of successful knowledge-based systems [17].

This paper contrasts the use of ontology with that of topic maps in building an online learning system. Undergraduate students of a networking class were motivated in experimenting with both technologies, a topic map and a ontology editor, in order to answer framed research questions such as what networking concept students find it difficult, how we can measure student's understanding of these concepts, why these concepts are difficult, and how we can design an e-Learning environment to help students learn more easily concepts that are difficult. It was then decided to start with the basic concept of the OSI (Open System Interchange) model. The difficulties involved in ontology development, such as the problems of building representations of a domain, and the involvement of parties from diverse backgrounds, were overcomed through collaboration and focused effort.

2. ONTOLOGY-DRIVEN E-LEARNING SYSTEM

The use of ontologies in education follows several goals. First, although an ontology is different from a topic map in may aspects, they both represent a knowledge structure. Second, an ontology may support reasoning for diagnosis of causes of learner's mistakes and misconceptions, which is a relevant functions of student diagnosis module. Moreover, each item of the ontology may be supplied with references to corresponding learning objects which may be used by students to correct their mistakes. Third, an ontology can represent not only definite concepts and semantics of their relationships but also all synonyms of both. This may rise flexibility and adaptability of knowledge assessment allowing students to use synonyms. And last, but not least, at the moment on the Internet there are available quite a lot ontologies that correspond to taught subjects. Their usage may help teachers who are creating courses to reach compatibility of the knowledge structure they wanted to create with corresponding ontology. Our experience confirms that for teachers it is much easier to edit a topic map generated from an ontology or even to build it from scratch instead of mastering formal ontology languages and specific tools for ontology construction [7].

Learning object metadata and domain ontologies (i.e. taxonomies) are often defined at different ontological levels. The role played by ontology in the construction of our learning object, OSI model, can be summed up in the following points: common knowledge-base of different concepts, shared terminology, extensibility and reusability of concepts, possibility to manage different roles and their information access, possibility to model the training object through its parts and relationships, and context-based definition of the verbs

3. ONTOLOGY DEVELOPMENT

The steps to developing an ontology, see figure 2, include: (1) defining classes; (2) arranging the class into subclasses within a hierarchy; (3) defining slots or properties and the possible value labels for them; and (4) documenting specific instances using the slot value labels.

The usual problems with the creation of an ontology involve content conceptualization and classification, and identifying and naming relationship between the concepts. Indeed, since there is no rule for determining the "best" criterion for classifying topics, we made subjective decisions on how classes could be broken down into subclasses. The main class is named OSImodel which has two subclasses: NetworkLayer and NetworkProtocol. The Network Layers class has seven subclasses which are the seven layers of the OSI model. The Network Protocols has twenty five subclasses of protocols belonging to the network layers. The object properties and their corresponding inverse were defined as: hasPart, isPartOf, hasProtocol, isProtocolOf, isPlacedAtopOf, and isBelowOf. The property hasPart has the OSImodel as its domain and the two subclasses NetworkLayer and NetworkProtocol as its range.

At the same time as ontology languages have been developed, tools have emerged for creating, editing and managing ontologies written in the various languages. Protégé 2000 is an ontology editor that was developed at Stanford University as an Open Source editing environment where ontologies can be constructed through a graphical user interface. It can handle ontologies in XML, RDF(S), XML Schema, DAML+OIL and OWL.[6]

Figure 1 shows our ontology of the OSI model created using Protégé's interface. In this example, two instances of the class "OSImodel" have been defined: NetworkLayer and NetworkProtocol. Any "slots" (attributes) can be defined - in this case, each instance has a name and three potential relations to other instances (hasPart, hasProtocol, and isPlacedAtopOF). "Forms" allow the user interface for entering ontology information to be manipulated for ease of use, and "queries" allow the ontology to be searched based on the attributes of instances.

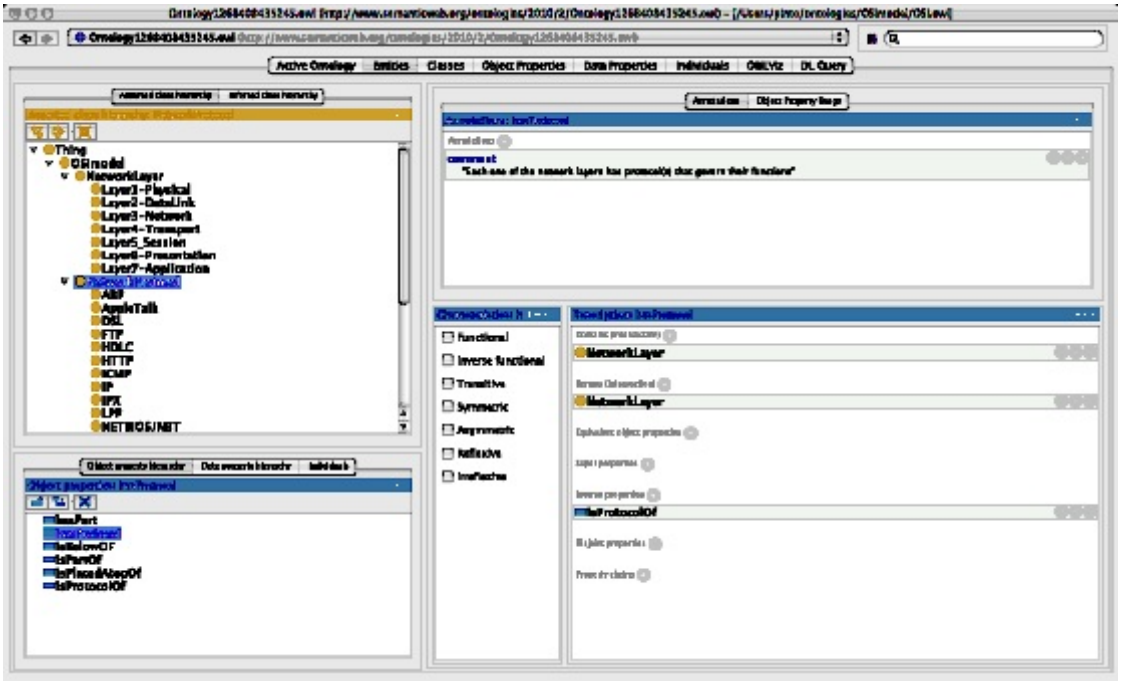


Figure 1: OSI Model Ontology - Classes and properties

The OWLviz plugin (Figure 2 and Figure 3) allows ontologies to be viewed graphically; visualization tools such as this can be helpful in the development process. These figures reflect the classes and subclasses that were selected by the majority of the students with the help of the instructor. The agreement reflects the aspect of ontology building with a common knowledge-base terminology.

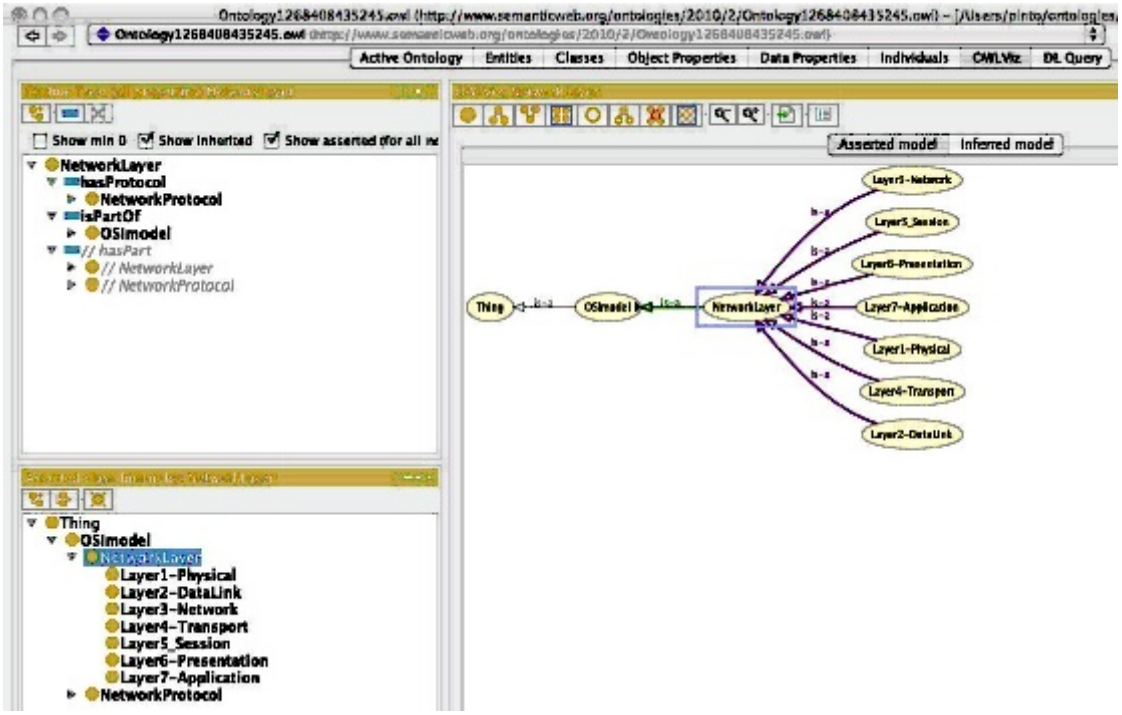


Figure 2 - Visualization of the NetworkLayer class in Protege

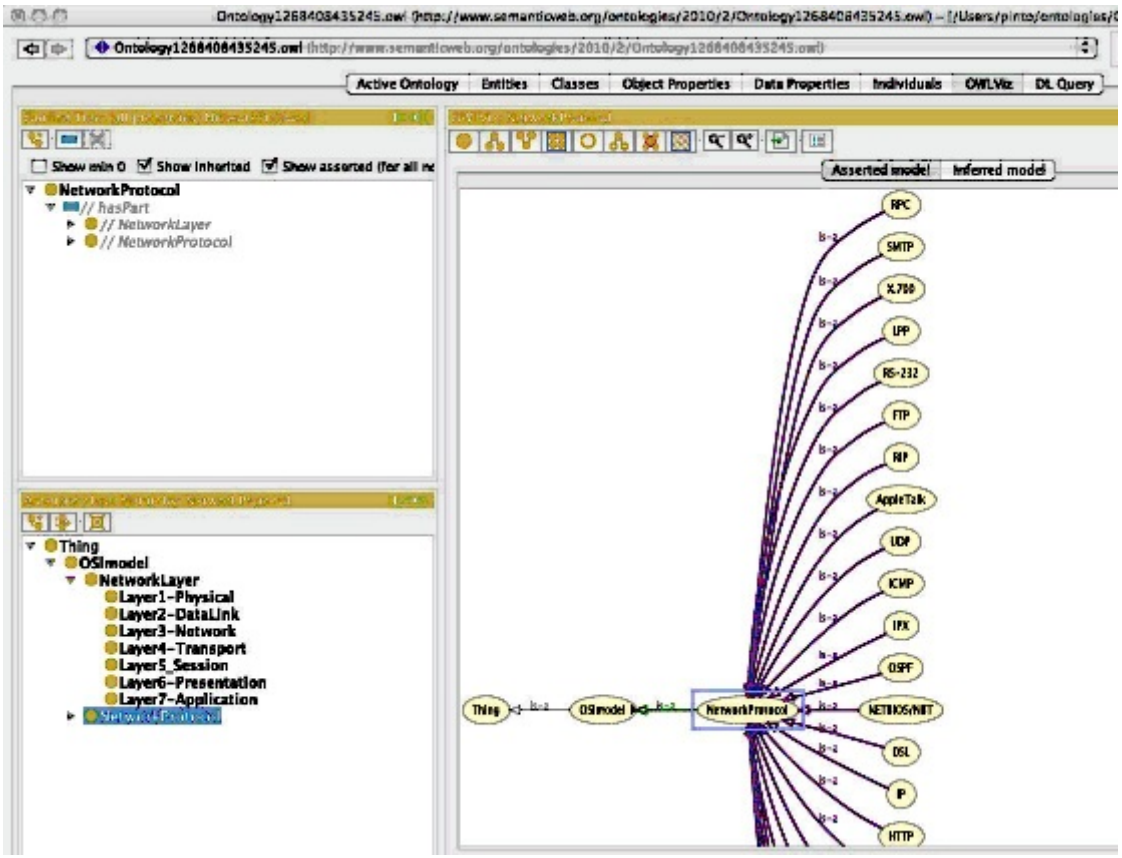


Figure 3. Visualization (partial) of the NetworkProtocol class in Protege

4. THE DESIGN OF OSI MODEL USING TOPIC MAP

Creating a topic map is easy once you classify the subject into topics and subtopics [1]. Topics are first fed into the software according to their hierarchy and their relationships (associations). Topics are then linked to resources (occurrences). Topic map is saved as a .xtm file and it is created according to the Topic Maps Reference Model [3]. We are using the TM4L editor [2] for creating our topic map for the OSI model.

The topic is OSI and subtopic structure is broken down into components (Physical, Data Link, Network, Transport, Session, Presentation, Application) and protocols (Physical Layer Protocols, Data Link Layer Protocols, Network Layer Protocols, Transport Layer Protocols, Session Layer Protocols, Presentation Layer Protocols, Application Layer Protocols).

The next thing is to create the associations (relationships) between the topics [14]. We made use of two associations: the whole-part and the placed-atop. The former is of the type includes/part-of-it, and the latter is of the type placed-atop-of/placed-below-of.

The whole-part is a typical relationship of topics that are related to each other hierarchically. The seven layers of the OSI model communicate with the adjacent layers,

therefore the association placed-atop-of would recognize this situation. For example, it can be formulated that the topic Application Layer is part of the OSI model and it is place on top of the Presentation Layer which is also part of the OSI model. The associations of whole-part (instance-of) and placed-atop are defined for each of the seven layers of the OSI model.

5. EVALUATION AND CONCLUSION

The majority of the students involved in this study had the same opinion that all the work involved in creating an ontology directly benefited and helped them to visualize and comprehend the relationships between concepts in their domain. This triggered associative ways of processing, reflecting and analyzing the information at hand. There was no students that had difficulty in applying either one of the technologies to create their own e-Learning system. The amount of valid content was the basis for the instructor to evaluate each individual e-Learning system developed by the students.

Topic maps are a form of semantic where the knowledge base is driven by visual features. These characteristics support our idea of providing an interpretive, semantic layer on top of document collections that classify these documents according to scope, context and constraints [18]. The TM4L Editor benefits from the Topic Maps' fundamental feature to support easy and effective merge of existing information resources while maintaining their meaningful structure. This allows for flexibility and expediency in re-using and extending existing repositories. The learning content created by the editor is compliant with the XML Topic Maps (XTM) standard and thus interchangeable and interoperable with any standard TM tools.[5]

Clearly, ontologies have a role to play in learning and teaching on the Web. The difficulties involved in ontology development, such as the problems of building representations of a domain, and the involvement of parties (students and faculty) from diverse backgrounds, were overcome through collaboration and focused effort.[8]

The two technologies, an ontology editor and a topic map editor, can be thought as alternate methods to create e-Learning systems. The distinction of these technologies is on the availability of reasoners for ontology editors that facilitates the discovery of new semantics whereas the discovery would be the result of individual or cooperative work on the analysis of topic maps. Despite of this difference, both technologies face many problems such as what is the best way to organize the ontology and ontology mapping repositories, and what kind of appropriate reasoning services over those ontologies and mappings are needed [16]. We plan to extend our system so students are able to register database tables as well as XML documents and make it part of a generic framework for semantic registration of scienti?c data. Moreover, the approach will also be applied to any computer systems courses and to college students at any level of study.[10]

6. REFERENCES

[1] ISO (2005) “ISO/IEC CD 13250-5 Topic Maps — Part 5: Reference Model”. Online at http://www.isotopicmaps.org/TMRM/TMRM-5.0/TMRM-5.0.pdf

[2] TM4L: Topic Maps 4 e-learning. http://compsci.wssu.edu/iis/nsdl

[3] Dicheva, D. et al. (2007): Authors support in the TM4L environment. *Information Technologies and Knowledge* **1**(3):215-218.

[4] Dichev C. et al. (2004). Using topic maps for web-based education. *Int. J. of Advanced Technology for Learning*, volume 1(1), pages 1–7, 2004.

[5] Berners-Lee, T. Hendler, J. & Lassila, O., The Semantic Web. *Scientific American*, 17 May 2001.

[6] E.Oguejiofor, et. al., (2004). Intelligent tutoring systems: an ontology-based approach. *International Journal of IT*, Vol. 2, Issue 2: 115-128

[7] W. Huang, D. Webster, D. Wood, T. Ishaya (2006). An intelligent semantic e-learning framework using context-aware Semantic Web technologies. *British Journal of Education Technology*, Vol. 37, No. 3: 351-373.

[8] L. Sanchez-Fernandez, N.Fernandez-Garcia (2005). The Semantic Web: Fundamentals and A Brief State-of-the-Art. *Upgrade*, Vol. 6, No. 6: 5-11

[9] C. Pahl, E. Holohan (2009). Applications of Semantic Web Technology to Support Learning Content Development. *Interdisciplinary Journal of E-Learning and Learning Objects*, Vol. 5: 1-25

[10] E. Moreale, M. Vargas-Vera (2004). Semantic Services in e-Learning: an Argumentation Case Study. *Educational Technology & Society*, 7(4): 112-128

[11] Carroll, J. (2002). Human-computer interaction in the new millennium. Addison-Wesley.

[12] Giarratano, J. (2005). *E*xpert systems: principles and programming, Thomson.

[13] Hart, A. (1992). Knowledge acquisition for expert systems. McGraw-Hill.

[14] Hatzlygeroudis, L., & Prentzas, J. (2004). Using a hybrid rule-based approach in developing an intelligent tutoring system with knowledge acquisition and update capabilities. *Expert Systems with Applications, 26*(4), 477-492.

[15] Lawler, R. W., & Yazdani, M. (1987). Learning environments and tutoring. London, UK: Ablex Publishing Corporation.

[16] Luger, G. (2005). Artificial intelligence: Structures and strategies for complex problem solving. Addison-Wesley.

[17] Macmillan, S. (2009). An architecture for a self-supporting instructional planner for intelligent tutoring system. *Computational Intelligence, 3*(1), 17-27.

[18] Riesbeck, C. et al. (1998). Authorable critiquing for intelligent educational systems. *International Conference on Intelligent User Interfaces.* San Francisco.

WUMPUS WORLD IN INTRODUCTORY ARTIFICIAL INTELLIGENCE*

Daniel Bryce
Utah State University
42055 Old Main Hall
Logan, UT 84341
(435) 797-8841
daniel.bryce@usu.edu

ABSTRACT

With an emphasis toward hands-on learning of Artificial Intelligence (AI) concepts, we present an overview of an introductory AI course that uses a series of five projects based in the game of Wumpus World. The advantage of using a single domain and corresponding simulation environment for each project is that students can better focus on the course material after a brief familiarization period. The projects involve students programming search algorithms, satisfiability algorithms, declarative planning domain descriptions, and Bayesian network inference algorithms applied to Wumpus World. We describe the course, Wumpus World, and each of the projects.

INTRODUCTION

Introductory Artificial Intelligence (AI) courses vary significantly in content, surveying the broad set of topics in the field. The number of possible topics is so large that assigning students a project on each is simply not feasible. Moreover, expecting students to implement algorithms within a number of different software harnesses is inappropriate because their time should be spent on the algorithms and not struggling to learn new source code. We describe a course designed around a single software harness for the Wumpus World (WW) game and the accompanying projects.

The introductory AI course is open to both undergraduates and graduates, and is considered a project intensive course. Graduate students have to opportunity for graduate

---

credit by completing, in addition to the five software projects, a self-directed project of their choice. The course teaches selected chapters from "Artificial Intelligence: A Modern Approach" [7], the most widely adopted text for teaching AI. The course subjects include: uninformed search, informed search, constraint satisfaction, game playing, propositional logic, first-order logic, automated planning, probabilistic inference, Markov decision process, and machine learning. The four course projects exercise search, propositional logic, automated planning, and probabilistic inference, and are set within Wumpus World.

Wumpus World is based on the computer game Hunt the Wumpus [10], which involves a treasure-seeker navigating a cave while avoiding pits and wumpii to collect gold. The dark cave challenges the player's senses by limiting their ability to identify pits, wumpii, and gold to indirect observations of breeze, stench, and glitter. The player also has a bow and single arrow, which they can use to shoot and possibly slay a wumpus. The game provides ample opportunity to illustrate AI concepts, as in the following projects:

- Project 0 asks the students to program a WW player (an agent) by any technique that they wish so that they can attain familiarity with the domain and simulator.
- Project 1 teaches the fundamentals of uninformed and informed search, including search infrastructure (e.g., successor functions, search nodes, goal tests, etc.), different search algorithms (e.g., depth-first, A*, best-first, etc.), and heuristic functions (e.g., straight-line distance, Manhattan distance, etc.). The project requires students to program each of the algorithms and heuristic functions using an existing search infrastructure that is specific to WW.
- Project 2 teaches inference in propositional logic by requiring students to convert logical rules about WW to Conjunctive Normal Form and implement three inference algorithms: resolution [6], DPLL [2], and WalkSAT [9]. The students are given a list of observations about WW, such as the locations of breeze, stench, and glitter, and asked if a particular location contains a pit, a wumpus, or gold. The students answer the queries by logical inference using the three algorithms and discuss the results.
- Project 3 combines knowledge representation, first-order logic, and automated planning by tasking students to declaratively specify a planning domain theory in a first-order planning language called PDDL [4]. The students use a number of existing planners to solve several WW problems that they formulate and compare the performance of the planners both empirically and analytically.
- Project 4 requires students to answer several probabilistic inference queries, such as the probability that a WW location contains a pit given several observations. The students must answer the queries with two Bayesian network algorithms: enumeration and variable elimination.

In addition to the projects noted above, graduate credit is awarded for self-directed projects, many of which have been applied in WW. We continue with a more detailed overview of the course, a description of Wumpus World, information on each of the projects, related work, and a conclusion.

COURSE OVERVIEW

Introduction to AI seeks to cover several of the most important topics in AI and prepare students, in the case graduate students, for later courses in AI. The course is approximately forty-five meetings, with several small assignments and quizzes, two exams, and the five projects. Table 1 lists the topics covered by week, and the duration of the projects. The projects are designed to focus on the simplest topic covered just prior to the beginning of the project, and during each project more advanced related topics and the topics for the next project are covered. For example, the first project on search is prefaced by uninformed search, during the project game playing and constraint satisfaction are covered, and the final week of the project covers propositional logic (to setup for project two). Each topic and project is designed similarly.

Week	1	2	3	4	5	6	7	8
Topic	History, Ratio-nality, Agents	Uninformed Search	Informed Search	Game Playing	Constraint Satisfac-tion	Propo-sitional Logic	Propo-sitional Inference	First-Order Logic
Project	P0: Build Your Own Agent		P1: Search				P2: Satisfiability	
Week	9	10	11	12	13	14	15	16
Topic	Knowledge Represen-tation	Planning	Probability Theory	Bayesian Networks	Advanced Bayesian Networks	Markov Decision Processes	Machine Learning	Applications
Project	P2 cont'd	P3: Planning			P4: Probabilistic Inference			

Table 1: Course Schedule.

WUMPUS WORLD

Wumpus World is used as a running example in the class textbook [7], and is adapted from the original computer game [10] to be simpler. The simplified version is widely adopted and the basis for the simulator [1] used in the projects.

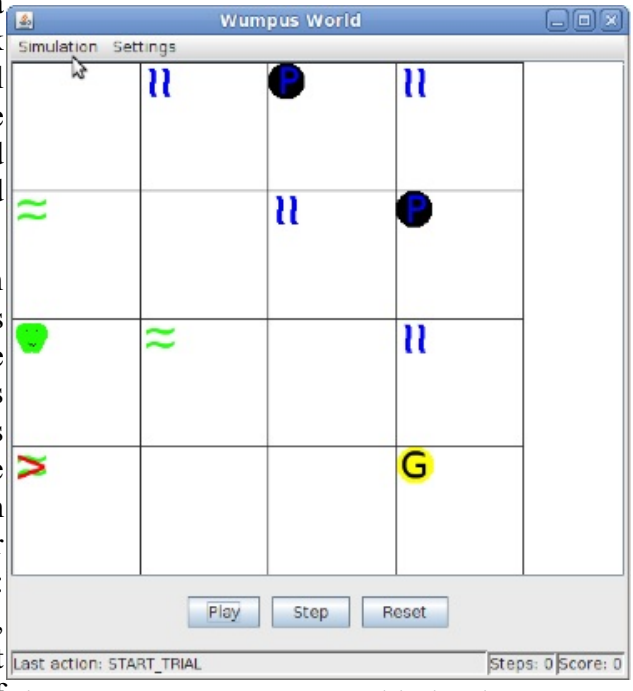


Figure 1: Wumpus World Simulator GUI.

A simulator visualization developed by a student in the course is shown in Figure 1. In WW, a treasure hunter (the player, a red arrow) is seeking gold "G", but must avoid pits "P" and wumpii (the smiling face). The player has a bow and one arrow that can be used to slay a wumpus. The player can perform the following actions: turn-left, turn-right, move-forward, shoot, and grab. The play cannot directly observe the pits or wumpii (if they enter a square with a pit or wumpus they will die), but can sense breeze (vertical waves) or stench (horizontal waves)

if a respective pit or wumpus is in an adjacent location. Likewise, the player can only observe glitter if they are in the same location as the gold.

A solution to WW is a sequence of actions that maximizes player reward; positive reward is given for attaining gold, and negative reward is attributed for performing each action or dying. Thus, optimal solutions avoid death and attain the gold in as few actions as possible.

PROJECTS

The five course projects start with Project 0, a brief assignment to introduce students to WW, and the remaining four projects 1-4 build upon course material.

Project 0: Build Your Own Agent

Within the first week of class, Project 0 is assigned so that students can develop familiarity with the WW simulator and uncover some of the challenges of building AI systems. Project 0 requires students to build an agent (a program that selects actions based on observations attained from the simulator) by any means they wish. The agent built by each student must be able to solve ten configurations of WW (different placements of pits, wumpii, and gold) with no prior knowledge of which configuration they are solving at run-time.

Students generally enjoy this project, but also become frustrated because of the challenges that they must overcome without the benefit of AI techniques developed in later projects. For example, at least one assigned configuration of WW provides observations that are not sufficient to determine the location of a pit, and the agent must guess and sometimes fail. We revisit this issue while discussing probabilistic reasoning to help calculate the probability that a location holds a pit and make better decisions.

In addition to the frustrations noted above, students also re-discover many AI techniques when programming their agent. For example, many students develop end-game databases (as made popular in the game of chess), or implement a form of search such as Djikstra's algorithm.

Project 1: Search

Project 1 requires students to develop search algorithms for deriving solutions for WW. In this project, the WW simulator is modified so that the layout of pits, wumpii, and gold is known *a priori*, which allows the agent to construct a sequence of actions (rather then require sensing and changing behavior while acting). However, to challenge the algorithms developed by the students, the number of pits and wumpii are increased quadratically and by double, respectively, in the dimension of the grid. For example, a grid of size sixteen will include 256 pits and 32 wumpii. The students must solve ten instances of grids size 4, 8, 16, and 32. Moreover, five instances of each dimension grid are solvable, and the other five unsolvable (e.g., there is row of pits barring the player from reaching the gold).

The students implement several uninformed search algorithms, including breadth-first search, depth-first search, uniform-cost search, and iterative-deepening depth-first search. The students also implement two informed search algorithms: greedy best-first search and A* search. For both of the informed search algorithms, the students implement three heuristic functions to guide the search: straight-line distance (from the player to the gold), Manhattan distance (from the player to the gold), and a custom heuristic of their choice. In all algorithms, the students must also implement cycle checking and a closed list.

The student run experiments on all of the WW instances and attempt all configurations of the algorithms. The students submit a report detailing their answers to several questions and use the experiments to explain their answers. Questions include: "What are the trade-offs between uninformed search techniques?", "What are the trade-offs between informed search techniques?", "What are the trade-offs between uninformed and informed search?", "How does the difference in the heuristics effect the search?", and "Which has more impact: duplicate detection, cycle checking, or both?".

Project 2: Satisfiability

Project 2 relaxes the assumption made in Project 1 that the agent has prior knowledge of the locations of pits, wumpii, and gold. The task in Project 2 is to answer queries regarding the locations of pits, wumpii, and gold, given a set of observations (locations where breeze, stench, and glitter are sensed). The students must combine the observations with their knowledge about how observations are generated in WW (e.g., that breeze can be sensed in squares adjacent to pits) to answer a query with a yes, no, or indeterminate response.

The students exercise their understanding of propositional satisfiability by encoding observations and WW rules as propositional logic and implement three inference algorithms: resolution, DPLL, and WalkSat. The rule that pits generate breeze in adjacent squares is stated as:

$$p_{ij} \rightarrow b_{i+1j} \wedge b_{i-1j} \wedge b_{ij+1} \wedge b_{ij-1}$$

which the students must translate to conjunctive normal form (CNF). Similar clauses are required for the gold and wumpii. The rules about WW make up a knowledge base KB and the observations O include facts about breeze, stench, and glitter. The students must answer a query of the form:

$$KB \wedge O \models \alpha$$

where, for example $\alpha = \neg w_{23}$ asks if it is entailed from the knowledge base and observations that location (2,3) does not contain a wumpus. To answer the query the students covert the query to:

$$KB \wedge O \wedge \neg\alpha$$

and attempt to prove the CNF sentence unsatisfiable (i.e., there is no consistent truth assignment of the propositional variables).

The students run experiments on several different size grids and instances of each, and write a report that compares the relative strengths of the algorithms.

Project 3: Knowledge Representation & Planning

Project 3 expands upon Projects 1 and 2 by requiring students to formulate WW as a planning domain theory in a first-order logic-based planning representation, called PDDL [4]. Upon formulating several problems in PDDL, the students experiment with three automated planning software packages. The planners are selected to represent alternative approaches to planning, such as heuristic search (as related to Project 1) or satisfiability (as in Project 2).

For example, as part of the project the students must formulate an action description in PDDL, such as:

```
(:action turn-left
 :parameters (?d1 ?d2 - direction)
 :precondition (and (facing ?d1) (left-of ?d2 ?d1))
 :effect (and (facing ?d2) (not (facing ?d1)))))
```

which represents their knowledge of how turning left will change the state of the player.

In addition to the actions, the students must represent the initial knowledge of the WW player (e.g., "(facing NORTH)", "(at X1 Y0)", and "(haveArrow)") and the goal of the player (e.g., "(haveGold)").

After formulating several solvable and unsolvable problems, the students invoke each planner on each instance and compare the results. As part of a report on their findings, the students address several questions, such as: "Why does a planner based on satisfiability take less time to show that a problem is unsolvable than a heuristic search planner?" and "Why does a heuristic search planner scale better in the dimension of the grid compared to a satisfiability-based planner?"

When asked, students indicate that they enjoyed this project because it is compelling to be able to represent their knowledge in a logical format as opposed to Java source code (as in Project 1). It was also instructive to see that the automated planners could solve any problem expressed in their input language (and faster than their Project 1 code).

Project 4: Bayesian Networks

Project 4 revisits Project 2 by adding probabilities to the rules describing WW. For example, if given the knowledge that two pits are placed with uniform probability in any one of the WW locations, it is possible to conclude that a breeze observation is more likely to be generated by a single pit than by two neighboring pits. Instead of asking students to answer queries about whether a location contains a pit or not, they must compute the probability that the location contains a pit. For example, a query of the form:

$$P(p_{12}|b_{22} \wedge b_{41} \wedge b_{11})$$

asks the probability that there is a pit in location (1,2) given that breeze is observed in locations (2,2), (4,1), and (1,1). To answer the queries, the students learn how to re-write a query in terms of a full joint probability distribution, apply the chain-rule, and remove

conditionally independent variables from each conditional probability (as encoded by the structure of the Bayesian network they are given for the project).

The students implement two algorithms to evaluate the probability of a query: enumeration and variable elimination. As part of a report on their project, the students must answer several queries with their code and also compute two queries manually.

RELATED WORK

A related course on introductory AI at UC Berkeley, teaches AI concepts using the game of Pac-Man [3]. The projects included in the course cover the topics of search, multi-agent search, probabilistic inference, and reinforcement learning. There are many similarities between Pac-Man and WW in that both concern grid-based environments with positive objectives and negative pit-falls. One reason that we believe that WW can be better for some students is that it follows the examples provided in the course text.

The Educational Advances in AI Symposium (EAAI) [8] includes a track on model assignments for teaching AI, but in its first year only included one integrative test-bed for teaching AI (see the Pac-Man framework above). The model assignments in EAAI tend to focus on one specific topic in AI, rather than an integrative curriculum, as we have presented. In our experience, the continuity that one framework provides is beneficial for illustrating the course topics and can reduce the initiation cost for students.

CONCLUSION

We have presented an overview of an introductory course on AI that uses a series of projects based in Wumpus World. The advantages of the course include: a fun and simple software framework for class projects, an emphasis on course material rather than learning a software framework for each project, and a hands-on approach to AI that requires student reflection on why different AI approaches work.

REFERENCES

[1] Biagioni, J. P. Wumpus-lite: A lightweight java-based Wumpus world simulator, http://www.cs.uic.edu/~jbiagion/wumpuslite.html, 2009.

[2] Davis, M., Logemann, G., and Loveland, D. A Machine Program for Theorem Proving. *Communications of the ACM* 5 (7): 394-397, 1962.

[3] De Nero, J., Klein, D. Teaching introductory artificial intelligence with Pac-man. In *Proceedings of the Symposium on Educational Advances in Artificial Intelligence*, 2010.

[4] McDermott, D. PDDL: The planning domain definition language. Technical report, http://www.cs.yale.edu/homes/dvm, 1998.

[5] Pearl, J., *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.

[6] Robinson, J. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM (JACM)* 12 (1): 23-41, 1965.

[7] Russell, S.J., Norvig, P. *Artificial Intelligence - A Modern Approach*. Pearson, 2010.

[8] Sahami, M., desJardins, M., Dodds, Z., Neller, T. Educational Advances in Artificial Intelligence. In *Proceedings of the 42nd ACM Technical Symposium on Computer science education*, SIGCSE '11, pages 81-82, New York, NY, USA, 2011.

[9] Selman, B., Kautz, H., Cohen, B. Local Search Strategies for Satisfiability Testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1996.

[10] Yob, G. Hunt the Wumpus. *The Best of Creative Computing,* 1, 1976.

CODE INSPECTIONS: A WEB CRAWLER EXERCISE FOR STUDENTS*

Steena Monteiro, Renee Bryce
Department of Computer Science
Utah State University
Logan, UT
Renee.Bryce@usu.edu

ABSTRACT

It is well known that peer reviews are often effective in finding software bugs. This paper provides a code inspection exercise in which students work in teams to identify problems in code for a web crawler. The code is intentionally poorly written and the code used to work until the website that it visits changed its format! Students must identify poor design and documentation issues in the code. We gave this assignment to student teams in two separate semesters and found that all students identified design and documentation issues. Students were then given an assignment to either modify the existing code or to rewrite a new web crawler from scratch. Even though the modifications would have been relatively small, every team chose to rewrite the code from scratch and cited poor design and documentation issues for this decision. Our in-class discussion of the exercise revealed that the exercise illuminated the importance of good documentation.

1. INTRODUCTION

Code inspections are a common practice in professional software development teams, but can also be useful to students. Early work by Meyers found that code walkthroughs with pairs of subjects that independently reviewed code and then pooled their findings was the most effective form of testing in their experiments [6]. In the classroom, Trytten found that code reviews not only helped students with better understanding of programming and code comprehension, but also assisted in team

building [7]. Hundhausen et al. saw an improvement in the quality of design and implementation when they integrated a formal code review process into a lower level Computer Science course [3]. Turner et al. helped to accelerate the learning of new programming concepts such as object-oriented programming by using peer reviews [8]. Clark et a. using peer testing which resulted in positive collaborations among students and increased student interest in software testing [1]. Of course, a large body of work on pair programming exists in which students work together on code and consequently review each other's code. Pair programming students often product better work in less time, have more confidence, and enjoy the experience more than working alone [4,5,9]. Of course, personality is an issue in the ability to review code [2]. Our work differs from previous work in that students do not review their own code, but rather review intentionally poor code that the instructor created for the exercise and the code that we use is a web crawler which is a topic that is outside of the scope of our curriculum, yet attainable for our students to understand.

2. CODE INSPECTION EXERCISE

The exercise is conducted as follows: *First*, students individually review the web crawler code and strive to identify at least 10 problems with the code. *Second,* the students meet in class to hold a peer review of the code. During this session, each team has a team leader and a recorder. The team leader makes sure that they review each page and that every team member has the opportunity to speak. The recorder keeps a log of each problem in the code. *Finally,* the students meet outside of class to prepare and type their report. The reports include: a list of the team members and their roles, a list of the total number of defects found and a description of each fault for each team member, a summary of the team recommendations to improve the code, and a summary of the effectiveness of each reviewer. The summary of effectiveness includes: the amount of time that each student spent reviewing the code and the number of defects that each student indentified.

3. CODE TO REVIEW

3.1 Problem Statement

The following code is a web crawler that visits CNN (http://www.cnn.com) to pull the headlines for my favorite four categories – *Technology, Health, Science & Space, and Education.* The code was originally written by *Jack Hacker* who is famous for writing code quickly (although some complain that his code is not very easy to read). Well, this code worked fine until recently – CNN changed the format of their website and the web crawler stopped working! Even worse, *Jack Hacker* is nowhere to be found! Rose, a local software engineer, was assigned to fix the code. She is known for her good design and coding skills. While she is usually very rational, she has decided that it is not worth fixing *Jack Hacker's* code below. She'd rather rewrite it from scratch! Your first task is to identify any 10 problems with this code that make it *bad* code. You will then either modify the code or write your own code to work with the new format on the CNN website.

3.2 Code

The code assigned to the students is as follows:

```
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
// File: cnnCrawler.java
// This code looks at the CNN website and follows some links to get info on articles
that I want more
// info on. All output is written in the working directory to: cnnCrawlerOutput.txt
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
import gnu.regexp.*;
import java.net.*;
import java.io.*;
public class cnnCrawler{
   public static void main(String[] args)
   {
       StringBuffer basePage = new StringBuffer();
         // Connect to CNN and get the document
       basePage = getBasePageContents("http://www.cnn.com");

         // Look at the area of interest (The "MORE FROM CNN" section)
       basePage = initialIsolateBasePageContents(basePage);

         // Pull all of the URLs out
       basePage = getInfo(basePage, " <a href=\"[^\"]*|/b> <a href=\"[^>]*|/b><a
              href=\"[^>]*");
       basePage = getInfo(basePage, "\"/[^(\")]*");
       basePage = getInfo(basePage,"\"[^&]*");

         // Go to the URLs and pull out the information of interest and
         // write to file.
       goToURLs(basePage);
  }
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
// Method: getBasePageContents
// This method opens a connection to the webpage we are interested in and stores all of
the text on the page
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
public static StringBuffer getBasePageContents(String myURL){
       try{
              // Set base document to CNN, open connection,
              // and copy the source text into a buffer
             URL cnnBaseDoc = new URL(myURL);
             cnnBaseDoc.openConnection();
             BufferedReader cnnBaseBuffer = new BufferedReader(new
              InputStreamReader(cnnBaseDoc.openStream()));
              String cnnBaseInputLine;
```

```
            StringBuffer tempDocument = new StringBuffer();
            while ((cnnBaseInputLine = cnnBaseBuffer.readLine()) != null){
                tempDocument.append(cnnBaseInputLine);
        }
            cnnBaseBuffer.close();
            return(tempDocument);
        }
        catch(MalformedURLException e) {
            System.out.println("Unable to create URL object");
            return(null);
        }
        catch(IOException e){
            System.out.println("Unable to open URL");
            return(null);
        }
}
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
// Method: initialIsolateBasePageContents
// This method isolates us to store only the section we are interest in -- the "MORE
FROM CNN" section
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
public static StringBuffer initialIsolateBasePageContents(StringBuffer basePage){
        try{
            RE document = new RE(basePage);
             // Define the left and right isolators
            String sLeft = new String("MORE FROM CNN[//w//W]*");
            RE leftCntxt = new RE(sLeft);
            RE rightCntxt= new RE("><b>SPORTS");
            StringBuffer sLIsolator = new StringBuffer("");
            int iLIsolatorIndex = 0;
            RE regLIsolator = new RE(leftCntxt);
            REMatch ctxtLMatch = regLIsolator.getMatch(basePage);
            sLIsolator.append(ctxtLMatch.toString());
            iLIsolatorIndex = ctxtLMatch.getStartIndex();

            // Find the Right Isolator
            StringBuffer sRIsolator = new StringBuffer();
            RE regRIsolator = new RE(rightCntxt);
            int iRIsolatorIndex = 0;
            REMatch ctxtRMatch = regRIsolator.getMatch(basePage);
            sRIsolator.append(ctxtRMatch.toString());
            iRIsolatorIndex = ctxtRMatch.getStartIndex();
            basePage.delete(iRIsolatorIndex, basePage.length());
            basePage.delete(0, iLIsolatorIndex);
            return(basePage);
        }
```

```
        catch(REException e){
                System.out.println("RE Exception");
                return(null);
        }
}
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
// Method: getInfo
// This method applies the specified regular expression to the string passed in
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
public static StringBuffer getInfo(StringBuffer textToSearch, String regExp){
        try{
                StringBuffer sIsolated = new StringBuffer("");
                int iLIsolatorIndex = 0;
                String sLeft = new String(regExp);
                RE leftCntxt = new RE(sLeft);
                RE regLIsolator = new RE(leftCntxt);
                REMatchEnumeration ctxtLMatch =
                        regLIsolator.getMatchEnumeration(textToSearch);
                while (ctxtLMatch.hasMoreMatches()){
                        sIsolated.append(ctxtLMatch.nextMatch().toString());
                        sIsolated.append("\n");
                }
                return(sIsolated);
        }
        catch(REException e){
                System.out.println("RE Exception");
                return(null);
        }
  }
public static void goToURLs(StringBuffer textToSearch)
        {
        try{
                StringBuffer interestingDoc = new StringBuffer("");
                StringBuffer sInfoForFile = new StringBuffer("");
                int numPage=0;
                FileOutputStream fCnnOut;
                PrintStream pCnnOut;
                String sLeft = new String("/[^\"]*");
                RE leftCntxt = new RE(sLeft);
                String sIsolated = new String();
                int iLIsolatorIndex = 0;
                RE regLIsolator = new RE(leftCntxt);

                REMatchEnumeration ctxtLMatch =
                        regLIsolator.getMatchEnumeration(textToSearch);
```

```
            fCnnOut = new FileOutputStream("cnnCrawlerOutput.txt");
            pCnnOut = new PrintStream(fCnnOut);

            while (ctxtLMatch.hasMoreMatches())
            {
                  numPage++;
                  sIsolated = "http://www.cnn.com";

                  sIsolated += (ctxtLMatch.nextMatch().toString());
                  interestingDoc = connectToURLs(sIsolated);
                  sInfoForFile = getDocInfo(interestingDoc, sIsolated, numPage);
                  pCnnOut.println (sInfoForFile);
            }
            pCnnOut.close();
            System.out.println("You may view the output in file:
                  cnnCrawlerOutput.txt.");
      }
      catch(REException e){
            System.out.println("RE Exception");

      }
      catch (Exception e)
      {
            System.out.println ("Error writing file.");
      }
  }
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
// Method: connectToURLs
// This method opens a URL and returns the text of the page
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
*~*~*~*~*~*~*~*~*~
public static StringBuffer connectToURLs(String urlText){
      try{

            URL cnnBaseDoc = new URL(urlText);
            cnnBaseDoc.openConnection();
            BufferedReader cnnBaseBuffer = new BufferedReader(new
                  InputStreamReader(cnnBaseDoc.openStream()));
            String cnnBaseInputLine;
            StringBuffer tempDocument = new StringBuffer();

            while ((cnnBaseInputLine = cnnBaseBuffer.readLine()) != null){
                  tempDocument.append(cnnBaseInputLine);
      }
            cnnBaseBuffer.close();
            return(tempDocument);
```

```
        }
        catch(MalformedURLException e) {
        System.out.println("Unable to create URL object");
                return(null);
        }
        catch(IOException e){
                System.out.println("Unable to open URL");
                return(null);
        }
  }
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
// Method: getDocInfo
// This method returns the interesting information that we were asked to parse out
// including: Date, Place, Headline, URL, and First paragraph.
//**~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~*~
public static StringBuffer getDocInfo(StringBuffer doc, String URL, int ID){
                StringBuffer importantInfoToReturn = new StringBuffer("");
                StringBuffer Headline = new StringBuffer("");
                StringBuffer Date = new StringBuffer("");
                StringBuffer Place = new StringBuffer("");
                StringBuffer FirstParagraph = new StringBuffer("");
                URL = URL.substring(0, (URL.length()-1));

                Date.append(getInfo(doc, "name=\"DATE\" content=\"[^>]*"));
                if(Date.length() > 0){
                        Date.delete(0,21);
                        Date.delete((Date.length()-1), Date.length());
                }
                else{
                        Date.append("No date Reported.");
                }

                Place.append(getInfo(doc, "<p><b>[^(<p>)]*|<p><b>[^-]*"));
                if(Place.length() > 0){
                        Place.delete(0,6);
                }
                else{
                        Place.append("No location Reported.");
                }

                Headline.append(getInfo(doc, "<title>CNN.com - [^-]*"));
                if(Headline.length() > 0){
                        Headline.delete(0,17);
                        Headline.delete((Headline.length()-1), Headline.length());
                }
```

```
            else{
                Headline.append("No headline Reported.");
            }
            FirstParagraph.append(getInfo(doc, "DESCRIPTION\" content=[^>]*"));
            if(FirstParagraph.length() > 0){
                FirstParagraph.delete(0, 22);
                FirstParagraph.delete(FirstParagraph.length()-1,
                    FirstParagraph.length());
            }
            importantInfoToReturn.append("\n");
            importantInfoToReturn.append((ID + " | "));
            importantInfoToReturn.append((Headline + " | "));
            importantInfoToReturn.append((URL + " | "));
            importantInfoToReturn.append((Date + " | "));
            importantInfoToReturn.append((Place + " | "));
            importantInfoToReturn.append((FirstParagraph));
            return(importantInfoToReturn);
    }
}
```

4. RESULTS

4.1 Review Time and Defects Found

Students reported the amount of time that they spent reviewing the code and a list of the defects that they identified.

Tables 1-(a-e) provide the results for each team. Table 1-(f) summarizes these results. Most students spent one hour reviewing the code as we recommended. One student reported that they spent less time. This student also reported that they found only 6 bugs whereas the average student spent 78.75 minutes and found 8.58 bugs. A total of 8 students spent more than one hour reviewing the code and found 9 to 10 bugs each. The average defects found per hour ranged from 2.5 to 10, but the average of 7.25 and median of 7.75 were close. We review the types of defects that they found next.

Reviewer	Review Time	Total Defects	Defects per hour
1	45	6	8.00
2	120	10	5.00
3	120	10	5.00
4	240	10	2.50
5	80	10	7.50
Average	121	9.2	5.60

(a)

Reviewer	Review Time	Total Defects	Defects per hour
1	60	9	9.00
2	60	8	8.00
3	60	9	9.00
4	60	7	7.00
5	60	10	10.00
Average	60	8.6	8.60

(b)

Reviewer	Review Time	Total Defects	Defects per hour
1	60	10	10.00
2	60	7	7.00
3	60	7	7.00
4	60	8	8.00
5	60	8	8.00
Average	60	8	8.00

(c)

Reviewer	Review Time	Total Defects	Defects per hour
1	100	9	5.40
2	60	5	5.00
3	120	10	5.00
4	90	10	6.67
5	75	10	8.00
Average	89	8.8	6.01

(d)

Reviewer	Review Time	Total Defects	Defects per hour
1	60	8	8.00
2	60	10	10.00
3	60	7	7.00
4	60	8	8.00
Average	60	8.25	8.25

(e)

	Review Time	Total Defects	Defects per hour
Average	78.75	8.58	7.25
Median	60	9	7.75
Min	45	5	2.50
Max	240	10	10.00

(f)

Figure 1 - Summary of Review Time and Defects Found

4.2 Student reported defects

Each student submitted their individual list of defects. Due to space, we compiled a list of all of the unique types defects that were reported by the 24 students:

1. Many comments are missing or are not informative enough
2. The indentation style is not consistent and makes it more difficult to read the code.
3. Some variables look like class names.
4. Some comments are misleading, i.e., one example is that connectToURLs only connects to one URL.
5. Many methods are too large and should be broken into smaller methods.
6. Some variables are never used.
7. Perhaps regular expressions should be assigned to strings to improve the readability.
8. Variables are reused and the meaning changes.
9. Many exceptions are not caught.
10. The code is inefficient in that it grabs all page content and then discards what is not needed.

11. Hard coding should be avoided, i.e., regular expressions and file names are hard coded.
12. A variable assignment in a loop should be moved to come before the loop

None of the students found all of the defects that we anticipated that they would found. However, everyone was effective in identifying the major issues associated with readability of code and documentation. More than half of the students noticed variables that were never used, that exceptions were not always caught, the regular expressions could be rewritten to improve readability, one of the variable assignments should have been outside of the loop that it was in, and that hard coding should be avoided. Fewer than 5 of the students noticed that the code was inefficient in regard to collecting page content, that some variable names look like class names, and that many methods should be broken into smaller methods.

4.3 Class discussion

We reviewed the list of defects in a class discussion. The students agreed that these were all important issues, but the most interesting comments were that the exercise made them more aware of the importance of documenting code. Many students agreed that their homework assignments in their prerequisite courses were relatively small and that they had previously felt that documenting the code was too much work and unnecessary. However, the web crawler exercise made them reflect on the importance of good design and documentation. They felt that Jack the Hacker may have understood the code and may have been able to edit the code quickly, but that outsiders needed to make too much effort to understand and modify the code. They also suggested that Jack the Hacker may forget how to edit the code if he finishes it and then revisits it after a prolonged period of time! When asked about using object-oriented programming more efficiently for this code, many students also said that they did not include this in their write-up but that they addressed it in their implementation. In summary, this exercise illuminated the importance of good design and documentation.

4.4 New code

Students were asked to modify the code that they reviewed or to implement new code from scratch in any language of their choice in order to address the new format of the website that the web crawler visits. All teams created new code from scratch. Four teams wrote the new code in JAVA and one in Perl. Again, all of the teams emphasized that the code that they reviewed was poorly designed and documented and that they preferred to avoid modifying the code.

5. CONCLUSIONS

Undergraduate students in the introductory courses often underestimate the importance of good design and documentation for code. This is not surprising because most students only have experience with small programs. We gave the students an exercise in which they reviewed code that had design flaws and poor comments. We

collected data from 24 students that reviewed the code and found that the average student spent approximately 79 minutes reviewing the code and identified 8.58 defects. There were 12 types of defects that the students reported. All students identified the major documentation issues, but only some of the design issues. Finally, a class discussion revealed that many students had previously underestimated the importance of good design and documentation because their limited experience from the prerequisite courses focused only on small problems where they worked individually. This exercise made them aware of the importance of creating good design and documentation that others can easily understand.

REFERENCES

[1] Clark., N., Peer testing in Software Engineering Projects. Conference on Australian Computing Education, 30,(30),41-48, 2004.

[2] Devito, A., Cunha, D., Does personality matter?: an analysis of code-review ability, *Communications of the ACM*, 50,(5), 109-112, 2007.

[3] Hundhausen, C., Agrawal, A., Fairbrother, D., Trevisan, M., Integrating pedagogical code reviews into a CS 1 course: an empirical study, In Proceedings of the 40th ACM technical symposium on Computer science education, pages 291-295, 2009.

[4] McDowell, C., Werner, L., Bullock, H., Fernald, J., The effects of pair-programming on performance in an introductory programming course, *ACM SIGCSE Bulletin*, 34,(1),38-42, 2002.

[5] McDowell, C., Werner, L., Bullock, H., Fernald, J.,The Impact of Pair Programming on Student Performance, Perception and Persistence, *Proceedings of the International Conference on Software Engineering (ICSE'03)*, page 602, 2003.

[6] Meyers., G. A controlled experiment in program testing and code walkthroughs/inspections, *Communications of the ACM*, 21(9):760-768, 1978.

[7] Trytten., D., A design for team peer code review, *ACM SIGCSE Bulletin*, 37,(1),455-459, 2005.

[8] Turner, S., Pérez-Quiñones, M., Edwards, S., Chase, J., Peer review in CS2: conceptual learning, ACM technical symposium on Computer Science Education (SIGCSE), pages 331-335, 2010.

[9] Williams, L., Kessler, R., Cunningham, W., Jeffries, R., Strengthening the case for pair programming, *IEEE Software*, 17,(4),19-25, 2000.

OOP: THE REST OF THE STORY*

Chuck Allison, Nathan Liddle
Department of Computing/Network Sciences
Utah Valley University
chuck.allison@uvu.edu

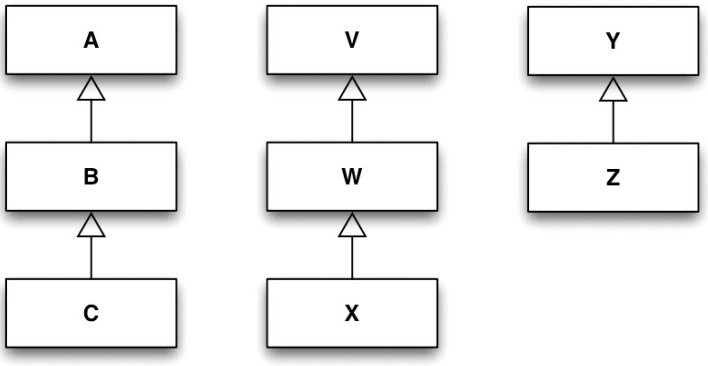
ABSTRACT

The object-oriented paradigm has influenced software development for decades now and has found its place in the mainstream of CS curricula. Its facility for improving program organization is well known and today's programmers are expected to readily employ its features when they apply. Few programmers, and even fewer CS curricula, however, are cognizant of some of OOP's more powerful aspects. Polymorphism, for example, does not only apply to one class hierarchy, hence *multimethods* are part of the OOP repertoire. Software designers should also understand how method pre-and-post conditions interact with inheritance, since class interfaces constitute a *contract* with client programs. Finally, it is important to understand that the features of object-oriented programming do not explicitly require statically defined classes. In this paper we explore multimethods, contract programming, and implementing objects without an explicit class type, with examples in popular, modern programming languages.

POLYMORPHISM IN MORE THAN ONE DIMENSION

Subtype polymorphism, as taught in typical CS curricula, consists of a *dynamic search* in a *single* class hierarchy for the most specific method consistent with a specific function call. While this is certainly the most common case of subtype polymorphism in object-oriented programs, it is not the only case to consider. The popularity of the Visitor Pattern [1], which implements two-dimensional polymorphism (aka *double dispatch*), illustrates the utility of methods where more than one parameter participates in runtime method selection. Visitor requires explicit cooperation between the two hierarchies, however, and is thus limited to the case of two dimensions.

Common Lisp supports runtime method dispatch with an arbitrary number of dimensions.[2] In Lisp you define a "generic function", a placeholder interface for stand-alone functions that take parameters from multiple hierarchies. Consider the following sample hierarchies.



Suppose we define a generic function where the first parameter comes from the A-B-C hierarchy, the second from V-W-X, and the third from Y-Z. Given a valid function call, the runtime mechanism must select the *most specific* method where each argument has an is-a relationship with its respective parameter. To determine an ordering for specificity, you can consider the three parameters as the indices of three nested loops traversing the possible type combinations, with the first hierarchy as the set of target objects for the outer loop, the second hierarchy for the next inner loop, and so on. For our sample hierarchy the following sequence of type triple applies:

(A,V,Y), (A,V,Z), (A,W,Y), (A,W,Z), (A,X,Y), (A,X,Z),(B,V,Y), (B,V,Z), (B,W,Y), (B,W,Z), (B,X,Y), (B,X,Z), (C,V,Y), (C,V,Z), (C,W,Y), (C,W,Z), (C,X,Y), (C,X,Z)

Now suppose that only the following method combinations actually have implementations:

(A,V,Y), (A,V,Z) , (A,X,Z), (B,W,Y) , (B,X,Z) , (C,V,Z), (C,X,Y)

The correct method is selected by traversing the second list in *reverse order* until the first combination that has an is-a relationship between each argument and its corresponding parameter is found. For example, the call **f(c,w,z)** will call the (C,V,Z) implementation since a W object "is-a" V. The following complete Common Lisp program shows the runtime bindings for all parameter combinations.

```
;; Define 3 class hierarchies
(defclass A () ())
(defclass B (A) ())
(defclass C (B) ())

(defclass V () ())
(defclass W (V) ())
(defclass X (W) ())
```

```
(defclass Y () ())
(defclass Z (Y) ())

;; Define Multimethods
(defgeneric f (p1 p2 p3))
(defmethod f ((p1 A) (p2 V) (p3 Y)) (print '(A V Y)))
(defmethod f ((p1 B) (p2 W) (p3 Y)) (print '(B W Y)))
(defmethod f ((p1 C) (p2 X) (p3 Y)) (print '(C X Y)))
(defmethod f ((p1 A) (p2 V) (p3 Z)) (print '(A V Z)))
(defmethod f ((p1 A) (p2 X) (p3 Z)) (print '(A X Z)))
(defmethod f ((p1 B) (p2 X) (p3 Z)) (print '(B X Z)))
(defmethod f ((p1 C) (p2 V) (p3 Z)) (print '(C V Z)))

;; Create objects
(setf a (make-instance 'A))
(setf b (make-instance 'B))
(setf c (make-instance 'C))
(setf v (make-instance 'V))
(setf w (make-instance 'W))
(setf x (make-instance 'X))
(setf y (make-instance 'Y))
(setf z (make-instance 'Z))


;; Test Output (in 2 columns):
(f a v y)   ; (A V Y)
(f a v z)   ; (A V Z)
(f a w y)   ; (A V Y)
(f a w z)   ; (A V Z)
(f a x y)   ; (A V Y)
(f a x z)   ; (A X Z)
(f b v y)   ; (A V Y)
(f b v z)   ; (A V Z)
(f b w y)   ; (B W Y)
(f b w z)   ; (B W Y)
(f b x y)   ; (B W Y)
(f b x z)   ; (B X Z)
(f c v y)   ; (A V Y)
(f c v z)   ; (C V Z)
(f c w y)   ; (B W Y)
(f c w z)   ; (C V Z)
(f c x y)   ; (C X Y)
(f c x z)   ; (C X Y)
```

Not all languages support multimethods, of course. For this particular example, one can hard-code the method selection process by manually searching the implementations in most specific to most general order, as illustrated in the C++ except below:

```
string dispatch(const A& a, const V& v, const Y& y) {
    const B* pb = dynamic_cast<const B*>(&a);
    const C* pc = dynamic_cast<const C*>(&a);
    const W* pw = dynamic_cast<const W*>(&v);
    const X* px = dynamic_cast<const X*>(&v);
    const Z* pz = dynamic_cast<const Z*>(&y);
```

```
    if (pc) {
        if (px)
            return "(C X Y)";
        if (pz)
            return "(C V Z)";
    }
    if (pb) {
        if (px && pz)
            return "(B X Z)";
        if (pw)
            return "(B W Y)";
    }
    // The first parameter must be an explicit A
    if (px && pz)
        return "(A X Z)";
    if (pz)
        return "(A V Z)";
    return "(A V Y)";
}
```

Although only applicable to this particular example, this code should help students understand what languages like Lisp have to do behind the scenes to make mutimethods work. C#, Java, Python and Perl have workarounds for programmers who need multi-dimensional polymorphism in those languages.[3]

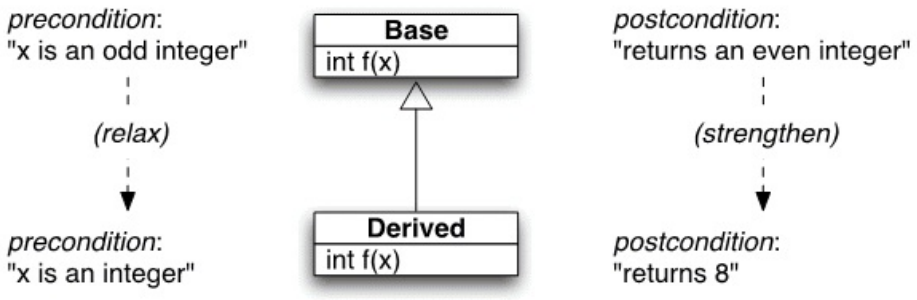
ENFORCING POLYMORPHIC METHOD CONTRACTS

All functions have, in essence, a "contract" with clients. Given certain *preconditions* (e.g., input or expected environmental settings), the function will guarantee specific *postconditions* (e.g., output or side effects).

If the function is an object method, the rules of inheritance interact with the contract. The Liskov Substitution Principle (LSP) specifies that instances of subclasses should be able to substitute for superclass instances without changing the contract. Liskov and Wing noted that this can be achieved through *contravariance* of arguments and *covariance* of return types in the subclasses. [4]

Consider a scheduler class with a method that consults an appointment database to find an available appointment on or after a particular date. The precondition for the base class method could simply be that the date given is a valid future date, while the postcondition might require that the date returned would be a previously open appointment on or after the date specified. In order for a derived class method to follow the LSP, it would have to accept at least all of the values accepted by the base class method. The output could be more specific, such as finding the next available *weekend* appointment, but it should still provide a date on or after the one specified.

The following diagram depicts a method, **f**, in a class hierarchy where the inputs in the subclass override for **f** are allowed to be more general (contravariance of arguments), and the output is more specific (covariance of return types).



Newer object-oriented languages incorporate some of these rules in the language design. The following D language code shows these conditions in its class declarations:

```
class Base {
public:
    int f(int x)
    in {
        assert(x % 2 == 1);          // Pre-condition
    }
    out (retval) {
        assert(retval % 2 == 0);     // Post-condition
    }
    body {
        int retval = x*2;
        return retval;
    }
}

class Derived : Base {
public:
    int f(int x)
    in {
        assert(is(typeof(x) : int));// Less strict than x % 2
        == 1
    }
    out (retval) {
        assert(retval == 8);         // More specific than x % 2
         == 0
    }
    body {
        int retval = 8;
        return retval;
    }
}
```

The subclass class above allows a broader range of input than the superclass, and its return type is more specific. The principles of contravariance and covariance for function contracts are summarized in the mnemonic: "Require no more; promise no less." [5]

OBJECTS WITHOUT EXPLICIT CLASS TYPES

The essence of object-oriented programming consists of

1. *Encapsulation* (separating client interfaces from implementation detail)
2. *Subtypes* (where one group of objects can be considered a subset of another, and where the functionality of the former is a superset of the latter)
3. *Dynamic Method Binding* (where the most specific, applicable method is chosen for execution)

Often this conceptual triplet is expressed with the terms *classes*, *inheritance*, and *polymorphism*. In most object-oriented languages, however, classes and inheritance are *static* software artifacts; an object's interface and relationship to other types is determined solely by source code processed by the compiler. A great deal of flexibility is gained, however, when the behavior of an object is left to runtime determination -- *dynamic inheritance*, if you will.

In static object-oriented languages, if class Y inherits from class X, then an instance **y** of type Y can invoke X's methods, and in addition can provide overrides for some or all of the methods inherited from X. Again, the inheritance relationship between X and Y is specified in the source code and enforced by the *compiler*. In a *prototypal* language, such as JavaScript [5], an object can be created at runtime to "inherit" properties of another, existing object; the latter is called the *prototype* of the former. Whenever an object fails to find a requested method, it delegates the request to its prototype object. There are no classes in prototypal languages; instead, data and methods are bound directly to an object. In the following JavaScript example, the object **x**, which implements a simple stack data structure, acts as prototype for the object **y**.

```
x = {
    data : [],
    push : function (a) {
        this.data.push(a);
    },
    pop : function () {
       return this.data.pop();
    }
};

// Make x y's prototype
var F = function () {}
F.prototype = x;
y = new F();

// Add a top method to y
y.top = function () {
    return this.data[this.data.length-1];
}

y.push(1);
println(y.top())   // 1
y.push(2);
println(y.top())   // 2
```

The relationship between **y** and **x** in this example is established at execution time. In addition, we have dynamically added a **top** method to the object **y**. Methods can also be removed by setting them to **null**.

Besides the awkward syntax of establishing **x** as **y**'s prototype, JavaScript does not directly support changing an object's prototype at runtime. Because Python is universally available, increasingly popular, and eminently readable, we have implemented a simple prototype system in Python using a message passing protocol, as seen in the following code excerpt. We use a single class (Object) to achieve the desired behavior, but classes are otherwise not used.

```
class Object(object):
    def __init__(self):
        self.prototype = None

    def send(self, msg, *parms):
        if hasattr(self, msg):
            f = getattr(self, msg)
           return f(self, *parms)                  # Plain function
call
        else:
            if not self.prototype:
                raise Exception, 'no such method: ' + msg
            return self.prototype.send(msg, *parms)   # Delegate
request

    def set_prototype(self, prototype):
        assert(isinstance(prototype,Object))
        self.prototype = prototype

    def clone(self):
        return copy.deepcopy(self)
```

An instance made a prototype via the **set_prototype** method, which may be called at any time, allowing an object to change prototypes during the lifetime of a program. Methods are invoked by **Object**'s **send** method, passing the method name as a string along with any required arguments. The recursive nature of **send** supports an arbitrarily long sequence of prototypes. The following code repeats the actions of the JavaScript example seen earlier.

```
# Bind data and methods to x
x = Object()
x.data = []
x.push = lambda this, a: this.data.append(a)
x.pop = lambda this: this.data.pop()

# Make x y's prototype; add method "top"
y = Object()
y.set_prototype(x)
y.top = lambda this: this.prototype.data[-1]

y.send('push',1)
print y.send('top')     // 1
```

```
y.send('push',2)
print y.send('top')     // 2
```

SUMMARY

The object-oriented paradigm offers more power and sophistication than many practitioners realize. Multimethods, contract programming, and dynamic inheritance give software developers useful tools for implementing quality software to solve computational problems. We have found these topics suitable for upper division students, particularly in an analysis of programing languages course.

REFERENCES

[1] Gamma et al, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1994.

[2] Keene, S., *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS,* Addison-Wesley, 1989.

[3] A nice summary with reference links is found on Wikipedia at http://en.wikipedia.org/wiki/Multiple_dispatch

[4] Liskov, B. and Wing, J., "A Behavioral Notion of Subtyping", *ACM Transactions On Programming Languages and Systems,* Vol. 16, No. 6, Nov. 1994, pp. 1811-1841.

[5] Cline, M. and Lomow, G., *C++ FAQs,* Addison-Wesley, 1994.

[6] Crockford, D., *JavaScript: The Good Parts,* O'Reilly, 2008, Chapter 5.

VALIDATING AN INSTRUCTOR RATING SCALE FOR THE DIFFICULTY OF CS1 TEST ITEMS IN C++*

Evelyn Lulis
College of Computing and Digital Media
DePaul University
Chicago, IL
evelyn@cdm.depaul.edu

Reva Freedman
Department of Computer Science
Northern Illinois University
DeKalb, IL
rfreedman@niu.edu

ABSTRACT

This paper describes the results of a study on the consistency and accuracy of an instructors' rating scale for multiple-choice C++ questions. The class included standard topics for CS1, including data types, control structures, arrays, pointers and a basic introduction to object-oriented programming. The study was carried out because polling a small number of instructors is easier than analyzing data from a large class. Eight instructors rated 64 questions from an exam taken by 214 students. The results showed that the instructors were highly consistent in their ratings and that the mean rating correlated well with the difficulty level of the questions as estimated by student results on corresponding questions. Several subsets of the results were also analyzed, including breakdowns by concept vs. code, by topic, and by number of topics per question. Both reliability and accuracy were demonstrated for the concept and code levels, for topics with a sufficient number of questions, and for questions involving two topics.

INTRODUCTION

While planning an intelligent tutoring system (ITS) for C++, sufficient research results on the difficulty level of C++ test items could not be found. This difficulty does not seem to be specific to computer science; for example, Jonassen and Hung discuss the lack of research regarding problem difficulty in problem based learning, stating that "most

often, teachers or instructional designers use their best judgment to determine an appropriate difficulty level based mainly on their experiences or intuition" [6, p. 8].

To ensure the validity of the items to be used in the ITS, a study was undertaken where eight instructors rated 64 questions on an exam taken by 214 students. To demonstrate the validity of the instructors' ratings would mean that further data for the ITS can be obtained by polling instructors, easier than analyzing data from a large class.

In this paper, the results are shown for both the consistency of the instructors' ratings and their accuracy with regard to the students' results. In addition, the data were broken down three ways, by whether the questions were concept- or code-based, by the topic involved, and by the number of topics required to answer each question. Results for each of these breakdowns are also shown in this paper.

METHODOLOGY

Permission was received to use the 64 multiple-choice items on a final exam administered to all 214 students in a C++ based CS1 class at a large state university in May 2009. Similar to other CS1 courses, topics include basic data types, control structures, arrays, pointers and a basic introduction to object-oriented programming. Due to the history of the course, it probably includes more material on pointers than the average C++ CS1 course.

Eight instructors were also given the 64 test items and asked to rate each item on a scale of 1 to 5, from "very easy" to "extremely difficult." The instructors were given a rubric explaining the meaning of these terms. All of the instructors were experienced CS1 teachers who had taught the class within the last five years and received good to excellent teaching evaluations.

The questions were classified on two axes. First, each question was classified as to whether it was a conceptual question or a question involving C++ code. Second, each question was labeled as to which of 10 categories of C++ knowledge were required to correctly answer it. To determine the reliability of the coding scheme, two raters identified the categories for each test question. There were no differences on the concept/code axis. On the knowledge axis, Cohen's kappa [1, 4] was used to obtain an estimate of the consistency of the categorization scheme. Following that, differences were resolved to produce a uniform coding scheme.

ID	Question Type	Item Count	% of Total
n	Concept	18	28%
x	Code	46	72%
	Total	64	100%

Table 1: Conceptual vs. code questions

Table 1 shows the breakdown between concept and code questions. Concept questions are those which were considered conceptual by the raters whether or not the question was also considered to belong to a content area. For example, the following

question [3, p. p. 464] was considered a conceptual question that also required object-oriented knowledge. (For reasons of security, textbook questions are used instead of actual exam questions.)

> Member functions of a class are normally made _____ and data members of a class are normally made _____.
> a. public, private b. private, public
> c. public, public d. private, private

Table 2 shows the categories and the number of questions in each category. No totals are shown for Table 2 since questions could require more than one type of knowledge. Table 3 shows the breakdown of the questions according to the number of categories required to answer each question correctly. No question required more than three types of knowledge. For example, this question [3, p. 381] was considered to belong to require understanding of both functions and pointers:

> Which of the following would be a function prototype for a function called *exchange* that takes two pointers to double-precision, floating-point numbers x and y as parameters and does not return a value?
> a. void exchange(double *x, double *y)
> b. void exchange(double *, double *)
> c. void exchange(double, double);
> d. both b and c are correct

while this question [based on 3, p. 406-407] would be categorized as object-oriented and input/output:

> Consider the following structure definition:
> struct Time{
> int hour;
> int minute;
> int seconds;};
> Assume that the pointer **timePtr** has points to a **Time** object, and that **timePtr** contains the address of **timeObject**. Which of the following would print member **hour** of **timeObject** with pointer **timePtr**?
> a. cout << timePtr.hour;
> b. cout << timeObject.hour;
> c. cout << timePtr-> hour;
> d. none of the above

No.	ID	Category Name	Item Count
1.	a	Arrays	15
2.	f	Functions	11
3.	i	I/O	6
4.	m	Math	2

5.	o	Objects	11
6.	p	Pointers	12
7.	r	References	6
8.	s	Strings	8
9.	t	C++ “Structs”	8
10.	y	Control Structures	5

Table 2: Knowledge categories

No. of Categories	Examples	Item Count	% of Total
1	See Table 2 for category IDs.	38	64%
2	Combinations of two categories, including ap, as, ay, fp, fr, fs, io, it, pr, and st	17	29%
3	Combinations of 3 categories, including afp, afr, and fpr	4	7%
Total		59	100%

Table 3: Number of categories per item

RESULTS

The first goal was to determine whether the instructors were consistent in their ratings. Cronbach's alpha [2, 5, 8] was used to determine the reliability of the instructors' opinions. The result was .834, which is considered excellent for this experimental design. If one professor rated an item relatively low, then the other professors also tended to rate that item relatively low, relative to how they rated the remaining items.

The second goal was to compare the instructors' average rating of item difficulty to the true level of difficulty of the question, as estimated by the percentage of students who got that item correct. This measure is consistent with Wood's gauge of difficulty level: "problem difficulty is the probability of it being successfully solved" [7, as cited in 6, p. 8]. Since the instructors' difficulty ratings are ranks, Spearman's rho was used to calculate the correlation. Results indicate that the ratings of seven of the eight instructors were significantly correlated with the percentage of students who correctly answered an item. Spearman's rho was also calculated between the mean instructor ranking and the true difficulty level, as estimated by the student results. This correlation was $r = -.463$, $p <$

.001, which indicates an extremely significant correlation, moderate to substantial in strength. The more difficult the professors' rated a given test item on average, the lower the percentage of students who answered the item correctly.

ID	Question Type	Items	Cronbach's Alpha	Spearman's Rho	p
n	Concept	18	.94	-.64	< .01 **
x	Code	46	.79	-.46	< .01 **

T able 4: Results: Conceptual vs. code questions
(** = significant at 1% level)

Analysis of subtypes

Having established the validity of the scale as a whole, the items were categorized according to the major topic involved. The consistency and accuracy analysis was then repeated for each category.

As shown in Table 4, the faculty have significant agreement among themselves as to which questions are concept questions and which are not. Furthermore, their estimate of difficulty level matches the students' for both categories.

No.	ID	Category Name	Item Count	Cr. Alpha	Sp. Rho	p
1.	a	Arrays	15	0.90	-0.77	<0.01 **
2.	f	Functions	11	0.52	-0.63	0.04 *
3.	i	I/O	6	0.78	-0.55	0.26
4,	m	Math	2	0.90	1.00	1.00
5.	o	Objects	11	0.82	-0.07	0.84
6.	p	Pointers	12	0.74	-0.76	<0.01 **
7.	r	References	6	0.69	-0.82	0.04 **
8.	s	Strings	8	0.98	-0.23	0.58
9.	t	C++ “Structs”	8	0.77	-0.42	0.30
10.	y	Control structures	5	0.77	-0.87	0.05 trend

Table 5: Results: Knowledge categories
(** = significant at 1% level, * = 5% level, trend = 10% level)

Table 5 shows the results by category of knowledge required to answer the question. As can be seen from the Cronbach's alpha results, the consistency among faculty members

is generally very good but varies with the category. In particular, the low result for questions involving functions is interesting, as it implies that faculty members disagree among themselves as to the difficulty level of questions involving functions. Many of the function questions also involved pointers or references being used as arguments. Perhaps breaking down the function questions according to other elements involved would enable us to identify the issues causing the low result for this category.

The correlation between faculty and student results is significant for the larger categories. When there is only a small number of items in a category, each item has an outsize contribution to the correlation calculation. Thus we need additional data to tell whether the correlation would be significant for the other categories.

No. of Catgs	Items	% of Total	Cronbach's Alpha	Spearman's Rho	p
1	38	64%	.91	-.24	.15
2	17	29%	.68	-.75	<.01 **
3	4	7%	.52	-.40	.75

Table 6: Results: Number of categories per item
(** = significant at 1% level)

Finally, Table 6 shows the results from classifying questions are classified by the number of categories assigned to each question. Reasonable consistency was obtained for all levels of this table. It is not surprising that consistency drops as the number of categories increases, since one would expect that the easier problems are easier to judge. A significant correlation with student results was obtained for the two-category questions. For the three-category questions, since there were only a small number of them, the lack of significant correlation is consistent with the results obtained in Table 5, where significance was only obtained for the larger categories. The surprising result is for the single-category items. Although it might be expected that faculty would find these questions easier to judge, and the faculty were certainly consistent as to which questions belonged in this category, the results show a greater discrepancy between faculty opinions and student scores than in the other levels of this table. Further research is required to identify the reasons for this discrepancy.

CONCLUSIONS AND FUTURE WORK

In this study, eight instructors rated 64 questions according to difficulty, using a C++ exam taken by 214 students. The overall results showed that the instructors were highly consistent in their ratings and that the mean rating correlated well with the difficulty of the questions as estimated by student results on the same questions.

The data were also subdivided three ways. When breaking down the results by conceptual vs. code-based questions, the results were significant for both levels. Second, when the results were analyzed by topic, both reliability and accuracy were demonstrated for topics with a sufficient number of questions. Finally, when the results were analyzed

by the number of topics a student needed to know to answer the question, the results were significant for some but not all subsets.

The exam used in this study was a non-comprehensive final. As a result, most of the test questions addressed material from the second half of the semester. In future work, we hope to add questions from the midterm exam to see if significance can be obtained for all the C++ topics. Additionally, two anomalies were observed in the analysis, one involving questions about functions and the other about questions involving only one topic. The midterm data will also give us additional data on both of those issues, which we hope will help us resolve those anomalies.

ACKNOWLEDGEMENTS

Jeanette Shutay of Calumet College of St. Joseph has our gratitude for suggesting the statistical approach.

REFERENCES

[1] Cronbach, L., Coefficient alpha and the internal structure of tests, *Psychometrika,* 16, (3), 297-334, 1951.

[2] Deitel, H. M., Deitel, P. J., *C++ How to Program*, Upper Saddle River, NJ: Prentice Hall, 2003.

[3] Gliem, J. A., Gliem, R. R. Computing, interpreting and reporting Cronbach's alpha reliability coefficient for Likert-type scales, Midwest Research to Practice Conference in Adult, Continuing and Community Education, 2003, http://hdl.handle.net/1805/344, retrieved January 31, 2011.

[4] Jonassen, D. H., Hung, W., All problems are not equal: Implications for problem-based Learning, *Interdisciplinary Journal of Problem-based Learning*, 2, (2), 6-28, 2008.

[5] Wood, P. K. A statistical examination of necessary but not sufficient antecedents of problem solving behavior, doctoral dissertation, University of Minnesota, 1985.

[6] Yu, C. H., An introduction to computing and interpreting Cronbach coefficient alpha in SAS, *Proceedings of SAS Users Group International 26 (SUGI)*, 2001, http://www2.sas.com/proceedings/sugi26/p246-26.pdf.

Papers
of the
Twenty-fifth Annual
Consortium for Computing
Sciences in Colleges
Southeastern Conference

November 11-12, 2011
Furman University
Greenville, South Carolina

WELCOME — 2011 CCSC: SOUTHEASTERN CONFERENCE

Welcome to the 25th Southeastern Regional Conference of the Consortium for Computing Sciences in Colleges. The CCSC:SE Regional Board welcomes you to Greenville, SC, the home of Furman University. The conference is designed to promote a productive exchange of information among college personnel concerned with computer science education in the academic environment. It is intended for faculty as well as administrators of academic computing facilities, and it is also intended to be welcoming to student participants in a variety of special activities. We hope that you will find something to challenge and engage you at the conference!

The conference program is highlighted with a variety of sessions, such as a pre-conference workshop, interesting guest speakers, tutorials, panels, student posters, and several sessions for refereed papers. After a double blind reviewing process 15 excellent papers were accepted for conference presentation for an acceptance rate of 60%. Two exciting activities are designed specifically for students, a research contest and an undergraduate programming competition, with prizes for the top finishers in each.

We especially would like to thank the faculty, staff, and students of Furman University for their help in organizing this conference, in particular Ms. Kala Kennemore, administrative assistant in the Dept. of Computer Science. Many thanks also to the CCSC Board, the CCSC:SE Regional Board, and to a wonderful Conference Committee. Thank you all so much for your time and energies.

We also need to send our deepest appreciation to our partners, sponsors, and vendors. Please take the time to go up to them and thank them for their contributions and support for computing sciences education. *CCSC:SE National Partners:* National Science Foundation, Panasonic Solutions Company, and Turing's Craft. *Sponsoring Organizations:* CCSC, ASM-SIGCSE, Upsilon Pi Epsilon.

We could not have done this without so many excellent submissions from authors, so many insightful comments from reviewers, and the support from our editors John Meinke and Susan Dean. Thanks to all of you for helping create such a great program.

We hope you enjoy the conference and your visit to Furman University.

Kevin Treu, Chris Healy, Conference Co-Chairs
Furman University
Hala ElAarag, Program Chair
Stetson University

REGIONAL BOARD — CCSC SOUTHEASTERN REGION

Kevin Treu, 2011 Site Chair, Regional Board Chair. . . . . Furman University, Greenville, SC
Chris Healy, 2011 Site Chair. . . . . Furman University, Greenville, SC
Ben Setzer, Secretary. . . . . Kennesaw State University, Kennesaw GA
John Hunt, Treasurer. . . . . Covenant College, Lookout Mountain, GA
Hala ElAarag, Program Chair. . . . . Stetson University, DeLand, FL
Stephen Carl, Program Co-Chair. . Sewanee: The University of the South, Swanee, TN
Susan Dean, Publicity Chair. . . . . UMUC Europe, Heidelberg, D
Julia Benson-Slaughter, Local Registrar. . Georgia Perimeter College, Dunwoody, GA
John Meinke, Proceedings Editor. . . . . UMUC Europe, Heidelberg, D
James Hale, 2010 Site Chair. . . . . Spelman College, Atlanta, GA
Alfred Watkins, 2010 Site Chair. . . . . Spelman College, Atlanta, GA
Bill Myers, At-Large Member. . . . . Belmont Abbey College, Belmont, NC
Robert Lover, At-Large Member. . . . . Belmont Abbey College, Belmont, NC
Kevin Treu, CCSC:SE Regional Representative. . . Furman University, Greenville, SC

CONFERENCE COMMITTEE — 2011 CCSC SOUTHEASTERN CONFERENCE

Chris Healy, Local Arrangements Chair, Programming Contest Co-Director . . . . . Furman University, Greenville, SC
Kevin Treu, Local Publicity Chair, Speakers Chair. Furman University, Greenville, SC
Kala Kennemore, Vendors Chair, Local Sponsors Chair. . . . . Furman University, Greenville, SC

Andy Digh, Programming Contest Co-Director. . . . . Mercer University, Macon, GA

REVIEWERS — 2011 CCSC SOUTHEASTERN CONFERENCE

Shankar Banik. . . . . The Citadel, Charleston, SC
Dean Brock. . . . . University of North Carolina Asheville, Ashville, NC
Peter Brown. . . . . Converse College, Spartanburg, SC
Rebecca Bruce. . . . . University of North Carolina Asheville, Ashville, NC
Stephen Carl.. . . . . Sewanee: the University of the South, Swanee, TN
Mike Dowell. . . . . Augusta State University, Augusta, GA
Joe Dumas. . . . . University of Tennessee at Chattanooga, Chattanooga, TN
Hala ElAarag. . . . . Stetson University, DeLand, FL
Alessio Gaspar. . . . . University of South Florida, Lakeland, FL

James Harris.. . . . . Georgia Southern University, Statesboro, GA
John Hunt.. . . . . Covenant College, Lookout Mountain, GA
William Jones. . . . . Coastal Carolina University, Conway, SC
Xinlian Liu.. . . . . Hood College, Frederick, MD
Yi Liu.. . . . . Georgia College and State University, Milledgeville, GA
Anne Olsen.. . . . . Winthrop University, Rock Hill, SC
Barbara Plaut. . . . . Maryville College, Maryville, TN
Andy Pounds . . . . . Mercer University, Macon, GA
Ian Sanders.. . . . . University of the Witwatersrand, South Africa
Todd Schultz. . . . . Augusta State University, Augusta, GA
Anil Shende. . . . . Roanoke College, Salem, VA
Yong Shi. . . . . Kennesaw State University, Kennesaw, GA
David Sykes.. . . . . Wofford College, Spartenburg, SC
Kevin Treu.. . . . . Furman University, Greenville, SC
Sally Wahba.. . . . . Clemson University, Clemson, SC
Laurie White. . . . . Mercer University, Macon, GA
Cong-Cong Xing. . . . . Nicholls State University
J.F. Yao. . . . . Georgia College & State University, Milledgeville, GA
Yingbing Yu.. . . . . Austin Peay State University, Clarksville, TN
Marsha Zaidman. . . . . University of Mary Washington, Fredericksburg, VA

SESSION PRESIDERS — 2011 CCSC SOUTHEASTERN CONFERENCE

Yi Liu.. . . . . Georgia College and State University, Milledgeville, GA
Jim Harris.. . . . . Georgia Southern University, Statesboro, GA
Anne Olson. . . . . Winthrop University, Rock Hill, SC
J.F. Yao. . . . . Georgia College and State University, Milledgeville, GA
Marsha Zaidman. . . . . University of Mary Washington, Fredericksburg, VA

KEYNOTE ADDRESS

Friday, November11, 2011

IT-oLogy - IT's working! *

Mr. Lonnie Emard
Executive Director
Consortium for Enterprise Systems Management

A full collaboration of higher education and the business community is not as simple as it sounds. However, when the common mission becomes one that has clarity, measurable results and in some ways is addressing "crisis" levels of concern, then effective collaboration can be possible. As a consortium of partners, ranging from companies in all industries that employ information technology professionals, colleges and universities that teach computing and IT related curriculum, to well meaning organizations that focus on entrepeneurship and economic development, IT-oLogy is proving it can be done. The intent of this session will be to engage in dialog around the framework for advancing IT talent that has produced tangible results and can scale across the region and across the country.

DIGITAL FORENSICS*

PRE-CONFERENCE WORKSHOP

Dr. Crystal Edge
Professor, Computer Science and Information Systems Department
Coastal Carolina University

This workshop demonstrates selected topics from an introductory digital forensics course within the context of a class project. Topics include general cell phone forensics, Windows forensics, data carving, forensic imaging, file signature analysis, registry analysis, volatile memory analysis, and other selected topics. The presenter taught this course during the Spring 2011 semester, and the final class project was to write, produce, and perform a short play that demonstrates potential uses and methods of digital forensics in a legal investigation. Students were to draw from the knowledge and skills gained from their lab experiences, which were the primary method of instruction throughout the course. In this tutorial, the attendees will see recorded scenes from that class project, and watch demonstrations of several techniques and tools demonstrated in the play.

ALGORITHMIC MUSIC COMPOSITION USING DYNAMIC MARKOV CHAINS AND GENETIC ALGORITHMS*

Chip Bell
ScienceTRAX
4701 Brae Burn Lane
Macon, GA 31210 USA
478-318-2283, chipbell4@gmail.com

ABSTRACT

Music has always been considered to be something uniquely "human" because of the creativity involved in its composition. Although this is the case, almost all composed music follows a set of rules chosen by either the composer or the musical norms at the time. These composers are following a set of rules and, despite their possible complexity, these rules can be followed by a computer to automate the composition process. I present a novel method for algorithmic music composition using two techniques employed in previous computational systems: Markov Chains and Genetic Algorithms. Markov Chains are utilized in this system to choose the next pitch, rhythm, and chord. Genetic algorithms are employed as a search method that finds the set of Markov chains that yield the most pleasant sounding music.

HISTORY AND PREVIOUS RESEARCH

Algorithmic composition is not as novel an idea as one would expect, nor is it restricted to the use of computers in composition. Rule-based and semi-automatic compositional methods can be traced back to the Middle Ages. Works from well-known composers such as Johann Sebastian Bach and Wolfgang Amadeus Mozart have even been observed to employ algorithmic techniques [7]. Furthermore, the rules of counterpoint employed most notably by Baroque composers also provide another example of algorithmic composition present in Western music.

My work is mainly in the merging of two different compositional techniques that have generally been employed separately: Markov Chains, and Genetic Algorithms. Therefore, I will restrict the rest of this section to research concerning these two methods.

Much work has been done on the macro-compositional level using genetic algorithms. In the system created by Bruce L. Jacob [8], music is composed using the idea of creating variations on a single theme. The composer begins with a set of motifs that are pre-composed by a human composer. Phrases are then combined and developed into a full song using a genetic algorithm. The system also evaluates those compositions using an "EAR" module which is also developed using evolutionary techniques.

Milkie and Chestnutt [9] used genetic algorithms to compose fugues entirely from scratch. The system first generated the main recurring melody in the fugue, the "subject", followed by the answer and counter-melody. These new compositions are based upon the subject, and generated completely by the algorithm and with little human input.

Liu utilized Markov chains in the identification of composers [5]. Liu's method records the frequencies of note transitions within a piece, building a Markov chain representative of the piece itself. Using a piece (or set of works), an input work could be compared to a database of composers and then matched to a composer by similarity. This is not composition per se, but it is the same way in which I will construct a song.

Farbood and Schoner [3] employed a dynamic programming approach to note sequencing. A state trellis diagram is made, which considers all possible pitches within a certain range and the number of notes desired in the title song. Using the transition rules from counterpoint, the designers were able to weigh different pitch progressions. These weights were represented as a Markov chain, which was then used to choose the note progression.

MUSICAL FOUNDATION

Music can be analyzed to great depths, especially with some of the larger symphonic works of notable composers like Bach, Beethoven, Wagner, Stravinsky, etc. However, for the scope of this research, an understanding of three basic music theory principles is required Pitch, Chord structure, and Rhythm.

An octave is the interval from two frequencies that are a factor of two in difference (such as 440 Hz and 880 Hz). Western music divides an octave into 12 different pitches named C, C♯/D♭, D, D♯/E♭, E, F, F♯/G♭, G, G♯/A♭, A, A♯/B♭, and B. Juxtaposed pitches are considered to be a half step away from each other. Pitches that are two half steps away from each other are considered to be a whole step away. Multiple pitches can be combined concurrently to form a chord. My research deals with only a certain set of three commonly occurring chords: major, minor, and diminished.

All three consist of a root pitch, with two notes above called the third and fifth respectively. Major chords start from a root pitch, with the third being four half steps above the root. The fifth is seven half steps above the root. The Minor chord is similar to the Major chord, but the third is lowered by a half step. The Diminished chord is much like the Minor chord, but the fifth is also lowered.

Most western music is based of of a series of equally spaced pulses called a beat. Rhythm specifies how melodic pitches fall upon these beats. In this case, I only consider eight durations: A whole note, lasting four beats, a dotted half note, lasting three beats, a half note, lasting two beats, a dotted quarter note, lasting 1 ½ beats, a quarter note, lasting a single beat, a dotted eighth note, lasting a ¾ beat, an eighth note, lasting a ½beat, and lastly a sixteenth, lasting only a quarter of a beat. There are many other commonly occurring rhythms that were omitted however, such as triplets and other tuples. I restrain composition to within the eight previously mentioned values for the sake of simplicity. Moreover, I restrict compositions to four beat measures.

This research leans heavily on music theory so a firm grasp upon its basic principles (that are relevant) is required. For a more complete introduction to this topic, Castellini [1] provides a good reference to all of the music theory topics considered.

MUSIC COMPOSITION ALGORITHM AND COMPOSER LEARNING

The basic idea underlying this system is the idea that music is probabilistic. Certain chord progression are more likely than others. Certain melodic progressions are more likely than others depending on the current chord progression. Because of this probabilistic nature, Markov chains are a natural choice. However, in order to perfect the chains guiding composition, an advanced search technique is employed - in this case a genetic algorithm.

The algorithm guiding composition consists of two different sections, a lower level system based of of Markov chains that guides the selection of next pitch, rhythm, and chord, based upon the current pitch, rhythm, and chord. Markov chains consist of a sequence of state vectors which give the probabilities of transitioning from one state to another [4]. In the case of this system, the current chord, pitch, and note duration i.e. rhythm, are all considered states.

For example, consider the following Markov chain for guiding pitch:

	C	D	E
C	0.5	0.2	0.4
D	0.1	0.4	0.1
E	0.4	0.4	0.5

The pitches in the top row denote possible current states, and the pitches in the first column denote possible future states. So, if our current state, that is to say a pitch, is a D, we have a 20% chance that we will transition to a C, a 40% chance we will stay on a D, and a 40% chance we will move to an E. By picking a random start state and moving through the transitions probabilistically, we can then generate a sequence of pitches which we can treat as a melody.

If we use durations as our step, we end up with the similar results - a sequence of durations forming a rhythm for our melody. We can also apply the same idea to chords,

yielding a sequence of chords that can be used for the underlying chord pattern of the song.

Each type of state is stored slightly differently. Since there are twelve pitches in an octave, a Markov chain for pitch would have 144 entries. However, I also assume that those values are contingent upon the current chord. I m considering three different chord types (mentioned above), and 12 different root values for those chords yielding 36 different chords possible. So, there is a 12 × 12 Markov chain for each of 36 chords, yielding 5184 different probabilities for the Markov chain controlling pitch. Each pitch is contingent upon not only its previous pitch but also the current chord occurring at the time.

Chord progressions are handled more simply. The current chord is simply based upon the previous chord. However a simplification is made. In a traditional jazz lead sheet, the chord can obviously change at any time. However, I restrict myself to one chord per measure. This is not because it is impractical or difficult, but instead it allows me to test the basic functioning of this algorithm without over-complication.

Rhythm works similarly to the chord progression. I restrict myself to only a set of common rhythms. Generally, in modern music the duration of a pitch is never restricted in any way. It can, in fact take on any real value number of beats. However, rhythm generally appears as a small set of different possibilities. I therefore take the liberty in restricting possible rhythms in this system which also aides in simplicity.

Markov chains have been shown to be effective in music composition in the past [6] [3]. However, because the Markov chains involved in composition for this particular system are quite large, it is difficult to attempt to calculate exact values for matrix entries. Instead, a search algorithm is employed, in this case, a genetic algorithm. Genetic algorithms are inspired by the idea of "survival of the fittest." If we can represent possible solutions in our search space as genomes, we can then compare the fitness of a set of individual solutions and choose the best ones to continue on and create a new set genomes. If we have chosen correctly, we would expect our next "generation" to have an higher overall fitness. What follows is an outline of the basic genetic algorithm [2]:

(1) Generate a random population of chromosomes.
(2) If the overall fitness of the population is high enough, terminate.
(3) Calculate the fitness of each of chromosome.
(4) Apply crossover and mutation to selected chromosomes from the current generation to generate a new population of chromosomes. Return to the second step.

In this case, I m considering the actual genomes to be the set of Markov chains that direct composition. Many similar systems in the past employ an automated fitness function that rates composition based upon a training set of songs. This system, however, is different in that the fitness function is the user. Once a generation of composers each write their respective song, the user then chooses the two that it likes the best. These two genomes are chosen to be the parents of the next generation.

Genome strings in the past have traditionally been a bit string encoding a potential solution in the search space. However, in the case of Markov chains this is not necessarily possible. So instead the genetic operator must be designed differently. Because Markov

chains are stored as matrices, we can use the tools of linear algebra to create our next generation. Follows is the algorithm:

(1) Let r be a random number in the range [0, 1).

(2) Let g_1 and g_2 be the user-selected best genomes from the previous population, and g_r a random genome used for mutation.

(3) Let α be our probability for mutation, and k another random number in the range [0, 1). If $k > \alpha$, then the genome will mutate.

(4) Because our genomes are a set of Markov chains and therefore matrices, we can essentially multiply all of the elements in the set by a constant, yielding a new set. So, if $g_1 = \{p_1, r_1, c_1\}$, i.e. a set of Markov chains representing pitch, rhythm, and chord structure, we can multiply elements of this set by a constant and yield a new genome (although it will be the same one because the probabilities will normalize out to the same value).

(5) However, we can create a new genome in this way: $g_{new} = rg_1 + (1 - r)g_2$. If $k > \alpha$, then we will mutate, which we achieve by adding our random genome g_r to g_{new} yielding $g_{new} = rg_1 + (1 - r)\, g_2 + g_r$ for cases in which there is mutation. If we mutated, the sum of the probabilities will not equal 1, so the values will need to be normalized (scaling the probabilities so that they all sum to 1). After normalization, we will have created a child genome. If we perform this process for n different r values, where n is the population size, we will have generated a completely new population based upon our two previous genomes.

The initial seed for a generation can consist of one of three possibilities. The first choice could simply be building random composers and hoping that the initial population shows potential. This method can yield interesting results if the initial population is somewhat good. However, the likelihood of this in the first few generations is low given the complexity of the Markov chains, especially the pitch chain.

Our second choice is using two seed parents. These parents could be two good composers from previous generations, which is the equivalent of saving progress for later, or parent composers constructed from some song in MIDI format. In this case, we see the composer take on traits of both composers. However, there is a major pitfall of this approach. When a composer is first created from some sample input, if there are no transitions for some state, all possible transitions are given equal probability. This can give unsuspected results.

Consider the two following probability vectors from two different Markov chains:

$$\begin{pmatrix} 0.0 \\ 0.5 \\ 0.0 \\ 0.5 \end{pmatrix} \qquad \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix}$$

The first of the two vectors has encountered a transition from the current state. In fact, its input only transitioned to the second and fourth state every time. The second

vector however either had an equal number of each type of transition or there were no transitions at all, so the probabilities were set equally. Consider what happens when recombination occurs with these two pieces of genes. We ll let our random number be r = 0.5. Our resulting vector becomes

$$\begin{pmatrix} 0.0 \\ 0.5 \\ 0.0 \\ 0.5 \end{pmatrix} + \begin{pmatrix} 0.25 \\ 0.25 \\ 0.25 \\ 0.25 \end{pmatrix} = \begin{pmatrix} 0.125 \\ 0.375 \\ 0.125 \\ 0.375 \end{pmatrix}$$

If the second vector was not intentional, that is to say all the probabilities were equal simply because there were no transitions, then we have just introduced unintentional randomness into the system. We are working away from our original goal. The first vector had been trained to only make certain transitions and completely avoid others altogether. By combining it with the second vector, this "knowledge" is somewhat destroyed. This places a limitation on the pairing of sample input. There must be a degree of similarity in the compositions in order for recombination to yield the best results.

The last case consists of a single composer or single MIDI file used as both parents, yielding a set of children composers that are the same as their parents. At this point, we can only rely on mutation to create variation in the population. This may create slower convergence, but allows us to fine-tune composer behavior.

STRENGTHS AND LIMITATIONS

Music is not a clear-cut as one would hope, so designing a system to compose music that a human can appreciate is difficult. In fact, many simplifications were taken in this system to reduce complexity. Simplicity is sometimes a better choice. The initial design of this system was not to create a perfect compositional system, but rather test a method of composition. Once a better understanding of the limitations and strengths of this system is acquired, we can then add functionality and adjust system behavior.

APPLICATIONS

The system s design provides many strengths and applications. A obvious one is that allows any person, regardless of musical background to be involved in composition. Although only a fraction of the population may have musical background enough to compose music, the majority of people do listen to music of some sort and have preferences about the music to which they listen. Fitness in this case is not determined by adherence to a certain musical style, but rather the users opinions, so musical knowledge is not required.

Another application of this system would be to use classical composition as input and attempt to emulate that style of composition. Markov chains have been successfully used for composer identification within a degree of significance [5], and extending that to actual composer imitation would not be that large of a step given the framework

already present. However, classical works are generally not as simple as this system s compositional ability, so extensions would have to made to accommodate multiple voices, dynamics, advanced rhythms, etc.

DISCUSSION

The system s style of composition can be most likened to that of a lead sheet. A lead sheet is a style of music notation that provides only the notated melody and the underlying chord structure. Lead sheets allow performers to take the basic song and modify the style and structure to their liking while preserving the original melody. This leans the system in the direction of jazz, which bases itself heavily on the usage of lead sheets. Jazz performers use the notated melody along with the chord progression to improvise. After using training from skilled performers the system would be able to "improvise" as well in a similar style to its training performer. This could eventually be used as a teaching tool for younger performers first starting improvisation. The system has its set of drawbacks as well, however. Music is difficult to completely formalize mathematically, as it truly is an art form. There is quite a bit of music theory built around western music, but eastern music oftentimes uses different scales, octaves that don t have 12 pitches, unconventional rhythms, etc. This, of course, is not of great concern because this type of music is not as commonplace in the United States and Europe. However, I have still had to omit some more conventional rhythms (such as triplets, lasting a ⅓ of a beat), and chords (such as suspended chords where the third is raised a half step).

Furthermore, the actual algorithm itself has some significant disadvantages. First of all, first-order Markov chains, like the ones used in this system generally do not yield "phrased" passages found in human-composed pieces [6]. This makes Markov-generated music easy to distinguish from human-composed music. However, this can be alleviated with the use of second-order chains, where the upcoming state is dependent upon the previous, but the two state before the previous state. In the context of music composition, the probability of moving to a certain pitch is dependent upon both the previous note and the note before that note. However, this adds an extra level of complexity, but helps to resolve the "meandering" found in Markov-chain generated music.

Another issue is convergence. The genetic algorithm employed in this system utilizes user opinion as its fitness function. This slows the algorithm down by a large margin, because evaluating the most fit individual from a population takes quite a bit of time because the user must listen to all of the population s composed songs and choose the two best ones. Furthermore, the Markov-chains take even longer to converge due to their size. The Markovchain for rhythm is small, 8×8, so it converges quite fast (in the course of 2-5 generations). However, the Markov chain for chord progression is 36×36, and for pitch is $36 \times 12 \times 12$. So, convergence is very slow when combining both user interaction and large Markov chains.

Convergence rate can be improved, however. An seemingly simple approach is to automate fitness evaluation. This approach is more difficult than one would realize because it requires the creation of a fitness function. This is difficult because instructing a computer to know how "pleasant-sounding" music actually sounds is the end goal of this research to begin with. However, this can be achieved to a degree by using a input song,

where fitness is determined by how close the composition is from the input. Another fitness function is using basic rules, such as "pitches should generally attempt to resolve to notes in the current chord," which is fairly reasonable assumption to make. Then fitness is calculated by how closely the composer adheres to those rules.

Convergence may possibly be improved by using a different search method. However, most of these will still require heuristics and we return to the same original problem of automation. One search method may perform better than an another in this case, but one still needs contrive a good heuristic evaluation function in order to progress toward "good" composition.

CONCLUSION

This system provides a good step towards human composition. It has strengths and weaknesses. The system does a relatively good job at composition, but still lacks the organic nature of human-composed music. I have proposed some steps that can be taken toward improving this system's capabilities and reducing its limitations.

The system as of yet does not prove that computers can emulate human composition, and this algorithm may never do so. However, it is still a tool that allows non-musician users to explore composition and play a role in the compositional process. Moreover, it allows us to better understand the differences between "good" and "bad" music. Such ill-defined terms are hard to classify, but by representing "good" composers mathematically, it moves us a step closer to understanding some of the thought process behind human-composed music.

REFERENCES

[1] Castellini, J., *Rudiments of Music*, New York, NY: W. W. Norton and Company, 1962.

[2] Coppin, B., *Artificial Intelligence Illuminated*, Sudbury, MA: Jones and Bartlett, 2004.

[3] Farbood, M. and Schoner, B., Analysis and synthesis of Palestrina-style counterpoint using Markov chains, *Proceedings of International Computer Music Conference,* 2001

[4] DeFranza J., Gagliardi D., *Introduction to Linear Algebra with Applications*, New York, NY: McGraw Hill, 2009.

[5] Liu, Y.-W., Modeling music as markov chains: composer identification, http://esf.ccarh.org/ 254/254_LiteraturePack1/ComposerID_Liu.pdf, posted June 11, 2002.

[6] Roads, C. 1996. The Computer Music Tutorial , 1 ed. MIT Press.

[7] Collins N. and D'Escrivan J., *The Cambridge Companion to Electronic Music*, New York, NY: Cambridge University Press, 2007.

[8] Jacob, B. L., Composing with genetic algorithms, *Proceedings of International Computer Music Conference,* p. 452-455, 1995.

[9] Milkie, E. and Chestnutt, J., Fugue generation with genetic algorithms, http://www.cs.cornell.edu/boom/2001sp/Milkie/index.html, retrieved October 1, 2010.

WANT TO BECOME A MUSIC COMPOSER? TRY WITH INTERMEDIATE PROGRAMMING SKILLS*

Lakshmi Prayaga
Computer Science Department
University of West Florida
Pensacola, FL 32514
(850) 474 - 2973
lprayaga@uwf.edu

ABSTRACT

Designing interesting assignments and projects is a task that every instructor attempts to accomplish in an effort to keep students motivated and engaged. This paper describes the design and implementation of a computer generated music assignment for students enrolled in a second course in java. A discussion on student performance in this assignment and their reactions are also included in the paper.

INTRODUCTION

Designing interesting assignments is a task that every instructor attempts to accomplish in an effort to keep students motivated and engaged. Motivating students also serves a secondary purpose, increased enrollment. Additionally motivated students also spread positive feedback that encourages future students to enroll in these courses. With this background we were exploring interesting projects and assignments for a second course in programming "Intermediate Programming" with Java (prior to Data Structures) for our CIS and IT tracks. We included the popular card games but were still looking for something more exciting and fun that students could categorize as "cool". The music composer project was designed to address this need.

LITERATURE REVIEW

In an attempt to increase, improve and maintain student motivation in computer science courses, several examples of interesting curriculum, assignments and grading practices including those based on robotics, networks, and gaming among others have been used in computer science programming courses ([5] Lawrence, 2004; [1] Angotti et al, 2010; [8] Swoboda, 2011) to provide a context for teaching and learning ([4] Forte and Guzdial, (2004)). However there is not much literature on the use of music in assignments as a vehicle to motivate students and pique their interest in computer science. Some universities, such as [2] Columbia, [9] University of California, Irvine, and [6] Ohio State School of Music, with strong music research activity, offer courses geared towards computer generated music. But there are not many cases where music has been embedded for teaching and learning in early programming courses for the CIS or IT tracks. Most recently this trend seems to be changing as evidenced by the research and work of ([7]Ruthman, Heines and Greher (2010)) who designed a multidisciplinary course for Arts and Computer Science majors using scratch, and ([3]Ericson and Guzdial (2011)) who designed assignments incorporating data structures (linked lists) to play music.

This paper describes an experimental assignment in a Java 2 or Intermediate programming course, which incorporates the basic idea of computerized orchestra by implementing inheritance, abstract classes, and arraylists. During the fall of 2010, in an Intermediate Programming (Java) course we designed an assignment that used the ideas of inheritance, abstract classes and GUI to mimic a music orchestra. The design, implementation and an example for this project are described in this paper.

DESIGN OF THE ASSIGNMENT

Overview of the project

The basic objective of the assignment was to write a java program that creates the basics for simulating an orchestra with computer generated music. The program generates music depending on the instruments chosen randomly.

The assignment had two phases. In the first phase students were given a UML diagram listed in figure 1 followed by a discussion on what each class is designed for including the methods. Students had to implement the project using the given UML diagram as a reference. The functional requirements for this phase were

- Let the computer randomly pick an instrument from a given list (used only two instruments, violin and piano for simplicity)
- Depending on the randomly chosen instrument a random sound file must be chosen from a list of available notes and have the instrument play that note.
- Create an ArrayList of eight instruments and have each of these instruments play a note that was chosen from a list of existing notes with a delay programmed between the notes.

There was no user interactivity built into this phase. This was a required assignment for all students.

The second phase was to create a GUI for this project. The functional requirements for this phase

- Let the user choose the type and number of instruments from a given list of instruments displayed
- Let the user choose the number of notes to play
- Have the program play the notes based on user choice

Phase 2 was a separate assignment where students had an option to choose the project for which they would like to build a GUI.

Student Learning Outcomes

- To implement inheritance in Java to produce reusable classes.
- To gain experience working with several classes in one program.
- To use ideas of inheritance, abstract classes,
- To gain experience with using object references.
- To implement a solution involving building classes that extend other classes, use ArrayLists and a random number generator

UML Diagram and Project description

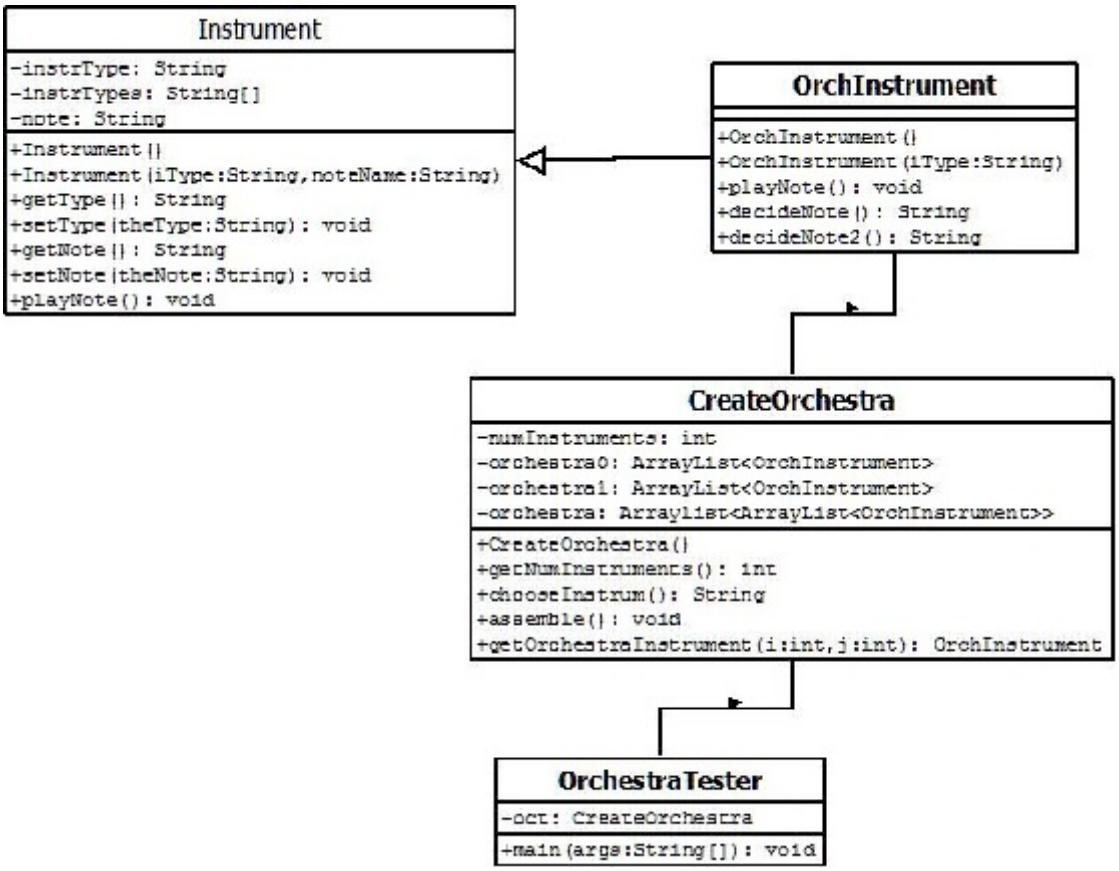


Figure 1 - UML diagram for the Orchestra program

As shown in the UML diagram, the project consists of 4 class files: instrument, orchInstrument, createOrchestra and OrchestraTester.

The instrument class defines a music instrument and has typical getters and setters for accessing and setting instrument type and notes. The constructor takes two strings (instrument type and note type) to instantiate an instrument. The instrument class also has an abstract method playNote()

The OrchInstrument takes the type of instrument and depending on the type assigns a note to that instrument. For example, if the instrument is a piano, a random note from a set of piano notes is assigned to the variable "note" and if it a violin, a random note from a set of violin notes is assigned to the note. The abstract method playNote from the instrument class, takes the note assigned to the instrument type and plays the note.

The CreateOrchestra class assembles the orchestra by creating an array of arrays with instruments mimicking a 2D array for a conceptual representation of an orchestra. We designed the orchestra to have 2 arrays, each having 4 instruments.

The OrchestraTester class has a nested loop to loop through the elements in the two arrays and prints the instrument type and plays the note.

RESULTS AND OBSERVATIONS

Assignment number 3 during the Intermediate programming course was to design and implement the music orchestra program. The goal of this assignment was to achieve the functionality of a music orchestra per the definition of the assignment. Out of twenty seven students in the course, twelve students scored partial credit at various levels (58% to 95%) and the rest of the students fulfilled the requirements for the program and scored a perfect score. Five students chose this assignment as their last assignment (assignment number 4) for which they had to build a GUI to one of the projects they worked on during the semester. Many students liked the idea of playing computer generated music with the notes they recorded vs. using the default musical notes provided for the assignment. Some students also used music that was available commercially. Students wanted to get a good quality of music and provide a simple GUI for people to interact with their project. Students were also interested and motivated to exchange views with their peers on this assignment on the aesthetics of music including sound quality, harmony, and melody. They experimented with changing some parameters including the quality of sound files and adjusting the delay between notes to produce a more melodious tune. Given this data we believe that this experimental assignment was not simply viewed by students as one more assignment to complete, but it was interesting and motivated students to research further on this topic.

Figure 2 is the output of the program from assignment 3, phase 1 described above. Each time a note is played as music, it's name and the instrument that played that note is printed to the output window. Figure 3 is an example of assignment 4 where assignment 3 is modified and a GUI was added to the basic music program. The program now has an interactive GUI where the user can choose the type of instruments, the number of instruments and the number of note to be played. The program then chooses the appropriate notes (based on the instrument type) and plays the notes.

```
Instrument: piano
Note: 25.wav
Instrument: piano
Note: re.wav
Instrument: piano
Note: 21.wav
Instrument: piano
Note: do.wav
Instrument: piano
Note: 11.wav
Instrument: violin
Note: so.wav
Instrument: piano
Note: 10.wav
Instrument: piano
Note: 11.wav
```

Figure 2 - Output from assignment 3

Figure 3 - GUI for assignment 4

CONCLUSION

Finding interesting assignments is crucial for student engagement and motivation. The music composer assignment was one such project that students could relate to and feel motivated by. This assignment also encouraged communication among students as they were interested in discussing and appreciating the various aspects of coding and music, to make the computer generate melodious and harmonious music. This implies that the quality of student work improves when they are motivated, creating and fostering a student driven learning environment rather than an instructor driven teaching environment. Future efforts include designing interfaces and GUIs to allow users to choose a scale, type in strings of music notation, specify length of each note and delays between notes, and include for overlapping notes to generate more elaborate music that is closer to live music.

Code for this project can be made available to interested parties

ACKNOWLEDGMENTS

We would like to acknowledge an undergraduate student Cody McDavid for his contribution towards generating the GUI project used as an example in this manuscript

REFERENCES

[1] Angotti, R., Panitz, M., Sung, K., Nordlinger, J., Goldstein, D., (2010), (as cited in Hillyard et al, 2010) Game Themed Programming Assignments for Faculty, A Case Study, SIGCSE, 2010, retrieved March, 25th, 2011, from: http://faculty.washington.edu/ksung/pub/2010_sigcse_fp075-hillyard.pdf

[2] Columbia University http://music.columbia.edu/cmc/courses/g6611/spring2011/

[3] Guzdial, M., Ericson, B., (2011), Listening to Linked Lists: Using Multimedia to Learn Data Structures, SIGCSE workshop

[4] Forte, A. and M. Guzdial. *Computers for Communication, not Calculation: Media as a Motivation and Context for Learning*. In *Hawaii International Conference on System Sciences*. 2004.

[5] Lawrence, R., (2004), "Teaching Data Structures using competitive games", IEEE Transactions on Education, vol 4, pages 459 - 466

[6] Ohio state school of music. (http://www.musiccog.ohio-state.edu/Music824/descript.html

[7] Ruthmann, A., Heines, J., Greher, G., Laidler, P., and Saulters, C., (2010). Teaching computational thinking through musical live coding in scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education* (SIGCSE '10). ACM, New York, NY, USA, 351-355. DOI=10.1145/1734263.1734384

[8] Swoboada, N., Bekios, J., C., Baumela, L., Lope, J., "An Introduction to AI course with Guide Robot Programming Assignments", SIGCSE 2011, retrieved March 25th, 2011, from: http://db.grinnell.edu/sigcse/sigcse2011/Program/viewAcceptedProposal.pdf?sessionType=paper&sessionNumber=136

[9] University of California, Irvine, Interactive Arts Program: http://music.arts.uci.edu/dobrian/IAP2010/syllabus.htm),

DEVELOPMENT AND USE OF AI AND GAME APPLICATIONS IN UNDERGRADUATE COMPUTER SCIENCE COURSES*

Eman El-Sheikh and Lakshmi Prayaga
Department of Computer Science
University of West Florida
Pensacola, FL 32514
850-474-3074
eelsheikh@uwf.edu, lprayaga@uwf.edu

ABSTRACT

Gaming and Artificial Intelligence (AI) are both seen as exciting domains by many Computer Science students. Many universities are using these two areas as a means to attract and retain students in Computer Science through course work and research projects. In this paper we discuss the development of Artificial Intelligence and game applications by students in undergraduate game and AI programming courses, and how these applications can be integrated into Computer Science courses to improve student engagement and attainment of learning outcomes.

1 INTRODUCTION

Using game-based curriculum to teach Computer Science concepts has become increasingly popular during the past several years [6]. Several universities have introduced courses with a gaming aspect and also developed minors and majors in game development to motivate students and generate student interest in Computer Science concepts at all levels including introductory programming courses, data structures, and algorithms [1, 3, 8]. In this paper we discuss the development of Artificial Intelligence (AI) and game applications by students in undergraduate game and AI programming courses, and how these applications can be integrated into Computer Science courses to improve student engagement and attainment of learning outcomes.

Our approach is novel in that it includes two phases: the first phase involves the development of AI and game applications for educational uses by our own undergraduate students. These applications are educational in that they include an interactive simulation component which visually allows users to experiment with and learn how the algorithms work. Additionally, these applications are also applied within several gaming scenarios: (a) application of Dijkstra's algorithm in navigating through a terrain and finding a shortest path to reach a destination, and (b) application of the A* algorithm in a Tic-Tac-Toe game. The second phase identifies the use of these applications in a variety of undergraduate courses. These interactive applications have the potential to increase students' motivation and learning outcomes.

The rest of the paper is organized as follows: Section 2 describes related work in the areas of (a) using games and AI to motivate students towards Computer Science and (b) using games as a medium to teach Computer Science concepts such as AI. Section 3 presents the development of two AI and game applications, and section 4 discusses how these applications can be used to support CS curricula. Finally, the paper concludes with practical considerations for adoption of the proposed framework and opportunities for future work.

2 LITERATURE REVIEW

Recent trends indicate that educators and researchers have been looking at gaming as a tool to both motivate students towards Computer Science in general [4], and learn about specialized topics in CS such as AI and implement them in game development projects. Wong, Zink, and Koenig [8] have been successful in developing courses where students learned and implemented A* algorithm in great detail. DeNero and Klein [1] describe a project-based course with a Pac-man game interface to teach and learn foundational AI concepts through game programming at the University of Berkley. DeNero and Klein provide the necessary scaffolding through a shell–based environment as a starting point so that students do not get overwhelmed with the initial programming aspects and fail to focus on the AI topics. Their findings indicate that students reacted positively to this approach in mastering the AI content and then used it to extend the gaming framework provided using these new concepts since the educational context was a well known and exciting gaming environment. Dodd's [2] study suggests that introducing AI in early CS courses can be a hook to attract majors into CS curriculum. Harris and Jovanovic [5] propose a detailed design for a complete introductory programming course using game programming in an effort to attract and retain students in CS.

This trend is also true at the University of West Florida. In our Game Programming 1 course, which is primarily 2D game development with Flash and is open to all students, many students were interested in the AI aspects of game development. Several students explored core AI methods for game development, including adaptive search algorithms, increasing levels of difficulty based on player progress, path finding algorithms, and strategic planning using decision trees. A gaming environment provided the context for them to explore these concepts.

3 DEVELOPMENT OF AI AND GAME APPLICATIONS

Several educational applications were developed by students in our undergraduate courses. The application for Dijkstra's algorithm described below was developed by students enrolled in Game Programming 1 in the spring 2010 semester. The course, which is open to all students, had 28 students enrolled, many of whom were not CS majors. Students were free to choose the games they would like to implement given an overview of the requirements. Surprisingly many students also chose to include some AI component in their projects. All of the students implemented decision trees in their projects which resembled a scavenger hunt. Two students implemented Dijkstra's algorithm, with other students implemented decision trees in controlling aircraft landing, aircraft conflict situations, and resource allocation situations.

Dijkstra's algorithm also has a tutorial aspect to it in that it provides an interface that is simply a basic interface of nodes connected by line segments with weights and the user predicts the next node to go to given an initial configuration. The application of this algorithm is provided in a gaming context. Such a framework provides users with both a conceptual understanding of the algorithm and allows them to see its application in an interesting context.

The application for the A* algorithm was developed by a student enrolled in Artificial Intelligence, an elective course for CS undergraduate students. Students worked on a topic of their interest for the project including an agent architecture, robotic path planning, machine learning, and intelligent networking security systems. The A* application was developed by one of the students as an algorithmic simulation tool for educational and evaluative purposes.

3.1 Application for Dijkstra's Algorithm

Dijkstra's algorithm offers a shortest path solution for a graph with non-negative edge path costs. The application developed is a simulation of Dijkstra's algorithm. It has two main aspects. The first is a tutorial on how Dijkstra's algorithm finds the shortest path through a graph. The second section of the program is an application of the algorithm to a gaming situation (guiding a wizard through a terrain). The user is prompted to guide a character through a simulated environment. Once the user has successfully guided the character to the ‘castle' the program then demonstrates how the algorithm would behave.

The tutorial first displays a graph that has nodes connected by edges. These edges are weighted. In the beginning of the tutorial all of the nodes are set to infinity, with the exception of the starting node. The user is prompted to click a button in order to start the tutorial. As the user continues through the tutorial, text is displayed and an explanation of what the algorithm is doing at each state is displayed. Figures 1 – 6 are sample screen shots from the application.

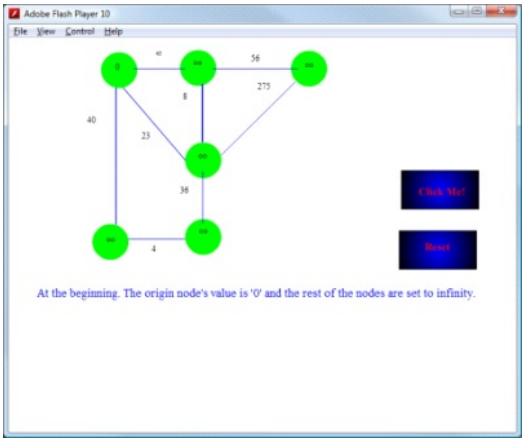


Figure 1 – Initial values of nodes and weights

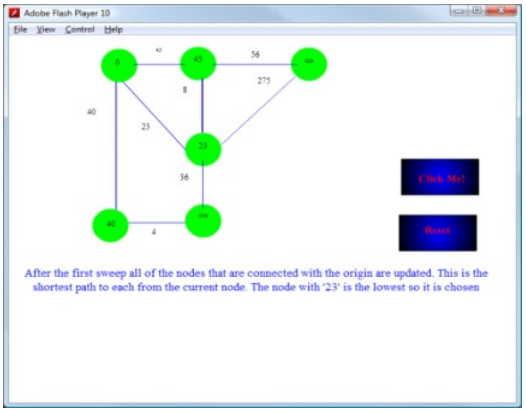


Figure 2 – Values of nodes after first pass

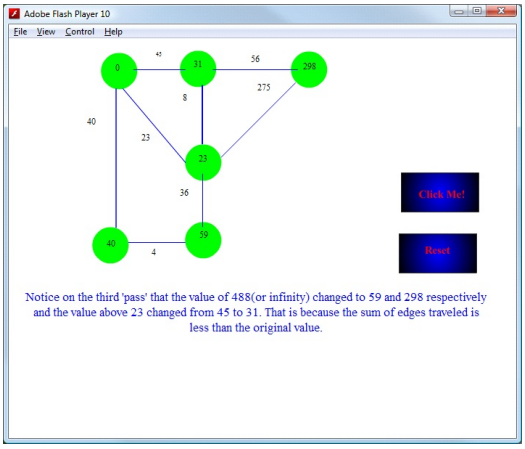


Figure 3 – Values of nodes after third pass

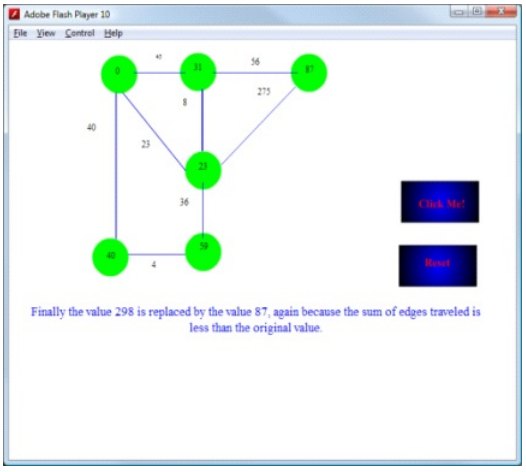


Figure 4 – Final state of nodes

Figure 5 – Guiding wizard through terrain

Figure 6 – Final location of wizard in terrain

The Dijkstra's algorithm application simulation was presented at our university's student research symposium last year (2010), and was selected as the Audience's Choice Award. The audience included members from the university and outside the university. Many participants were able to use the application to learn about the algorithm. Students involved in building the application developed a good understanding of the algorithm and enrolled in the AI course at the undergraduate level. These informal results suggest that gaming provides a motivating and exciting context for students to learn about algorithms and AI, confirming previous studies [1].

3.2 Application for A* Algorithm

The A* search algorithm is widely use in many applications, including internet search agents, navigation in computer games and robots, routing, scheduling and planning, and is also the most popular choice for path finding [7]. The A* algorithm application serves as an algorithm visualization tool by allowing users to trace the execution of the algorithm using a graphical user interface. The application also serves as an educational tool, allowing students to explore how the algorithm works. The application provides several main functions:

- Taking different measurements of the execution of the algorithm of the algorithm.
- Tracing the execution of the algorithm on a graphical interface.
- Implementing and solving a game using the algorithm.

The application allows students to trace the execution of the algorithm graphically as shown in figure 7, which shows a board of 16 x 10 squares representing a set of nodes. The user can set the start node and goal node, and then run the algorithm to trace its execution graphically. Figure 8 illustrates that the program found the optimal path and highlighted it in yellow. It also shows the opened nodes in blue, limit nodes opened in grey, and number of each kind of node that was opened.

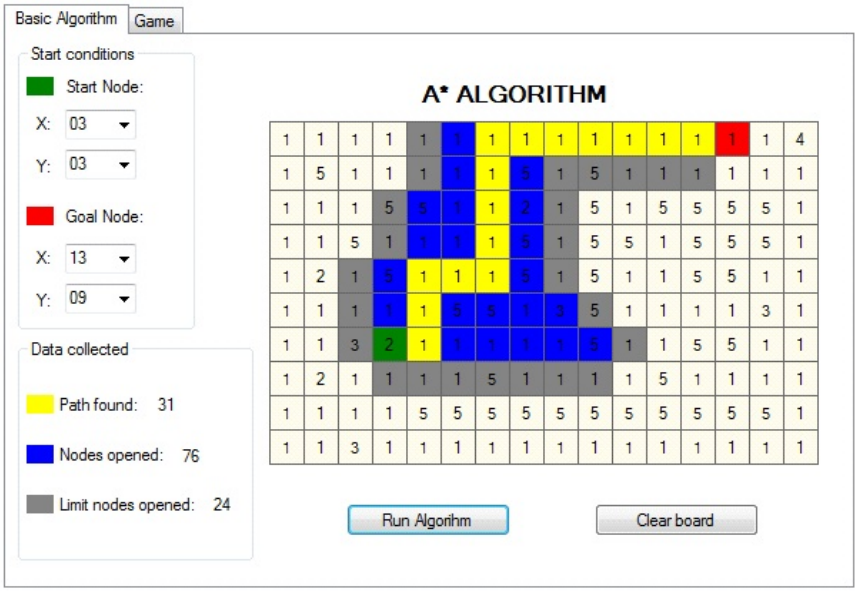


Figure 7 - A* Application Screenshot

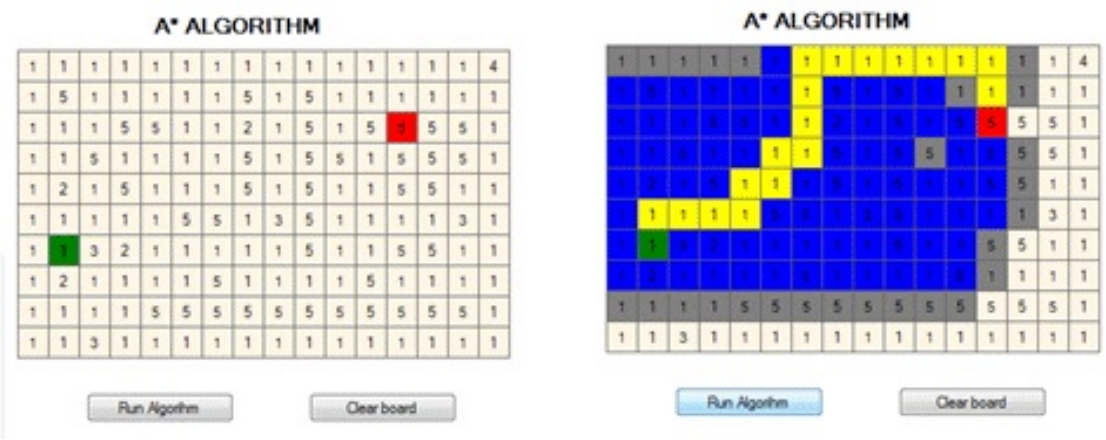


Figure 8 - A* Application Before and After Running the Algorithm

The application also allows students to play a tic-tac-toe game, started by either the user or computer. Using this panel, the user will be able to select between both options. The game, shown in figure 9, allows each player to play in turn, and shows the game result when the game ends. The A* application was also presented at our university's 2010 student research symposium, and received the Best Simulation Award. Users were able to use the simulation tool to learn about the A* algorithm and apply it to a game.

Figure 9 - Tic-Tac-Toe Game Start Screenshot and Game End Screenshot

4 INTEGRATION AND SUPPORT FOR COMPUTER SCIENCE CURRICULA

The applications described above can be integrated in a variety of computing programs to support student attainment of the programs' learning outcomes. For example, such applications can be used to support undergraduate programs in Computer Science, Computer Information Systems, Software Engineering, and Information Technology. Within such undergraduate programs, the applications can be utilized to engage students in a variety of computing and programming courses. We describe the use of these applications to improve learning outcomes in three types of courses:

1. Algorithms courses

   The applications can be used to illustrate the implementation and analysis of fundamental algorithms. Students can use these applications to simulate the execution of algorithms such as Dijkstra's and A*, and evaluate their performance including correctness and efficiency.

2. Artificial Intelligence courses

   The applications can be used to demonstrate the use of AI algorithms to solve problems, and to analyze their performance. For example, students can use the applications to solve a maze, and compare their performance on several dimensions, including time and space requirements, completeness, and optimality. Other types of problems that can be solved using the applications include board problems (e.g., n-queens or 8-puzzle), games (e.g., tic-tac-toe or checkers), or routing problems (e.g., network flow or traveling salesperson).

3. Gaming courses

   The applications can be used to demonstrate core programming techniques for game development. For example, students can use the applications to explore the role of finite state machines and algorithms such as A* and Dijkstra's algorithms for developing games. Students can simulate these algorithms as part of a game environment, and evaluate their impact on game characteristics such as real-time performance and accuracy.

Whether in courses such as those described above or other appropriate courses, the applications can be easily integrated into the curricular plan. After the topics associated with the use of the application are covered in the course, students can be given a simple overview of the application and then asked to experiment with the application in the lab or on their own time. The application can also be linked to specific course assignments. For example, students can be asked to use the application to compare the performance of several algorithms and report on which algorithm was most appropriate for solving specific problems. The use of these applications to support courses in undergraduate computing programs provides students with interactive learning experiences to help engage students and improve their learning outcomes.

5 CONCLUSIONS AND DISCUSSION

To address declining enrollments and other challenges, many Computer Science departments are exploring creative ways to recruit and retain more students, by sparking

their interest in CS topics and engaging them in the learning process. We described a novel two-phase approach for the incorporation of game-based content into the curriculum to motivate students and help address these challenges. The first phase involves the development of AI and game applications for educational uses by our own undergraduate students, and the second phase involves the use of these educational applications in various undergraduate courses. The proposed framework has the potential to increase students' motivation and learning outcomes.

Our experiences in teaching game programming and AI indicate that both these domains are topics of interest and fascination for students. These topics provide an inherent motivation for students to continue learning about these and other Computer Science topics. Integrating these two broad areas to introduce both beginning and advanced concepts of Computer Science can help improve the recruitment, retention, and engagement of students in CS programs.

6 ACKNOWLEDGMENTS

We would like to acknowledge our undergraduate students, Billy Abston, David Brett, and Carlos L. Calvo, for their contributions to the development of these applications.

7 REFERENCES

[1] DeNero, J., Klein, D., The Pac-man projects software package for introductory artificial intelligence, *Symposium on Educational Advances in Artificial Intelligence (EAAI)*, Model Assignments Track, 2010, www.denero.org/content/pubs/eaai10_denero_pacman.pdf, retrieved March 29, 2011.

[2] Dodds, Z., AI assignments in a CS1 course: reflections and evaluation, *Journal of Computing Sciences in Small Colleges*, 23 (6), 262-271, 2008.

[3] Gudzdial, M., Soloway, E., Teaching the Nintendo generation how to program, *Communication of the ACM*, 45 (4), 17-21, 2002.

[4] Harbour, J., *Visual Basic Games Programming with Direct X*, Portland, OR: Premier Press, 2003.

[5] Harris, J., Jovanovic, V., Designing an introductory programming course using games, *Issues in Information Systems*, 11 (2), 104-113, 2010, http://www.iacis.org/iis/2010_iis/Table%20of%20Contents%20No2_files/104-113_LV2010_1444.pdf, retrieved April 3, 2011.

[6] Masuch, M., Nacke, L., Power and peril of teaching game programming, *Proceedings of CGAIDE: Computer Games: Artificial Intelligence, Design and Education (5th Game-on International Conference*, Reading, UK, 347-357, 2004.

[7] Russell, S. J., Norvig, P., *Artificial Intelligence: A Modern Approach*. Upper Saddle River, N.J: Prentice Hall/Pearson Education, 2010.

[8] Wong, D., Zink, R., Koenig, S., Teaching artificial intelligence and robotics via games, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10),* 2010, www.aaai.org/ocs/index.php/EAAI/EAAI10/paper/download/1607/2339, retrieved March 29, 2011.

APOGEE – A TOOL FOR AUTOMATED GRADING PROGRAMMING PROJECTS*

CONFERENCE WORKSHOP

Xiang Fu, Mike Powell, and Mike Bantegui
Computer Science Department
111 Hofstra University Hempstead, N.Y. 11549 516- 463-4787
Xiang.Fu@hofstra.edu, michaelpowellcs@gmail.com, and
mbante1@pride.hofstra.edu

Boris Peltsverger, Rustico David, and Lei Wang Computer Science Department 800
Georgia Southwestern State University Drive Americus, GA 31709 229-931-2100
Boris.Peltsverger@gsw.edu, rdavid@radar.gsw.edu, and lwang@radar.gsw.edu

ABSTRACT

This automated grading tool, APOGEE, implemented a trial-and-failure learning strategy. The idea is simple: given a sophisticated course project, let students try project submissions as many times as they want before the project deadline. For each submission, a thorough inspection is performed by an automated grading system called APOGEE. A student project has to accomplish not only the functional requirements but also many desired quality attributes such as robustness and security. A grading report is generated instantly within several minutes. The report starts with a summary, e.g., the grade and timestamp. Then the report contains an itemized list of test cases. The running results of test cases are classified into two categories: the passed and the failed. For each test case, clicking the corresponding link redirects a viewer to the test case report. This workshop provides an opportunity for educators to explore the automated grading tool and discuss requirements for its implementation in their colleges/universities.

A general Trial-and-Failure approach in teaching of programming classes will be discussed [1]. The authors will present an Automated Grading Tool – APOGEE, which is based on the mentioned above approach [2]-[5]. This tool was developed with support of the NSF Grant (DUE-0836859, 0837275, and 0837020 - Collaborative Research: A Trial-and-Failure Project Tutoring System) and currently implemented in nine universities and colleges. APOGEE consists of five major components: (1) a class management module, (2) a project specification tool for instructors to create test cases, (3) a grading

engine that invokes the WatiN library for evaluating student projects, (4) a back-end database, and (5) a project report generator. At this moment, APOGEE grades web programming projects only.

The participants, during the workshop, will be able to run APOGEE on the virtual labs, located in Hofstra or Georgia Southwestern State University. The students, research assistants, will help to set it up and run APOGEE; hands on materials will be distributed as well. APOGEE is a free tool and can be downloaded along with the instructor and student manuals from the following website [6].

REFERENCES

[1] C.C. Bonwell and J. A. Eison, "Active Learning: Creating Excitement in the Classroom," *JB ASHE Higher Education Report*, No. 1, George Washington University, Washington, DC, 1991.

[2] Xiang Fu, Boris Peltsverger, Kai Qian, Lixin Tao, and Jigang Liu, "APOGEE – Automated Project Grading and Instant Feedback System for Web Based Computing," in *SIGCSE'08*, 2008.

[3] Xiang Fu, Kai Qian, Boris Peltsverger, Kent Palmer, Chong-wei Xu, Yu Zhang, "New Trends in Automated Project Grading for Web Programming Classes," (poster) in *SIGCSE'10*, 2010.

[4] X. Fu, K. Qian, K. Palmer, B. Peltsverger, B. Campbell, B. Lim, and P. Vogt " Making Failure The Mother of Success," *Proceedings of the 40th Frontiers in Education,* Arlington, Virginia, October 27-30, 2010.

[5] X. Fu, B. Peltsverger, "Workshop - Automated Grading," *The Course, Curriculum, and Laboratory Improvement (CCLI) Program Principal Investigators (PI) Conference*, January 26-28, 2011, Washington, DC.

[6] Xiang Fu, "APOGEE Project Website", *available at* http://people.hofstra.edu/Xiang_Fu/XiangFu/Projects/APOGEEHomepage

PROGRAM BY DESIGN: GRAPHICS-FIRST PROGRAMMING WITHOUT DROWNING IN SYNTAX*

TUTORIAL PRESENTATION

Stephen Bloch
Math/CS Dept, Post Hall
Adelphi University, One South Avenue
Garden City, NY 11530
516-877-4483; fax 516-877-4499
sbloch@adelphi.edu

ABSTRACT

The "Program By Design" (neé "TeachScheme!") curriculum has been used to great success in middle schools, high schools, community colleges, liberal-arts colleges, and research universities, and won Matthias Felleisen the two highest awards in computer science education, the ACM Karlstrom award and the SIGCSE Distinguished Educator award. The curriculum starts with a beginner-friendly IDE (DrRacket) and a simple language (a tiny subset of Scheme), emphasizing an explicit, concrete problem-solving strategy and motivating central CS and math concepts (variables, functions, composition, data types, classes, polymorphism, etc.) with graphics and animation. Using this curriculum, I routinely teach non-CS-majors to write separable-model, event-driven GUI programs involving higher-order functions and recursive traversal of linked data structures, all within their first semester. For CS majors, we then show (in a second semester) how the same concepts and strategies apply in a more difficult language like Java.

PRESENTER'S BACKGROUND: Stephen Bloch has taught beginning programming for eighteen years, including both CS1/CS2 for majors and a first-and-last programming course for non-majors. He was introduced to the present approach at a 1998 workshop by Matthias Felleisen, and first implemented it in 1999-2000. Since then he has applied the same principles variously in Scheme, Java, and C++ (hint: it's easiest in Scheme, and hardest in C++). He has led numerous NSF-funded week-long workshops to train high

---

school and college faculty in these techniques, and has just received a "CS4HS" grant from Google to conduct similar workshops in the future.

INTENDED AUDIENCE: High school, College and University faculty who teach first-year programming courses for either majors or non-majors. Students with an interest in teaching are also welcome.

MATERIALS: handouts. All software and textbooks used are available for free on the Web.

AUDIO/VISUAL/COMPUTER REQUIREMENTS: Desktop or laptop computer for each participant. Free software to be downloaded and installed in advance (available for Mac, Windows, Unix).

TEACHING AN APPLIED HCI COURSE USING MULTIPLE, INDIVIDUAL, HIGH FIDELITY, PROGRAMMING PROJECTS*

Ron Zucker
Dept. of Computer and Information Sciences
East Tennessee State University
Johnson City, TN USA
(423) 439-6405
zucker@etsu.edu

ABSTRACT

Human Computer Interaction (HCI) courses typically teach HCI theory, supported by one or two paper-prototype-centered prototyping projects. This paper describes an applied HCI course that attempts to bridge the gap between HCI theory and technical application. The course, which was given in spring 2010, was divided into three areas of software interfaces: form based, spatial, and a hybrid of form based and spatial. Examples of the projects are given with a rationale for each. Students' reaction to the course, on the whole, was quite favorable, with 60% strongly agreeing and 40% agreeing that the approach was beneficial. Possible improvements to the course's format include requiring an introductory course on HCI as a prerequisite and reducing project scope to reduce student workload.

INTRODUCTION AND PURPOSE

HCI courses typically consist of an initial series of units on underlying psychological factors of cognition, followed by further units on approaches to design. Students are usually required, individually or within a team, to design an interface for a particular application [2][4][5][6].

Many courses use prototypes to develop their interfaces. These prototypes come in many forms: verbal, paper, interactive sketches, and working prototypes. The verbal and paper approaches are generally the most popular, enabling the students to create an

---

interface not restricted by technical abilities. The paper prototype may be low fidelity, consisting of rough sketches, or high fidelity, consisting of more elaborate, often computer generated, designs. Some courses also require one or two working prototype projects, usually as a team effort, to give the students experience in realizing their design.

One of the most important distinctions between this course and traditional HCI courses is the number of assigned projects, six versus the more usual one or two, and the fact that this course involves individual effort rather than a team effort. Class lectures covered a series of typical scenarios for software interfaces, progressing from graphical widget based interfaces to spatially oriented interfaces and finally to a hybrid of spatial and graphical widget based interfaces. Here, "graphical widget based interface" refers to an interface composed of the usual widgets common to most graphical Interactive Development Environments (IDE), e.g. labels, text boxes, radio buttons, check boxes, etc.. Spatial interfaces, sometimes referred to as direct manipulation interfaces, are based on the manipulation of graphical objects on a canvas to allow users to visualize the relation of objects via placement [3]. A spatial interface requires little or no typing and is based on gestures using a mouse, touchpad, or similar device. Finally, hybrid interfaces marry spatial objects to supporting text or data, which may be entered via a dialog based form.

STUDENT BODY

This course was offered as a junior/senior level course with the Intro to Computer Science (using Java) as the only prerequisite. The students were assumed to have knowledge of, and experience in, inheritance and polymorphism. While some students had taken a traditional HCI course, this was not a prerequisite for this course. There were 14 students who originally signed up for the course, 13 males and one female. Eleven students completed the course, all males.

COURSE PHILOSOPHY

Applied HCI was developed to fill the gap between theoretical HCI concepts and applied implementations. Many HCI courses concentrate on interface design and relegate implementation to one or two projects. In this course the students were required to implement six smaller projects providing them with a "toolbox" of techniques.

The course's learning outcomes were as follows:

- Students will learn to write programs using the basic features of the graphical user interface as opposed to the character based interfaces they have seen previously.
- Students will learn the basics of event-based programming, the message loop, and system message routing.
- Students will learn to use a modern class library through in-class instruction and through their own explorations initiated through software development assignments.

COURSE PROJECTS

The course was split into three pairs of projects, with approximately two to three weeks per project. The NetBeans IDE, which is free and provides GUI support, was used for developing projects. The six projects were divided into three areas: textual, spatial, and hybrid, each area consisting of two projects: the first of each pair introduced the area's basic concepts, and a second, closely related project, for reinforcement, refinement, and to help teach transfer. Transfer is an HCI term used to describe the transference of user experiences from one interface to another.

The areas and their respective projects are described in the following sections.

Textual Projects

The textual projects were designed to introduce HCI applications allowing the students to reuse existing IDE supplied widgets.

The first textual project was to create an application to tutor students in basic HTML. The application was to provide a tutorial (as a textbox), a target sample, and a practice area showing raw HTML and the resulting rendered information. The students were shown a basic interface developed by the author and were given lectures to aid them in the development of the interface. Much care was taken to encourage students to develop their own interface. A class critique of the basic interface resulted in a great deal of participation and involved students in the creative process. As a result, the students produced some remarkable results, and each one was different from the others.

The second textual project required the students to develop a tool for developing the tutorial used in the first project. One of this project's goals was to teach students to develop tools that would separate user content from code content. A second goal in this project was to introduce and demonstrate the concept of transfer. Students were required to show that the tutorials could be created in a useful way for importing into the first project, but were not required to modify the first project to accept the new data.

Spatial Projects

Spatial projects were introduced to force students to create their own widgets and to allow users to interact with the computer solely through pointing devices.

The first spatial requirement was to construct a simple network diagramming tool consisting of three types of components: servers, routers, and terminals. Apart from the use of a keyboard to start an application, the mouse was the only input device allowed for this project. Interfaces were required to support the use of clicks and/or dragging actions to create, move, or delete components.

In the first spatial project, components were placed on a blank background. The second spatial project required students to create a room editor module. This module was required to support the creation, resizing, moving, and deleting of simple rectangular "rooms"; the placement of components created in the first spatial project in these rooms;

and, finally the components' repositioning and removal. Students were also required to provide the ability to print the room layout with its respective components.

Hybrid Projects

The last two projects involved a hybrid of textual and spatial interfaces. Hybrid projects made the students consider the combining of textual data with spatial environments in order to provide more detail as it related to the spatial objects.

Hybrid projects run the risk of cognitive interrupts as the user must switch from the mouse to the keyboard and back. Cognitive interrupts distract the user from the main task and should be minimized where possible. The first hybrid project was to create a UML class diagram allowing users to position class and interface objects within a space and allow the generalization/specialization connections to reflect inheritance as well as connections to create composition. The students were required to obtain class, interface, or abstract class names as well as properties and methods with their respective modifiers, parameters, and return types. Typically, this was accomplished through pop-up dialog boxes. This project also reinforced the use of polymorphism as classes, abstract classes, and interfaces were best handled via polymorphism.

The final project reinforced design and implementation factors while incorporating Java code generation, refined printing, and file handling capabilities into interface development. The result was a tool that was user friendly and specifically designed for UML class diagrams. Some of the students' designs broke from the traditional Visio-like interfaces and demonstrated the benefits of application-specific user interface design. Other projects that used a more generic, Visio-like approach yielded less effective interfaces. In some cases, as the student entered textual information, the relationships were drawn or as the relationships were drawn, the textual data was updated. An example of this would be specifying inheritance through text and having the correct connections made or by dragging a mouse to create a connection and having the text indicate the extension or implementation of the relationship.

CHALLENGES

This course is challenging to teach as it is imperative to stick to the stated goals. Students who have not taken a course in HCI may be inclined to see Applied HCI as an advanced programming class and concentrate on programming, ignoring HCI factors. Students who have taken an HCI course may see Applied HCI as an HCI course and concentrate on the design without concern for the implementation.

Another challenge in a class like this is encouraging the freedom of design and creativity within the constraints of a working application. Adding to this challenge is the need for an instructor to provide tools and lectures to enable students to implement programming specific solutions without creating an atmosphere where students feel compelled to use any particular approach [1]. As an example, an application may be implemented using the Java card layout, where different pages are overlaid and the current view is selected using menu selections versus using tabbed pages to switch between

pages. The student may feel that the instructor is advocating one approach over the other when that topic is presented and will include that particular technique, just to please the instructor, regardless of the appropriateness of the application to the goals.

COURSE FORMAT

Applied HCI was offered as a three credit course, with each week divided into three distinct teaching/learning formats. Because this course did not assume familiarity with HCI as a prerequisite, the first format was instructional. Lectures were provided in basic HCI concepts and generic programming solutions to implementation concerns. The second format entailed the application of instructor-provided design strategies to a particular HCI problem, with the entire class evaluating and critiquing the design and implementation. The final format, a lab component, allowed students to pair up and work on the project and also critique each other's ideas and implementations. The students were encouraged to change pairings from week to week to obtain fresh insights into the effectiveness and usability of their projects. Each format was approximately one hour per week but the time allocation was not rigid. In the earlier weeks of the semester, the lecture and critique formats dominated the time. As the semester progressed and the students got involved in the projects, the lab component increased to approximately 40% of the weekly activities. During the lab portion, the instructor would float from pair to pair answering questions and offering suggestions.

Category	Sub category	Weight
Usability Measures		35%
	Gestalt: Proximity - Objects placed close to each other perceived as a group.[1]	5%
	Gestalt: Similarity -Similarly shaped objects will be perceived as a group.[1]	5%
	Affordance -Does form invite function (e.g. Does a button look like a button (to be pushed)?)	5%
	Visibility -A control such as a button on a product should appear to be obvious about how it is used.[2] (e.g. A button may look like a button, but may not be obvious what happens when the button is pushed.)	5%
	Feedback - A response to an action indicating that the action has been taken/is taking place.	5%
	Transfer - The ability to apply prior knowledge or experience.	5%
	Fitt's Law - Average time to complete movement is given by the equation: Time=start/stop time + device speed(log2(1 + Distance/Target Size))	5%
Predictability		10%
	Degree of match between system's model and user's actual behavior[3]	10%
Privacy		0%
	Amount of information a user has to supply to obtain value from application.[3]	0%
Cognitive Interrupt		10%
	Time taken from primary task. [3]	10%
Learnability		5%
	Ease/time to learn how to use the interface.	5%
Ease of use		10%
	Ease/difficulty to use system after learning	10%
Efficiency		10%
	Amount of time to perform a particular task.	10%
User Satisfaction		20%

	Likeability- How well do the users enjoy using the system?	10%
	Beneficial - How well does the system satisfy their needs?	10%
Total:		100%
Deductions	Points to be deducted (not added)	
Working	Program properly demonstrates capability	[50%]
Invalid submission	Zip file not submitted properly (does not contain complete project with .jar file) or zip and/or project file not properly named	[10% minimum]
Late	Zip file not submitted on time	[50/100%]
Final Grade		

[1] http://www.interaction-design.org/encyclopedia/gestalt_principles_of_form_perception.html

[2] http://openlearn.open.ac.uk/mod/resource/view.php?id=159819

[3] http://www.itl.nist.gov/iad/vvrg/newweb/ubiq/docs/2__ubi_eval_scholtz.pdf

Figure 1 Final rubric for projects

METHOD OF EVALUATION

Because of the creativity aspect of this course, students were evaluated strictly on project performance. The project specifications were fairly broad, allowing students wide latitude in their approaches. While exams could have been given on the HCI concepts and/or coding solutions, it was not considered fair—in view of the variety of strategies that students used to develop projects—to test on specific HCI concepts and coding solutions.

Prototypes are difficult to measure because of the subjectivity of the evaluator [2]. Hartfield also noted that, beyond a listing of guidelines, there is little basis for systematic assessment. While the project approach appears to be very subjective, adherence to the HCI rubrics, attached to each project, afforded more objective grading criteria. The earlier rubrics stressed HCI concepts (e.g. transfer, affordance, Fitt's Law, visibility, etc.). It became evident early on that the rubric had to be modified to also include workability, as students discovered that adherence to HCI concepts alone enabled less than fully functional projects to receive good grades. The final rubric, as shown in Figure 1 includes a deduction for non-working interfaces allows for the grading of HCI concepts and interface functionality without diluting the grading.

STUDENT FEEDBACK

Students were surveyed to determine how the course was received. Since survey participation was anonymous and voluntary, only five surveys were returned. The first section of the survey contained the following possible responses: strongly agree, agree, disagree, strongly disagree, and not applicable.

Based on the limited feedback, the students either strongly disagreed (40%) or disagreed (60%) that the course repeated material from earlier classes. This was less important with respect to HCI theory as only a few students had taken the theoretical HCI course, while all had taken at least two courses using Java.

With respect to focus: the emphasis on HCI theory was mixed with 40% agree, 20% disagree and 40% strongly disagree: and the emphasis on programming was 60% strongly

agree and 40% agree. Perhaps the results of the programming focus reflected the need for a working prototype. Two students agreed that there was a balance between HCI theory and programming.

The survey asked if the grading rubric reinforced HCI concepts. Here, again the results were mixed. Twenty percent (20%) strongly agreed, 40% agreed, and 40% disagreed while no one strongly disagreed that the grading rubric reinforced HCI theory. Perhaps the disagreement was based on the addition of "working" to the rubric which included a 50% deduction.

With respect to the question "Overall the class was beneficial" 60% strongly agreed and 40% agreed. This was a positive, yet surprising result, considering one student disagreed when responding to the question "I would recommend this course to a friend".

The survey also contained a section relating to the level of the course: 80% responded "Senior/Graduate" and 20% responded "Junior". No responses indicated "Sophomore" or "should not be offered". The fact that no one responded "should not be offered" was taken as another indication of the course's merit.

Finally, the survey contained a section about the approximate number of hours spent on each of the projects with possible responses of 1-2, 3-4, 5-6, 7-8, and 8+. Table 1 shows the spread of responses over the first five projects. The sixth project was not included as it was not completed at the time of the survey.

Students mentioned spending as many as forty hours on a project because the projects were fun and interesting. One student continued to work on implementing the UML Class Diagram project long after the semester ended.

Table 1. Approximate hours spent on projects

Approximate hours spent on project	1-2 hours	3-4 hours	5-6 hours	7-8 hours	8+ hours
1			1	2	2
2				4	1
3			1		4
4			2	2	1
5				1	4

In summary, the students found the course to be new, interesting and rewarding.

CONCLUSION AND FUTURE WORK

Based on formal and anecdotal student feedback, this course should be made a permanent three credit course in the curriculum, being offered either as a major elective at the senior/graduate level or as a required course in an IS/IT program. The course reinforces object-oriented material, introduces critical HCI theoretical material, and demonstrates to students the importance of usability testing. The course could be improved by requiring a theoretical HCI course as a prerequisite to provide a background in proper HCI design methods. The lectures would have to be altered slightly to reinforce, rather than introduce, the HCI theory and to allow for a stronger connection between theory and application.

Future offerings of the course will have to create a more proper balance of students' time as some students became too involved in the projects to the detriment of other classes. Perhaps smaller scale projects would reduce the students' perceived workload.

REFERENCES

[1] BROWN, C. AND PASTEL, R. 2009. Combining distinct graduate and undergraduate HCI courses: an experiential and interactive approach. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, Chattanooga, TN, USA, Anonymous ACM, New York, NY, USA, 392-396.

[2] HARTFIELD, B., WINOGRAD, T. AND BENNETT, J. 1992. Learning HCI design: mentoring project groups in a course on human-computer interaction. *SIGCSE Bull*. 24, 246-251. http://doi.acm.org/10.1145/135250.134559.

[3] KONG, J., ZHANG, K. AND ZENG, X. 2006. Spatial graph grammars for graphical user interfaces. *ACM Trans.Comput.-Hum.Interact*. 13, 268-307. http://doi.acm.org/10.1145/1165734.1165739.

[4] KOPPELMAN, H. AND VAN DIJK, B. 2006. Creating a realistic context for team projects in HCI. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, Bologna, Italy, Anonymous ACM, New York, NY, USA, 58-62.

[5] PASTEL, R. 2005. Integrating science and research in a HCI design course. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, St. Louis, Missouri, USA, Anonymous ACM, New York, NY, USA, 31-35.

[6] WRIGHT, T., NOBLE, J. AND MARSHALL, S. 2005. Using a system of tutorials and groups to increase feedback and teach user interface design. In *ACE '05: Proceedings of the 7th Australasian conference on Computing education*, Newcastle, New South Wales, Australia, Anonymous Australian Computer Society, Inc, Darlinghurst, Australia, Australia, 187-192.

THE POTENTIAL BENEFITS OF MULTI_MODAL SOCIAL INTERACTION ON THE WEB FOR SENIOR USERS*

Anjeli Singh
Computer Science & Software Engineering
3101 Shelby Center
Auburn University
Auburn, AL 36849-5347

Andrea Johnson
Human-Centered Computing Division
School of Computing
100 McAdams Hall
Clemson University
Clemson, SC 29634-0974
andrea5@clemson.edu

Hanan Alnizami
Human-Centered Computing Division
School of Computing
100 McAdams Hall
Clemson University
Clemson, SC 29634-0974
hanana@clemson.edu

Juan E. Gilbert
Human-Centered Computing Division
School of Computing
100 McAdams Hall
Clemson University
Clemson, SC 29634-0974
juan@clemson.edu

ABSTRACT

The Internet has become a key factor in today's society for improving lifestyles, from entertainment to obtaining every day life necessities. Despite this, most seniors hesitate to use the Internet. As a result, there is a lower growth rate in Internet usage among seniors compared to those of younger populations. In this research, a web interface using a virtual conversational agent has been designed to address the problems of existing web interfaces; especially e-commerce sites. Data from a comparative study between an avatar design and a traditional design for a web-based retail store has been collected and analyzed. The results support the avatar design over a traditional web design;

---

however, in some cases, the evidence collected showed no statistical significance.

1. INTRODUCTION

Senior citizens, ages 65 and older, form a total of 12.9% of the total US population according to 2009 US Census Bureau records [1]. In comparison with 90% of adults ages 19-29 who use the Internet, only 35% of the total senior citizens use the Internet. Unaccommodating hardware and software interface designs that disregards seniors' needs have been a primary cause for this age-based digital divide [2].

Progressing in age, senior citizens find it laborious to navigate the web as a result of age–related changes to their sensorimotor and cognitive capacities. Specific causes for cognitive decline have been associated with deficient attention, working memory, information processing speed, reaction time, response inhibition strength, encoding rates, or quality and rate of inference formation [3].

In a time where technology has become an important component of one's every day life, the necessity constitutes that seniors engage in web-based consumer environment transactions to attend to their daily needs. With the rising numbers of seniors' population, it was important to address this issue with a novel solution that suits seniors' needs while keeping them up to speed.

Unfortunately, current web interfaces are not designed with regard to seniors' needs. This observation has provided motivation for research in means to gain a better understanding of seniors' needs as a way to bridge the gap between age and technology. Key questions investigated in this paper are: (1) Is it uncomfortable for seniors to use the Internet, and if so what is the cause? (2) If uncomfortable, will conversational agents be helpful in enhancing senior users' online experience? (3) How does an avatar web interface design differ from traditional web interfaces design?

2. LITERATURE REVIEW

2.1 Effects of Aging on Web Usage

Aging has an effect on seniors' cognitive abilities as well as their physical condition as their cognitive processes become slower. Some of the issues that arise with senior citizens Internet use are:

1- Response Time: Research proves that age results in slower response time as indicated in the adult learning theory and practice. It also entails that additional processing is needed for a senior adult to learn and process new information.[4]

2- Learnability: Learning ability is crucial to Internet adoption. Aging causes learning concerns such as memory loss and emotional barriers [4].

3- Visibility: As a part of aging, declining in one's vision implies suffering from a decline in focusing power, sharpness, and identifying colors [5].

4- Hand-eye coordination: One of the biggest hurdles that seniors face when learning how to use a computers is coordinating between the use of the keyboard and mouse

while focusing on a screen visual as they perform a task. Controlling a mouse requires good vision and good motor controls.

5- Mobility: Using the computer requires that a user sits in one position focusing on a screen visual while performing a computer task, controlling devices such as a keyboard and/or a mouse. This position could become tiresome and constraining to a senior user.

2.2 Agent based Learning

It has been proven that people display a natural propensity when interacting with machines as if they were interacting with people [6]. In various studies, participants have shown social rules of human computer interaction ranging from frustration, politeness to treating computers as people with personalities [7]. Reeves and Nass developed the Computers As A Social Agent (CASA) paradigm that aimed to study the rules between human-human interaction and human-computer interaction. They demonstrated computers as a social actor [8].

Humans build relationships based on face-to-face communication through verbal and non-verbal conversational behavior [9]. To achieve interaction on the web, a natural conversational interface is required. This can be obtained by the use of embodied conversational agents (ECA). ECAs are computational artifacts equipped with verbal and non-verbal communication skills, designed to offer longterm, social-emotional presence. They can take various embodiments including nonhumanoid, physical, or nonphysical forms [9]. Users' experiences on the web could be improved with the help of an ECA as studies have shown that immediate positive feedback is key in learning succession [8]. Such agents could provide instructional assistance that depends on user's input. In addition, research has shown increased cognitive and emotional trust, and a sense of control over the computer interaction in websites when using ECAs [10].

3. EXPERIMENT

It is proposed from the previous section that the use of an ECA could improve senior citizens' experience by providing instructional assistance to perform a certain online task, via direct contact between the user and the avatar. A mock up of an e-commerce apparels website called Sophisticated Casuals was created housing Gina, an animated virtual customer representative, to offer assistance to senior users when needed.

Establishing a trust worthy social presence is of great importance when using virtual agents. To establish this presence, Gina was equipped with social responses that demonstrate politeness, clarity, and regret. Her responses depicted those of a customer service representative in shopping malls, making this interaction realistically familiar; she demonstrated non-verbal facial expression (e.g nodding, blinking and moving lips). She offered indirect training on web maneuvering, explaining the use of navigational buttons and image zooming.

A laboratory-controlled experiment was conducted to study the comparison between the virtual agent and the control interface. See Figures 3.1 and 3.2. The procedure for the

experiment went as follows: Gina welcomes the user and asks if she could be of assistance. The user types the kind of help they need. Gina then provides step-by-step detailed instructions via speech and text.

Figure 3.1: Control Interface

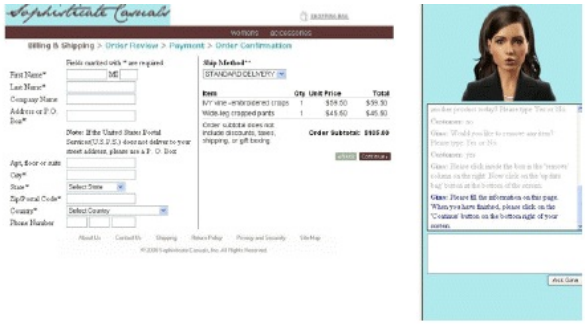


Figure 3.2: Gina providing instructions to user

3.1 Demographics

The focus of the study was to test the comparative impact of the avatar vs. the control interface with senior users. 61 senior members of the Osher Lifelong Learning Institute (OLLI), a membership program for learning in retirement, with various demographics were recruited for this study. Demographics are shown in Table 3.1.

Age	50-89 (Min=50, Max=89, Mean = 68.34, Std Dev =8.78)
Gender	25 female, 34 male (2 participant did not specify a gender)
Computer Literacy	Minimum 1 year, maximum 10 years
Internet Literacy	Minimum 6 months, maximum 10 years
Ethnicity	Non Hispanic White and Non Hispanic Black
Income	Minimum $20,000 a year, Maximum $250,000 a year

Table 3.1: Study demographics

3.2 Experiment Design and Procedure

A between-subject design was used in this study with two treatments, the control interface and the avatar interface. Each participant was randomly assigned to one of the two designs to perform the task. After consenting to participate in the study, participants were given task description details. Tasks were divided into two parts, searching for a given product and purchasing the product. Pictures of items were provided to eliminate the need for conversations with the experimenter. All additional information required to

complete the purchase, such as credit card number, address and other related information, was provided to the user. Textual descriptions that could have possibly served as search keywords, such as black jacket or white pants, were omitted from the task description.

Desktop computers running windows XP were used to access the websites using Internet Explorer. The study lasted about 20 minutes. Upon task completion, participants completed a post-survey questionnaire that collected demographics information and data on user experience.

Since a primary goal of the study was to examine the effectiveness of an online agent as a virtual assistant for website control and navigation, a number of usability attributes were measured. Dependent variables measured were: difficulty in getting started, learnability, discovering new features, feedback, successful task completion, recoverability, user satisfaction, and flexibility. Participants were asked to rank each dependent variable on a 5 point Likert scale; where 1-strongly disagree 2-disagree 3-neutral 4-agree 5-strongly agree.

4. RESULTS AND DISCUSSION

Data collected from the questionnaires was treated as ordinal; to test for statistical differences between the conditions, the researchers performed an independent two sample t-test, for each measure in the study. There were 61 participants with an average age of 68.34 years participating in this study. Of those participants 57.6% were female and 42.4% were male. Between the control sample mean and the experimental sample mean, the overall mean values for the avatar interface were higher than the control interface.

Mean values of the key points identified in this study are addressed in the survey listed in Table 3.2 below. The results indicate that senior participants were more capable of successfully completing their task with the avatar design in comparison with the control design. Senior participants reported having a satisfying experience and felt that the avatar website was more learnable in comparison to the control website. They believed their understanding of online shopping improved after using the avatar website, which the researchers found promising.

Senior participants also felt that they could easily recover from errors. At a 5% level of significance there was sufficient evidence to conclude that if they made a mistake, it was easy for them to correct it (i.e. Recoverability t-ratio=2.18 p=0.0336). The feedback provided by the avatar was found to be understandable and assisted in the recovery process. Moreover, seniors found the avatar design was helpful in getting them started with their tasks in comparison to the control design. After observing a small difference in the means for getting started, we detected a higher standard deviation in the control group (Std $Dev_{control}$ =1.24, Std Dev_{avatar}=1.09) which could indicate a difference of opinions about getting started. It was also observed that there was a statistical significance in the seniors' ability to discover new features with ease (i.e. Discovery of new features t-ratio=-2.51 p=.015) and the results suggested that the avatar prompts served as a factor in this increase. One possible interpretation of the increase could be attributed to the fact that the avatar offered navigational help to participants according to their needs.

Further analysis indicated a significant difference in evaluating the design according to dullness versus stimulating (t-ratio=2.10 p=.040) in user experience. The mean value for the avatar design was higher implying that participants found the avatar site more stimulating than the control site. The avatar design was also found to be more flexible.

Survey Key points	**Control Design**	**Avatar Design**
Able to successfully complete the task	4.31	4.61
Had satisfying experiences	4.14	4.24
Felt the avatar website was more learnable	4.13	4.32
Improved understanding of online shopping	3.40	3.46
+Found they could easily recover from errors	3.90	4.40
Feedback provided by the avatar was understandable	4.03	4.38
*Seniors found getting started to be hard design	1.97	1.77
+*It was difficult to discover the new features	2.03	1.83
Design flexibility	3.88	4.14

Table 4.1: questionnaire key points * denotes negative question, making the lower mean the better result, + denotes statistically significant

5. CONCLUSION

This research presents findings of a study aimed to identify the effectiveness of utilizing a virtual conversational agent to enhance seniors' experiences online. The avatar interface housing a conversational agent did show benefits over the traditional control interface. By providing emphatic social presence serving instructional agents, users were able to complete shopping tasks effectively. Findings suggest that well designed instructional agents help to address web interface related issues among novice seniors. However, limitations to the virtual agents design proposed have been discovered and will be improved in future work.

6. ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under Grant Number IIS-0955763. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] US Census Bureau, 2009. http://quickfacts.census.gov/qfd/states/00000.html

[2] Bucar, A., Kwon, S., Computer Hardware and Software Interfaces: Why the Elderly Are Under-represented as Computer Users, *Cyberpsychol. Behav*, 2, 535-543, 1999.

[3] Fairweather, P. How Older and Younger Adults Differ in Their Approach to Problem Solving on a Complex Website. In *Proceedings of ASSETS2008*, ACM Press, 67-72, 2008.

[4] Timmermann, S., The Role of Information Technology in Older Adult Learning, *New Directions for Adult and Continuing Education*, n77 p61-71, 1998.

[5] Vision, Aging, and Age-related Diseases, 2004, http://www.springboard4health.com/notebook/health_vision_disease.html

[6] Reeves, B., Nass, C., *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places* (CSLI Lecture Notes). Center for the Study of Language and Inf, 2003.

[7] Fogg, B.J, Nass, C., Silicon sycophants: the effects of computers that flatter. *Int. J. Human-Computer Studies* 46, 551D561, 1997.

[8] Bickmore, T., Picard, R., Establishing and Maintaining Long-Term Human-Computer Relationships, *ACM Transactions on Computer-Human Interaction,* Vol. 12, No. 2, Pages 293–327, 2005.

[9] Qiu, L., Benbasat, I., Online Consumer Trust and Live Help Interfaces: The effects of text-to-speech voice and three-dimensional avatars. *International Journal of Human-Computer Interaction,* 19(1), 75-94, 2005.

[10] Wang, L. C., Baker, J., Wagner, J. A., Wakefield, K. 2007. Can A Retail Website Be Social? *Journal of Marketing,* 71, 143-157

CORRELATION BETWEEN CLASS ATTENDANCE AND GRADE*

Jenq-Foung "JF" Yao, PhD
Georgia College & State University
478 445-5483
jf.yao@gcsu.edu

Tsu-Ming Chiang, PhD
Georgia College & State University
478-445-0863
tm.chiang@gcsu.edu

ABSTRACT

This study investigates the relationship between students' class attendance and their overall grades in multiple computer science classes. The correlation between attendance of the first class and the overall grade is also examined. The Pearson Correlation analysis was performed to analyze the data. The results show that a student's overall grade highly correlates with his or her class attendance; it is also more imperative for CS majors than for non-CS major to attend classes. Furthermore, students who missed the first class have lower grade averages than students who did not miss the first class.

INTRODUCTION

This paper systematically examines relationships between class attendance patterns and overall grade in various computer science courses. Although it is logical to assume that a student who misses more classes will receive a worse grade, comprehensive research is needed to verify such an assumption. In addition, this study examines students' overall grades in relation to their attendance of the first day of the classes, as some students tend to ignore attending the first day of classes. The statistical analysis verifies the assumption and shows interesting results.

METHODOLOGY

Participants

Students' attendance records for the past five years along with students' grades were used in this study. The computer science courses included in the study are Computer Literacy, C++ Programming, Data Structures, Discrete Math, Operating Systems, and Linux Programming.

Procedure

Students were asked to sign in at the beginning of each class and were informed that there was no penalty for missing classes. The total number of missed classes and percentage of attendance for students were calculated at the end of each semester. For students who missed the first day of class, we marked their *missed-1st* variable as 1; otherwise, we marked it as 0. Students who dropped the class were excluded from the data.

Statistical Procedure

One-tailed[1] Pearson Correlation analysis using the statistical package SPSS was performed to analyze the data. Students' overall grades and attendance rate/percentage from all classes, including a computer literacy class designed for non-computer science majors, were analyzed first. Then, students were divided into two groups: CS majors and non-CS majors. The analysis was then performed separately for majors vs. non-majors. Finally, students' attendance on the first day in relation to their overall grades was examined.

RESULTS AND DISCUSSION

The statistics analysis results are listed in Table 1. The second column is the sample size; third column, missed the first class (yes or no); fourth column, average of attendance rate; fifth column, grade average; sixth column, standard deviation of the grade average; seventh column, correlation of grade and attendance (r value); eighth column, p value (1-tailed); ninth column, the significant level. The rows are grouped into three sections: non-computer science majors, computer science majors, and all students. Each section has three rows: first row for students who did not miss the first day of the class, second row for the otherwise, and the third row for all students in the same category (either CS, non-CS, or all).

First, an interesting observation: taking the average of both the attendance rate and the grade has revealed that the two values are very similar (column 4 vs. column 5 in Table 1). These numbers indicated that students' grades are closely related to their attendance. The percentage of class attendance almost equaled the grades students ultimately received, showing that more class attendance means higher grades. The other

[1] Two-tailed analysis was also performed and yielded similar results.

observation is that CS majors attended class slightly more than non-CS majors did (80.59% vs. 77.64%).

The correlation between grades and attendance rate is very high (r=0.664, p=0.000, N=472) for all students. Attendance was linked strongly to good grades, as suggested by the data. We further separated students into CS majors and non-CS majors, and the analysis results show that CS majors (r=0.646, p=0.000, N=197) and non-CS majors (r=0.695, p=0.000, N=275) are about the same.

In addition, the data show that students who miss the first day of the class have a significantly lower grade average (61.891 out of 100) than those who do not (80.214). The grade point average difference between these two groups is 18.323, which indicates that missing the first day of class is significantly related to the students' performance in said class. Furthermore, the correlation (r=0.854, p=0.000, N=24) between the grades and attendance rate of the students who missed the first day of the class is higher than the other groups who did not miss the first day of the class (r = 0.616, p = 0.000, N = 448). The correlation (r=0.854) of the students who missed the first day of the class is also much higher than that of the students overall (r=0.664). Once again, the results suggest the importance of the first day of the class.

Majors	Sample size (N)	Missed 1st class	Attend-ance Rate	Grade Average	Standard deviation	Correlation of Grade and Attendance (r value)	P value (1-tailed)	Signifi cant level
Non Computer Science Majors	260	No	78.86%	81.367	16.6291	0.661	0.000	0.01
	15	Yes	56.42%	65.311	29.4007	0.821	0.000	0.01
	275	**n/a**	**77.64%**	**80.491**	**17.8577**	**0.695**	**0.000**	**0.01**
Computer Science Majors	188	No	81.89%	78.620	17.1679	0.579	0.000	0.01
	9	Yes	53.62%	56.191	30.6608	0.935	0.000	0.01
	197	**n/a**	**80.59%**	**77.595**	**18.4829**	**0.646**	**0.000**	**0.01**
All students	448	No	80.13%	80.214	16.8927	0.616	0.000	0.01
	24	Yes	55.33%	61.891	29.5548	0.854	0.000	0.01
	472	**n/a**	**78.87%**	**79.282**	**18.1581**	**0.664**	**0.000**	**0.01**

Table 1: Statistics Summary of All students

Overall, Students' grades were lower in the group that missed the first day of class (column 5 of Table 1). CS majors also have lower grades than non-CS (77.595 for CS, 80.491 for non-CS), despite the fact that they had a higher attendance rate than non-CS majors (80.59% vs. 77.64%). In addition, the difference in the grade average between those who attended the first day of class and otherwise is (78.620 - 56.191 = 22.429) for the CS students, which is higher than that of the non-CS students (81.367 - 65.311 =

16.056). The data demonstrate that attending CS classes is more important than non-CS classes to obtain a better grade. Because CS courses are much more difficult than non-CS courses, attending classes helps. Also, attendance on the first day of class is a better grade indicator for CS majors.

The correlation (r=0.935, p=0.000, N=9) between the grades and attendance rate for CS students who missed the first day of the class is stronger than those who attended the first day of the class (r = 0.579, p = 0.000, N = 188). These findings reaffirm that, for CS students who were absent from the first day of the class, attendance is highly correlated with grades. It suggests that whether or not students miss the first day is a good indicator of how students will perform in the class.

Table 2 lists the statistics that have been broken down into individual courses. The significant set level varies by the sample size; the significant level was set at 0.01 for the larger sample size, otherwise at 0.05. For the sample size 0 or 1, only the grade average can be calculated. For data with the small sample size, the statistics results may be skewed.

Course	Sample size (N)	Missed 1st class	Grade Average	Standard deviation	Correlation of Grade and Attendance (r value)	P value (1-tailed)	Signifi-cant level
Computer Literacy	260	No	81.367	16.6291	0.661	0.000	0.01
	15	Yes	65.311	29.4007	0.821	0.000	0.01
C++ Programming	63	No	76.738	20.8201	0.664	0.000	0.01
	1	Yes	11.250	n/a	n/a	n/a	n/a
Data Structures	32	No	75.272	20.1828	0.509	0.001	0.01
	3	Yes	70.683	30.1996	0.991	0.043	0.05
Operating Systems	25	No	78.360	13.3522	0.366	0.036	0.05
	1	Yes	24.500	n/a	n/a	n/a	n/a
Linux Programming	32	No	84.904	12.5527	0.550	0.001	0.01
	0	Yes	n/a	n/a	n/a	n/a	n/a
Discrete Math	36	No	79.485	11.4327	0.476	0.002	0.01
	4	Yes	64.480	24.2742	0.940	0.03	0.05

Table 2: Descriptive Statistics broken down into individual courses

Note that in the higher-level courses, such as Operating Systems, and the more difficult courses like Discrete Math, the correlation between students who missed the first day of the class and their grades is less significant in comparison to easier courses like C++ programming. One possible explanation is that most of the students showed up on the first day of upper level classes because they anticipated the material would be more difficult. The other reason may due to the fact that the harder material makes it more difficult for some students to get a good grade despite their attendance on the first day of the class. Also note that there was an extremely high correlation between grades and attendance (r=0.949, p=0.03, N=4) for students who missed the first day of class in Discrete Math. This result continues to suggest that missing the first day of class is a significant factor in their grades. It also implies that showing up for the first day of class may or may not help their grades, but not showing up on the first day of the class can definitely predict their poor performance in the class.

Possible reasons contributing to the significance of the first day of classes are threefold. First, an instructor usually states her/his expectation and class requirements at the first class. This sets the stage for the rest of the semester. Students who miss the information tend to fail to connect with the professor's expectations from the beginning. Second, students who ignore the first day of classes are more likely to be unmotivated, and thus perform worse than their counterparts. Third, students who are absent from the first day of classes may have health issues or family crises; they are subsequently more likely to be distracted and cannot perform as well as others.

LITERATURE REVIEW

There are a few studies on this topic in other fields. LeBranc [6] and Broucek [2] demonstrated a positive relationship between students' class attendance and grades in various business classes. Millis et al [7] showed a significant association of classroom participation with examination performance in a first year medical school class. Aldosary [1] reported that the correlation between final scores and homework performance is stronger than the correlation between final scores and the attendance among Environmental Design students. Davenport [4] showed a statistically significant positive relationship between attendance and grades in three business law classes. Clump et al [3] examined the effect of attending class in a general psychology course. Their findings support the notion that attending class is very influential on a student's grade. In [5] the authors conducted a similar research in Physiology Education classes. The findings indicate that regular attendance was helpful but was not a decisive factor in learning human physiology. Urban-Lurain and Weinshank [8] investigated the relationship between attendance and outcomes for non-computer science students in large introductory computer science labs, in which no lectures were given. The research concluded that the strongest predictor of student performance is overall class attendance, comparing the specific days attended with year-in-school or prior computing experiences.

CONCLUSIONS

Our analysis of the correlation between attendance and grades reveals that computer science classes have a higher correlation between overall grade and attendance than previous research findings from different disciplines. It also showed a strong correlation between the overall grade and absence from the first day of the class.

Students who have poor attendance, regardless of being CS or non-CS majors, tend to have worse grades. Students who missed the first class had a grade average that is 18.323 points lower than the students who were present at the first class. The difference is greater for CS majors (22.429 points) than non-CS majors (16.056 points). These findings suggest that attending classes is crucial for the overall learning outcome in grades. The first day of class attendance is a strong indicator of high likelihood of failure. Furthermore, it is more imperative for CS majors than non-CS majors to attend classes. Therefore, an attendance policy that encourages students' attendance may ultimately be beneficial to students' learning outcomes. In addition, faculty members may classify the students who are absent from the first day of classes as an "at-risk" group to further

communicate the class expectations. Further research in examining students' attendance from multiple disciplines will help us to understand whether the results are similar across disciplines.

REFERENCES

[1] Aldosary, A. S. (1995). The Correlation between Final Grade Score, Attendance and Homework in the Performance of CED Students. *European Journal of Engineering Education* , 20 (4), 481 - 486.

[2] Broucek, W. G., & Bass, W. (2008). Attendance Feedback In An Academic Setting: Preliminary Results. *College Teaching Methods & Styles Journal* , 4 (1).

[3] Clump, M. A., Bauer, H., & Alex, W. (2003). To Attend or Not to Attend: Is That a Good Question? *Journal of Instructional Psychology* , 30.

[4] Davenport, W. S. (1990). *A Study of the Relationship between Attendance and Grades of Three Business Law Classes at Broome Community College*. Nova University.

[5] Hammen, C. S., & Kelland, J. L. (1994). Attendance and Grades In A Human Physiology Course. *Advances in Physiology Education* , 12 (1), 105-108.

[6] LeBranc, P. (2005). The Relationship between attendance and grades in the College Classroon. *The 17th Annual meeting of the International Academy Business Diciplines*. Pittsburgh, PA.

[7] Millis, R. M., Dyson, S., & Cannon, D. (2009). Association of classroom participation. *Advance Physiology Education*, 33, 139-143.

[8] Urban-Lurain, M., & Weinshank, D. J. (2000). Attendance and outcomes in a large, collaborative learning, performance assessment course. *the Annual Meeting of the American Educational Research Association*.

TEACHING EXPERIENCES WITH ALICE FOR HIGH SCHOOL STUDENTS*

Brenda Parker
Computer Science Department
Middle Tennessee State University
Murfreesboro Tennessee 37132
(615) 898-2389
csbrenda@mtsu.edu

ABSTRACT

As participants in the Partners for Innovation in Information Technology (PIIT) National Science Foundation (NSF) grant, I was privileged to work with 15 high school students during a week-long Alice workshop. Alice is free software provided by Carnegie Mellon [5] and provides a visual programming environment to help introduce beginning programming concepts in a fun and exciting way. This paper is an experiential paper denoting efforts in the organization and delivery of the Alice workshop. Organizational efforts in preparing for the workshop, experiences in the delivery of the workshop, the effectiveness of the workshop and suggestions for delivering an Alice workshop will be shown.

INTRODUCTION

In an effort to improve the current downward enrollment in computer science at our university, we determined to offer several workshops for high school students in this area. As a result of a recent PIIT NSF grant, several universities and community colleges in this area received funding to conduct a weeklong summer workshop for high school students. The workshops focused on three areas of interest: robotics, multi-media and Alice. The workshops were publicized via local high school principals and through selected mathematics and science teachers. A web site was developed allowing students to register and choose their desired workshop. Students were asked to indicate their workshop

preference (Alice, Robotics or Multimedia) and state why that preference was chosen. Each workshop had a workshop leader and two college student assistants (SA). The workshop leader's roll was to present material and plan student activities. The SAs were asked to aid students with problems and help plan games and paper activities for the students.

This paper presents a brief summary of the Alice workshop and activities. Alice has been used in numerous summer camps as indicated in [1,2,4] and presents a visual programming environment which helps to introduce programming to students in a fun and enjoyable way. It has been shown to improve student understanding and retention of beginning programming concepts.[2,3]

ORGANIZATIONAL EFFORTS

Presentations and activities were required for each day of the workshop. The activities for each day followed the same guideline: show short, slide presentations introducing a programming or Alice concept and follow each presentation with hands-on student activities to allow students to practice the presented topic. Organization efforts were divided up into two major components: a) determine topics for each day and create the slide content based on the indicated topics and b) develop a list of student activities for each day.

Workshop Topics

After the format for each day was outlined with the end goal of helping students learn the basics of programming and computer science, major topics were listed which would help achieve this goal. It was discovered that most students who requested the Alice workshop were mainly interested in creating video games. Therefore, a list of objectives was created for each day of the workshop so that students would have the knowledge needed to create and demonstrate their own video game by the end of the week. Table 1 below indicates a brief description of the objectives for each day of the workshop as well as a brief description of the student assignment to help achieve each day's objective.

Day	Objective	Student Assignment
1	• Explain the Alice IDE • Explain how to create/play/save a simple Alice animation • Explain how/why to set preferences • Explain the purpose and use of selected control structures	1. Set preferences, create a world, add objects and position the objects properly 2. Add methods to your objects 3. Modify video by using a control structures (do together or loop) 4. Create a new video that tells a story AND contains control structures

2	• Explain the Software Development Cycle • Create and call world custom methods • Use camera control arrows • Use control structures	1. Create a sample animation that has two world custom method called dance and bow. The video should use the methods that would cause 3 objects to bow, perform a dance and then bow at the end of the animation 2. Create animations that would require user input. Sample projects were distributed.
3	• Simple user input • Control Structures • Event programming	Sample projects were distributed which required event programming. Students were allowed to choose their desired project and their team members. Each team member was given specific tasks to complete the project.
4	• Simple game creation • Billboards • Variables • Manipulate the camera	Student teams designed their game, designed and inserted billboards into their game and worked on game project
5	Work on game and demonstrate game to all participants	Students completed games, and demonstrated game to all participants as well as to other workshop participants. Students played Alice Jeopardy game

Table 1. Daily Objectives

Student Activities

As stated earlier, student activities were divided up into two main components: 1) creating short Alice files to incorporate knowledge learned through the slide presentations and 2) participating in "fun" activities to practice knowledge learned during the presentations. Table 2 below indicates a listing of some of the "fun" activities.

Activity	**Description**
Around the World	A student (S1) stands behind another student (S2). The instructor asks a question. If S1 answers the question first, S1 will move to next student. If S2 answers the question first, S1 must sit down and S2 moves to next student. This game can continue for a "long" time!

Crossword Puzzle	Simple crossword puzzle handouts are given to each student. An award is given to those who can complete the puzzle. Easy crossword puzzle software is freely available or at a minimal cost. Crossword puzzles provide a great way to help students learn basic computer science vocabulary.
Alice Jeopardy	An Alice Jeopardy game can easily be developed using PowerPoint slides. Our group was divided up into two teams. A student from each group is asked to come to the front of the room and one of the students chooses an amount and a topic. All students must eventually come to the front of the room to participate. The score is kept for each team. See Figure 1 for a list of Jeopardy topics.
Quiz	Short quiz handouts are given to the students. Awards are given to those who answer correctly.

Table 2. "Fun" Activities

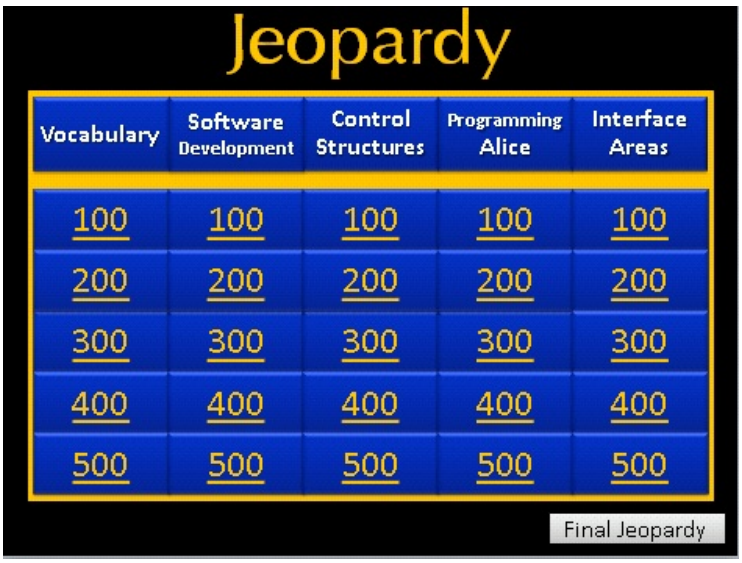


Figure 1. Alice Jeopardy Topics

WORKSHOP DELIVERY

The delivery of the workshop was carefully planned with all PIIT leaders in agreement with regard to the workshop timeline. The schedule follows in Table 3.

	Day 1	Day 2	Day 3	Day 4	Day 5
9:00 AM	Introduction & Survey	(Games, CS Unplugged)			Lab Work
9:30 AM		Lab Work			
10:00 AM					
10:15 AM	Lab Work				
11:30 AM	Lunch and Computer Game				
12:30 AM	Lab Work	Campus Tours			
2:00 PM	Break (snack)				
2:30 PM	Lab Work				
4:00 PM	Departure				

Table 3. Daily Schedule

Presentations were given during each Lab Work assignment period. These presentations were very short and were divided up into two components: First, introduce the Alice concept to be emphasized and second give students the opportunity to practice the presented concept. Sample practice sessions are shown below.

Day	Practice Session
1	1. Create a folder on flash drive, open Alice and set preferences 2. Create a new world, add objects, learn to position objects, rename objects, save world 3. Add methods to objects, play the video, capture a picture 4. Create a video that tells a story (Examples were given from the Alice text) 5. Add Control Structures to video and demonstrate video to class and/or instructors
2	1. Use Alice and practice the software Development Cycle 2. Open Alice and create an Alice video called bunnyDance.a2w 3. Create new world method dance 4. Call the new world method 5. Add documentation to Alice file 6. Choose assignment from text or create own story - must have primitive methods as well as new world methods 7. Demonstrate video to class
3	1. Create video that requires user input 2. Use the while looping structure 3. Create a simple video that uses event programming 4. Use Billboard to display instructions for user in event programming

4	Design and create video game. Sample games may include: 1. Shark Attack - maneuver an object (O1) around other objects to prevent a shark getting to close to the O1 2. Whack-A-Mole - world should have several objects (moles) that appear at random. Player will gain points by hitting moles during a given time period 3. Frogger - create a world in which an object (O1) is trying to cross an obstacle course safely to the other side. 4. Space Travel - create a world in which an object (O1) is trying to reach a destination in space and must avoid flying debris in the process 5. Alice in Wonderland - an Alice object attempts to follow the rabbit through a maze
5	Instructors and student assistants created a long list of questions related to each workshop (Alice, Robotics and Multimedia). The questions were pasted on the walls

Table 4. Practice Sessions

Workshop Effectiveness

Basic pre and post surveys were conducted at the beginning and at the end of the workshop. Figure 2 indicates the results of some of the viewpoints expressed by the students.

The highlight of the workshop was the development of the video games. We found that many students wanted to work on their project at home. Also, we noted that quite often students preferred to continue working on their game instead of taking the scheduled "snack" break.

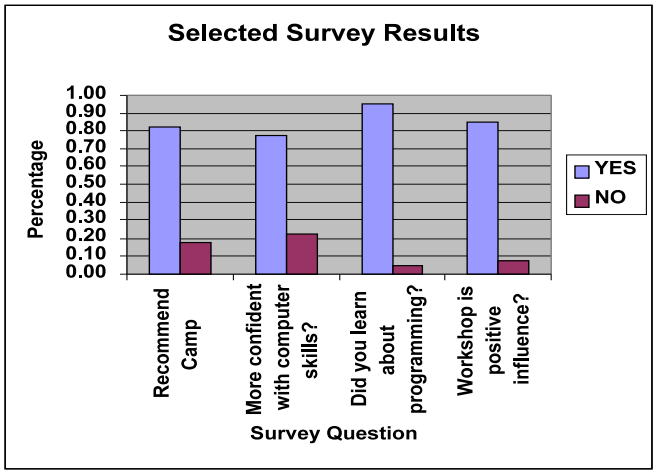


Figure 2. Survey Results

Workshop Suggestions

Various lessons were learned from the workshop.

1) Plan well. The most time-consuming aspect of the workshop was the planning that preceded the workshop.
2) Keep the students busy. Depending on the makeup of the students, many students are motivated to learn on their own. Therefore, it is important to have additional suggestions and ideas available for those who wish to work ahead.
3) Students were highly motivated to create their own games. It is important to help students learn the Alice basics and quickly provide opportunities for students to begin designing and creating their own games.
4) Students seemed to really enjoy the games - Jeopardy, Around the World and Crossword Puzzles were popular.
5) Prizes were well received. Awarding excellence and participation was a great motivational tool.
6) Candy was a very popular prize!
7) Students enjoyed showing off their final projects to other Alice participants and to the other workshop members.
8) Allowing the students to work in teams on the final project was very well received. This allowed each student to provide their own expertise to the project.
9) Handouts are important. All students were given a notebook containing different handouts related to each day's topic. It was hoped that the notebook would encourage the students to continue working on Alice at home and be used to introduce Alice to friends of the workshop participants.

CONCLUSIONS

The objectives of the Alice workshop were accomplished. Based on the final projects, students learned basic computer science concepts and had fun in the process. We already have students signed up for the 2011 summer camp and hope to use the lessons learned during this camp to aid future students in learning the joys of programming in Alice.

REFERENCES

[1] Adams, J., Alice middle schoolers and the imaginary world's camps. *Thirty-eight SIGCSE Technical Symposium on Computer Science Education*, 307-311, 2007.

[2] Bruckman,A., Biggers,M., Ericson,B.,McKlin,T.,Dimond,J., DiSalvo,J., Hewner, M., Ni,L., and Yardi,S., Georgia computes!: Improving the computing education

pipeline, *Fortieth SIGCSE Technical Symposium on Computer Science Education,* 86-90, 2009.

[3] Cooper, S., Dann, W., Pausch,R., 2000.Alice: A 3-D Tool for Introductory Programming Concepts. *Journal of Computing Science in Colleges*.15, n.5. 107-116.

[4] Peluso, E., Mauch,E., Incorporating Alice into a summer math and science outreach program, *Alice Symposium*, 1-4, June 2009.

[5] http://www.alice.org

TEACHING PEER-TO-PEER PROGRAMMING METHODOLOGIES IN INTRODUCTORY COMPUTER SCIENCE COURSES TO FACILITATE COLLABORATIVE PROGRAMMING PARADIGMS*

Alan Shaw, Ph.D.
Kennesaw State University
CSIS Department
Kennesaw, GA 30144, USA
ashaw8@kennesaw.edu

ABSTRACT

Many sophisticated collaborative technologies and social computing systems are connected to a growing cyberculture that continues to emerge among today's population of students. This type of collaboration is so prevalent in popular Web 2.0 computer systems, that it makes sense to begin to incorporate some type of coverage of such systems into CS 1 and CS 2 course material. Learning how to write collaborative software using peer-to-peer programming models can help introductory students understand collaborative systems better and it might encourage more students who are active users of these systems to major in computing disciplines in order to pursue related computer programming career path. Very few CS 1 and CS 2 textbooks contain content related to the development of collaborative software systems, presumably because of the extreme complexity involved in developing software that communicates with other software over a network. In this paper, a centralized peer-to-peer system framework is presented that offers a simple way to incorporate class work and projects that involve collaborative programming and networking paradigms at the level of introductory Computer Science courses.

Keywords

CS Education (CSED), Pedagogy, Collaborative Programming, Peer-To-Peer Programming

1. INTRODUCTION

One obvious outcome of living in the "Information Age" for modern students is that they typically grow extremely familiar from a user's point of view, with internet based systems like the Web in order to do well in society at large, as well as when dealing with university systems. Social networking sites like Facebook, Myspace, Twitter and Bebo are also beginning to play an increasingly significant role in the online experience of many students. The biggest of them all, Facebook, declared that it had topped 500 million active users in July of 2010 [1]. These factors are leading to the emergence of a growing cultural phenomenon that has been called the cyberculture [2].

The cyberculture is associated with Web 2.0 systems which involve collaborative and social computing applications that are very popular with many students. For members of the cyberculture, networking technologies have become critical to communications, community building and learning about the world. Moreover, this is the case whether or not they have an indepth understanding about how collaborative networking software works from a programming point of view.

Because of these factors, many of today's students who take introductory Computer Science classes will already have certain online experiences which would help them to see the relevance of learning about programs that work collaboratively within a social computing context. This predisposition is an argument in favor of including this type of material in introductory Computer Science classes where a fundamental goal is to introduce core CS concepts in a way that helps students to understand their application and their relevance. This is a different but related idea to the argument put forth by Buckley et. al [3], who argue that Computer Science education should be more "socially relevant":

> We propose that by reconsidering Computer Science education in the light of societal and interpersonal relevance, and by coupling it with early outreach and improving the image of our profession, we can have a significant impact in increasing the number and diversity of students in our programs. [3]

In their argument, both societal and interpersonal relevance is an important goal in making CS education effective for a wider audience of potential CS students. They reference the work of Gehrke et al [4] to make the case that, "true real-world problems have not traditionally been brought to undergraduate majors, and therein lies a missed opportunity to make Computer Science an attractive choice for incoming freshmen." Their argument goes farther, stating that, "We believe that computing for a cause, or computing as a means to solve problems, rather than as an end in and of itself, is key to attracting this generation to our discipline."

In discussing real-world problems, their article mentions, "how can computation help in organizing my files, my music, my vacation photos," and then, in the same

sentence, it goes on to mention, "can we use computation to determine how to evacuate Houston in 72 hours under the threat of a Category 4 hurricane?" The first is an example of interpersonal relevance, and the second is an example of societal relevance. Although their article makes the case that relevance should play a critical aspect in introductory Computer Science courses, their article focuses on a different type of relevance than is being addressed in this article. In this article, the argument is not to advance the goal of societal or interpersonal relevance in CS curriculum, although we do not dispute that goal. Instead, the goal here is to encourage students to see the "pedagogical relevance" of CS concepts by making direct links between those concepts and the online collaborative activities that students are already involved in outside of the classroom via their collaborative networking and social computing activities.

In this article, we argue that most students already are involved in a collaborative online culture that sees the technology of collaboration as a relevant issue in and of itself. Rather than argue that we need to radically alter the set of learning goals and outcomes we have for introductory Computer Science courses, in this article we argue that we can approach those same learning goals and outcomes by adding curricula that links established CS 1 and CS 2 material with collaborative programming paradigms that have connections to social computing activities within the cyberculture.

Social computing is implemented on top of collaborative networking systems, and teaching about collaborative networking systems can provide opportunities to augment the teaching of traditional computing concepts in new and engaging ways. Students can learn the same core CS principles while creating collaborative networking programs.

The concept of creating collaborative networking programs is not the same idea as programming collaboratively, and so this idea should not be confused with the important work that is being done in that area. An example of the idea of teaching computer science by having students program together collaboratively in "pair programming" teams has been shown to have demonstrable benefits in computer science education at collegiate levels [5] [6]. However, those teams produce a single program, instead of producing programs that communicate with each other. The approach presented in this article is about creating collaborative programs, and we are not addressing whether to have students write such programs while programming collaboratively or not. The benefits or difficulty in doing that, are not addressed in this article.

2. THE BASICS OF CREATING SIMPLE PEER-TO-PEER (P2P) SESSIONS

To explore approaches for creating collaborative networking programs within introductory Computer Science courses, we have developed a centralized peer-to-peer server and a framework of Java libraries that allows students to write Java programs that communicate with one another using certain collaborative programming paradigms. We believe that the classes and methods involved in this framework are no more difficult to use than the traditional classes and methods students learn to use in introductory Computer Science courses. For example, in most CS 1 text books, students create a Scanner object from the Scanner class to read in input from a user with something similar to the following commands:

```
Scanner input = new Scanner(System.in);
System.out.print("What is your name? ");
String name = input.nextLine();
System.out.println("Hello " + name);
```

These four lines ask the person running this program to enter his or her name, and then the program prints out "Hello ", followed by the name that the user has entered. Within our framework, the programmer can ask similar questions of a person over a network, instead of just questioning a user sitting at the same computer the program is running on. In order to do this, the programmer needs to know the username of a student they want to communicate with over the network in advance. But that is all they need to know. The more complicated networking details are hidden from the beginning programmer. The following five lines use our P2PSession class to implement a question and answer session over a network in a way that is intentionally similar to what was done with the Scanner class above, but in this example, a centralized peer-to-peer network server connects one programmer's program to another program written by a peer.

```
P2PSession p2p = new P2PSession("Student 1");
p2p.talkTo("Student 2");
p2p.sendString("What is your nick name? ");
String nickname = p2p.receiveString();
p2p.sendString("Hello " + nickname);
```

In the lines above, "Student 1" and "Student 2" would be usernames chosen by two students in a CS 1 course who are working to get their programs to communicate together. They can also use their real names if that is what they decide. When the P2PSession object is created in the above code from the P2PSession class on the first line, the P2PSession constructor registers the name of the first student with the server. On the second line, the P2PSession object tells the server the name of the other student's program that this P2PSession object will be communicating with in subsequent commands. After that, the student doesn't have to worry about how the server maintains a connection between the two students' programs. The programs just send and receive communications between each other in real time. With this basic framework, there are many collaborative applications that students can create.

One type of application that students can create with this framework is a chat application in which two programs enter a loop sending messages and responses back and forth until one of them sends some sentinel value to end the loop. Such a program would still use the Scanner class to get input from the keyboard of each user, but the keyboard input would be sent as output to the other user over the network using the P2PSession class, which would also receive the other user's output as this user's input:

```
String outputLine, inputLine;
Scanner keyboard = new Scanner(System.in);
P2PSession p2p = new P2PSession("Student 1");
p2p.talkTo("Student 2");
do {
    System.out.print("Message to send: ");
    outputLine = keyboard.nextLine();
    p2p.sendString(outputLine);
```

```
        if (outputLine.equalsIgnoreCase("quit"))
              break;
        inputLine = p2p.receiveString();
        System.out.print("Message received: " + inputLine);
    } while (! inputLine.equalsIgnoreCase("quit"));
```

The above loop implements a simple chat application between two users. Both students in the chat can run versions of this program after changing it so that they each put their own name in for "Student 1," and their partner's name in for "Student 2." Then they are both going to send and receive messages to one another until one of the two of them types in "quit."

This type of program presents students with the simple idea that there are text processing rules involved in a chat conversation. In a previous paper, we show how this can lead to a curricular unit where students use selection statements and String handling commands to process the text received and then generate automatic responses based on matching potential keywords in the input that comes from over the network [7]. When a student produces a program that generates automated responses to inputs, they have created a type of program called a "chatbot." A chatbot can be designed to continually respond to text inputs received from another program or user in a way that can hold a dialogue with that program or user. In fact, some chatbot programs attempt to respond in as human-like a fashion as possible, at times attempting to act as a general conversational program. This type of chatbot has the distinct potential for fooling a user at times into thinking that they are chatting with a human. For this reason, discussions of chatbots can lead to a discussion about the Turing Test and the potential and limits of artificial intelligence. Perhaps the earliest example of this type of software was created by Joseph Weizenbaum in 1966 [8]. His Eliza program acts as a psychoanalyst to some extent in the way that it responds to the user, and we show students a version of his program in order to help them see the potential.

In writing complicated chat programs, students learn how to handle strings, arrays, selection statements and loops, which is addressed in a previous paper on that topic [7]. What is not addressed in that paper is how other types of collaborative networking software can help student to learn basic object oriented programming principles. These other types of collaborative programs involve what we call the "Shared Object Model" and the "Shared Object Passing Protocol Model."

3. THE SHARED OBJECT MODEL

To learn simple object oriented program design, students in CS 1 courses are often taught to develop special classes for storing and managing data objects. One example of a program that needs to store and manage data is a TicTacToe program. In CS 1 classes, we have students create a TicTacToe program. In that programming project we have them create a TicTacToeBoard class that instantiates an array of nine integers to represent the TicTacToe board. This class must also have accessor and mutator methods for accessing and modifying the array according to the rules of the game. Each of the nine integers corresponds to one position on the board such that if it is a zero that position is empty, if it is a one that position has an X in it, and if it is a two that position has an O

in it. The students then use their TicTacToeBoard class to create a "board" object, and then to play a TicTacToe game using that object. The TicTacToeBoard class must also include a gameState() method. That method returns a zero if the game is not over, a one if X has won the game, a two if O has won the game and a three if the game is a draw. Using their "board" object, the students must create a game loop that allows two players in turn to put an X or an O into an empty position on the board, being sure to call gameState() each time to determine when the game is over.

This game can be played by one player against himself or herself, and that is how the students initially implement this game. However, the game is much more interesting to play as a peer-topeer game played over a network. To do this, two programmers write separate programs that interact with one another using the "Shared Object Model." That model consist of two programs that send a shared data object back and forth between themselves in turn while each program can modify the values stored in the data object during its turn according to agreed upon rules.

For the TicTacToe game, the object sent over the network is the array of nine integers, and when the students do this project they have already learned how to use that array in the TicTacToeBoard class to play the game. In the networked version of the game, one student will just write the X player's program and the other will just write the O player's program. The X player goes first and the O player goes second. Each student's program in turn marks a position on the board by changing one of the zeros in the array to the appropriate value for that player (a one or two) and then passes the array to the other player's program. The passing of the array back and forth goes on until one player wins or there is a draw after nine moves that do not result in a win.

The students must each use a method called sendIBuf() that takes an array of integers and sends it to the other player over the network. After registering with the P2P server and determining who they will "Talk To" in the same way discussed above, the students must write code that will send and receive updates to the board's integer array during a game loop that goes on until the game is over. To do this, students create code something like the following where an array called "squareValues" is in the TicTacToeBoard class, and it has a getSquareValues() accessor:

```
p2p.sendIBuf( board.getSquareValues() );
if (board.gameState() == 1)
      System.out.println("Game Over!\n The Winner Was X");
else if (board.gameState() == 2)
      System.out.println("Game Over!\n The Winner Was O");
else if (board.gameState() == 3)
      System.out.println("Game Over!\n It Is A Draw");
```

This shows how a student's program sends the shared integer array object of the board to the other student's program and then tests it to see if the game is over or not. Each program does this after its user has made their move, and the new board is sent to the other user before testing its state because each program needs the updated board values before either program can end the game. But each program must also have code to receive the shared integer array object sent by the other program, and that code needs to use a mutator method to set the "squareValues" array sent by the other user. To

do this the players must use a P2PSession method called recieveIBuf() that receives an integer array from the other player over the network:

```
board.setSquareValues( p2p.receiveIBuf() );
if (board.gameState() == 1)
      System.out.println("Game Over!\n The Winner Was X");
else if (board.gameState() == 2)
      System.out.println("Game Over!\n The Winner Was O");
else if (board.gameState() == 3)
      System.out.println("Game Over!\n It Is A Draw");
```

Most of the logic for playing the TicTacToe game is not shown here, but what is shown in the above two sets of program examples are the lines that need to be added to the un-networked TicTacToe game so that it can now send and receive the shared array of data back and forth between two programs that are collaborating over a simple centralized peer-to-peer network using the sendIBuf() and receiveIBuf() methods of the P2PSession class.

This type of peer-to-peer code can work within a program that uses a graphical user interface (GUI) for the TicTacToe game or one that just uses text. So, those details have been left out, although if a GUI is used, then it would be better to use methods from the JOptionPane class for the inputs and outputs instead of the using the Scanner class and the System.out.println() method.

4. THE SHARED OBJECT PASSING PROTOCOL MODEL

The next peer-to-peer model that we go over with students is the "Shared Object Passing Protocol Model." In this model, students don't just share a particular data object. What they do is work with a shared protocol for passing data objects back and forth during a shared activity like a game. To learn how to do this the students write a networked version of the Battleship game. In the Battleship game, separate coordinate boards exist for each player, and they each must guess the coordinates of five battleships that will appear on their board when they find them. Each player starts with their own set of five images at five different coordinate positions that they determine. They each know the coordinates of their own five battleships, but they must guess the coordinates of their opponent's battleships.

When the game starts, each player in turn guesses an x and y coordinate and sends those coordinates to their opponent in a two integer array using the sendIBuf() method. The battleship images have both width and height, so each guess just has to fall somewhere within the range on the board where that image would be shown if it was drawn. If a student's guess hits one of their opponent's battleships, the hit is recorded for the student who guessed right, and then that student sees the battleship image show up on their board. The players take turns guessing coordinates and seeing if they get any hits. A hit is only recorded the first time a student gets a particular battleship. The first player to get five separate hits wins.

In this game, each player sends and receives more than just the x and y battleship coordinates, they must also send a set of other data values which are needed to play the

game. Within the game loop, one player sends a two integer coordinate position to the other player, and then either receives a value of one or zero back depending on whether they hit a ship or not, respectively. If they do hit a ship, they also get back another 4 integers that determine the ship's position, consisting of its top left (x,y) coordinates and its width and height. And lastly, if they do hit a ship, the player gets sent the data of the image object for the ship which was hit. This model of first sending two integers and then receiving up to five integers and an image object back during each player's turn represents a simple protocol that each student must learn how to implement in their game loop code.

Programmatically sending and receiving the two integer array of the coordinate guesses is pretty much identical to sending and receiving the nine integer array in the TicTacToe game. But after sending a two integer coordinate guess to another program, each student needs to create code that receives a one integer value in response, and then depending on the value of that response, the student must also implement code that receives four more integers followed by an image. The order and interpretation of all the data sent and received represents the shared protocol. Implementing the protocol and getting it to work in the game is the challenge that the students face. Below is an example of what one implementation of this protocol might look like if xGuess and yGuess were integer variables with values determined when the player made their (x,y) coordinates guess:

```
int [ ] guessCoordinates = { xGuess, yGuess };
p2p.sendIBuf(guessCoordinates);
int [ ] answerReceived = p2p.receiveIBuf();
if (answerReceived[0] == 1)  {
        int [] imageCoordinates = p2p.receiveIBuf();
        Image battleship = p2p.receiveImage();
        int x = imageCoordinates[0];
        int y = imageCoordinates[1];
        int width = imageCoordinates[2];
        int height = imageCoordinates[3];
        page.drawImage(battleship,x,y,width,height,null);
}
```

The above code uses an additional P2PSession method called receiveImage(). There are many other methods in the P2PSession class to facilitate this type of peer-to-peer programming, including the sendImage() method. The above code also assumes that the user has created a Graphics object called page to do the drawings on, which is a concept covered in many CS 1 and CS 2 textbooks.

5. SYNCHRONICITY

To get the Battleship game working, each student must do more than just produce something like the code shown in the above lines. The students must also develop an algorithm and write program code which handles the back and forth between the two players, with one player's algorithm being for the first player, and the other player's algorithm being for the second player. And the students must also deal with the issue of

"synchronicity." An important fact in peer-to-peer programs is that each program must act in synch with the other. That is to say, when one program is sending an image to the other program, the other program must know it is an image being sent and not an integer array. This is addressed to some degree when the students agree on a shared protocol by which to implement their algorithm, but that does not completely address this issue. That is because one player's program might crash while the other player's program is still running, and then the player with the crashed program may start it running again before the other players program was stopped and started again. Now the two programs are both at different points in their execution, and this will cause major problems even if they are using a shared protocol.

To resolve this type of problem, the P2PSession class has a method in it called handshake(). The handshake() method is a special method that requires that other program also runs its handshake() method before it continues on with the rest of the program. So each program can use this method to be certain that the other program is starting at the same starting point. In other words, each program should start out with a handshake before beginning the rest of the algorithm that the students are implementing. If a handshake is at the beginning of each program, then if one program crashes and then starts again, that program will wait for the other student to end their program and start it again from the beginning. This approach means that each student's program should start off with a call to the handshake() method right after the talkTo() method has been called:

```
P2PSession p2p = new P2PSession("Student 1");
p2p.talkTo("Student 2");
p2p.handshake();
```

6. CONCLUSION

By adding the P2PSession class and the framework we have developed and its methodologies to introductory computer science courses, it is possible to teach students how to implement collaborative programming paradigms, and to create social networking activities like chat applications, as well as applications that share data objects or that share data object passing protocols. By having students develop these types of programs in CS 1 and CS 2 classes, we have found new ways to demonstrate traditional CS 1 and CS 2 programming principles through activities that we believe resonate with students as being part of their growing cyberculture. This may help some to see the issues presented in a light that seems more relevant and compelling since it relates to that cyberculture. This is possible in introductory classes because the framework's back-end libraries hide some of the complexities of the client server networking communications involved in a way that does not remove the challenging and compelling nature of building powerful networked applications that engage students in interesting programmed interactions. In the future, we will seek to develop further curricular units that use collaborative programming in related ways to allow students to create other types of networking applications and explore their underlying paradigms.

7. REFERENCES

[1] It's Official: Facebook Passes 500 Million Users. Mashable / Social Media. (July 7, 2010). DOI= http://mashable.com/2010/07/21/facebook-500-million-2/

[2] Manovich, L. "New Media From Borges to HTML." *The New Media Reader*. Ed. Noah Wardrip-Fruin & Nick Montfort. Cambridge, Massachusetts, 2003: 13-25.

[3] Buckley, M., Nordinger, J., and Subramanian, D. "Socially Relevant Computing." *ACM SIGCSE Bulletin*, 40(1) (2008) : 347-351.

[4] Gehrke, M., Giese, H., Nickel, U.A., Niere, J., Tichy, M., Wadsack, J. and Zündorf, A. 2002. "Reporting about Industrial Strength Software Engineering Courses for Undergraduates." *Proceedings of the 24th International Conference on Software Engineering*, ACM, May 2002.

[5] Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., Balik, S. "Improving the CS1 experience with pair programming." *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, Reno, NV, 2003, pp. 359 – 362.

[6] Wiebe, E., Williams, L., Petlick, J., Nagappan, N., Balik, S., Miller, C., Ferzli, M. "Pair Programming in Introductory Programming Labs." *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition*.

[7] Shaw, A. "Using a Collaborative Programming Methodology to Incorporate Social Computing Projects into Introductory Computer Science Courses." *Proceedings of the 8th International Conference on Information Technology: New Generations (ITNG)*, 2011, Las Vegas, Nevada.

[8] Weizenbaum, J. Eliza. Massachusetts Institute of Technology, *Communications of the ACM*, 9 (1) (January 1966): 36-35.

SIMULATION-BASED COMPARISON OF SCHEDULING TECHNIQUES IN MULTIPROGRAMMING OPERATING SYSTEMS ON SINGLE AND MULTI-CORE PROCESSORS*

Hala ElAarag, David Bauschlicher, and Steven Bauschlicher
Department of Mathematics and Computer Science
Stetson University
Deland, FL 32723
helaarag@stetson.edu

ABSTRACT

The operating systems course is one of the essential courses in the computer science curriculum, and CPU scheduling is a crucial component of this course. Multi-core processors have become the norm of modern day processors. In this paper, we present a project that not only helps students understand the different CPU scheduling techniques but also studies the effect of multi-core processors on the different scheduling techniques. We show the results of simulating three interesting scheduling techniques using different ratios of CPU-bound to IO-bound processes on single core and multi-core processors.

INTRODUCTION

CPU Scheduling is an essential component of multi-tasked operating systems. Love [1] covers scheduling in Linux. Solaris scheduling is described by Mauro and McDougall [2] while Russinovich and Solomon [3] discusses scheduling in Windows. The objective is always to maximize CPU utilization and hence increase computer productivity. This objective is obtained if the CPU switches among processes in order to have some process running at all times.

Operating systems textbooks [4-7] describe many scheduling techniques. The simplest scheduling algorithm is First-Come, First-Served. Shortest Job First provides the optimal waiting time. However, in this technique the length of each process's next CPU burst should be known a priori, which makes it impossible to implement in practice. In the Shortest Remaining Time Next scheduling algorithm, the process with shortest remaining time is scheduled for execution. In Priority scheduling, each process has a priority and the one with the highest priority is executed first. Round Robin scheduling is the most widely used technique. In this technique, each process is assigned a quantum, and if the quantum expires and the process did not finish it goes back to the tail of the ready queue.

In this paper we present a project where students studied three CPU scheduling techniques: Multilevel Feedback Queue (MLFQ), Lottery Scheduling and Fair Share Scheduling. MLFQ utilizes multiple queues and allows processes to move between queues dynamically. In this approach, processes "find their own level" based on their CPU burst [8]. Processes that have a larger CPU burst moves to a lower-priority queue. In Lottery Scheduling [9], each process is given some lottery tickets. A lottery is held at regular intervals and the winner is determined by selecting a ticket at random. The winning process gets to be executed next. A process will have a chance of winning proportionate to the number of tickets it has. To increase the chances of winning, higher priority processes can get more tickets. Lottery Scheduling guarantees a non-zero probability for any process to get executed and hence solves the starvation problem. The basic idea behind Fair-Share Scheduling [10] is to divide CPU time evenly among users and then among processes. For example, if we have two users X and Y in the system and each has one process, A and B respectively. Then these will be scheduled in this fashion: A B A B A B …., resulting in each user getting 50% of the CPU time. However, if user X has two processes A and C, with 50% to each user, the processes will be scheduled as A B C B A B C B…., then A will take 25%, B 50% and C 25%. If on the other hand a third user Z has a process C then each user will take 1/3 of CPU time and the processes are scheduled as A B C A B C….., with A taking 33.3%, B 33.3% and C 33.3%.

The rest of the paper is organized as follows. Section 2 presents the different metrics that could be used to evaluate the different scheduling techniques and explains details of our simulation. In section 3 we discuss our results for a single core processor, and section 4 presents the results for multi-core processors. Finally we conclude the paper in section 5.

METRICS AND SIMULATION

There are multiple performance metrics that could be used for comparing different scheduling algorithms. They include [4, 6]:

- Minimize waiting time: lower the amount of time the process spends in the ready queue.
- Minimize turnaround time: get out of the system quickly. The turnaround time is calculated from the time of process submission till the time of completion.
- Minimize response time: quickly finishing interactive processes. The response time is calculated from the time of submission until the time of getting the first response.

- Maximize CPU efficiency: keep the CPU as busy as possible.
- Maximize throughput: execute as many processes as possible per time unit.
- Ensure fairness: treat all processes equally and give them equal amount of CPU time.

In this paper we chose waiting time as our performance metric. The scheduling techniques under investigation were implemented in Java and run under its virtual machine. A process object was created and used for all scheduling techniques. Each process object contained the total amount of time a process would take to complete and the amount of CPU time it needed during every execution. If a process's CPU execution time is low, it is considered an I/O bound process because it will quickly be swapped in and out of the CPU. On the other hand, if it is high, it is a CPU bound process and will probably be preempted before it finishes. A class was created for each scheduling technique that determines the next process object to be executed and keeps track of the time required to execute an array of process objects.

To simulate the Lottery Scheduling technique, we started every process with 20 tickets. To decide the next process, one of the tickets is drawn from the entire pool at random. If the process is I/O-Bound, and therefore needs to be called more frequently, 5 more tickets will be added every time the process is run. In this way, the I/O process will have a higher chance at getting called each time. The maximum number of tickets each process can obtain is 200. So eventually the I/O processes will have a 10:1 chance of getting picked over the CPU processes. The quantum value used for the CPU was set to 200 time units, so most processes will be distinctly I/O or CPU bound.

The Fair Share Scheduling technique deals with users and splitting up their time quantum equally. In our simulation the entire CPU process time is split equally between all users, and those users can implement their own way of scheduling their processes. For instance, if a system had three users running processes, each user would get 33.3% of the processing time to use how they wish. If another user was added, each user would now get 25% of the entire process. Theoretically each user could implement a different scheduling technique for its own processes, but we choose to use fair share for those as well. Each process gets the same time quantum before it is stopped and moved to the end of the queue which is a basic implementation of round-robin. In our implementation, the time quantum each user's queue had is 50 time units, with each user getting the same slice of processing time.

The Multilevel Feedback Queuing (MLFQ) technique makes use of multiple queues, raising and lowering CPU processes based on their burst time. Our implementation is made up of three queues: top, second, and third priorities. Each time the scheduler wants to select a process, it randomly picks a queue. The queues are weighted so the top queue is picked 70% of the time, the second 20% of the time and the third 10% of the time. The top queue has a time quantum of 50, the second 200, and the third 350. Once a queue is picked, it is processed either for its full time quantum or until the process's CPU burst is finished. If the process does not finish its burst, it is lowered down one queue if possible. Those that do finish their burst are raised one level if possible. Because the first queue is picked most often, those processes having a short CPU burst will be rewarded. CPU

processes that are I/O bound are prioritized for short CPU bursts while CPU bound are lowered in priority.

RESULTS OF SINGLE CORE SIMULATIONS

Each technique was tested with three different ratios of CPU-bound to IO-bound processes: a 25-75 ratio, a 50-50 ratio, and a 75-25 ratio. The specific CPU execution times are chosen at random but on average will be around 350 time units for CPU-bound processes and 50 for I/O-bound processes. The specific total time to execute each process is a random number between 1000 and 2000. Processes are all created initially in order to simulate a relatively standard load that can be compared between all the techniques. The average waiting time, which is the amount of time a process spends in the ready queue, for the Lottery, Fair Share and MLFQ scheduling techniques are shown in Figures 1-3.

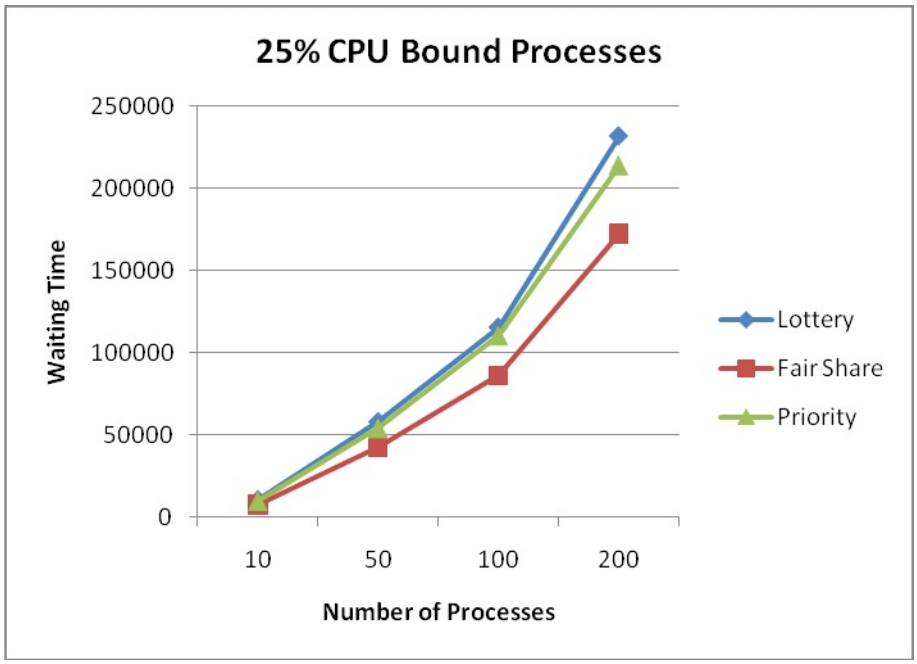


Figure 1: Waiting time with process mix 75% I/O bound and 25% CPU bound

As shown in the figures, Fair Share Scheduling performed noticeably better than Lottery Scheduling and Priority Scheduling under the different loads. Both Lottery and Priority Scheduling processes have similar waiting times no matter what the ratio of CPU-I/O bound processes is. However, Priority Scheduling has less waiting time than Lottery Scheduling if there are many more I/O bound processes than CPU bound

processes.

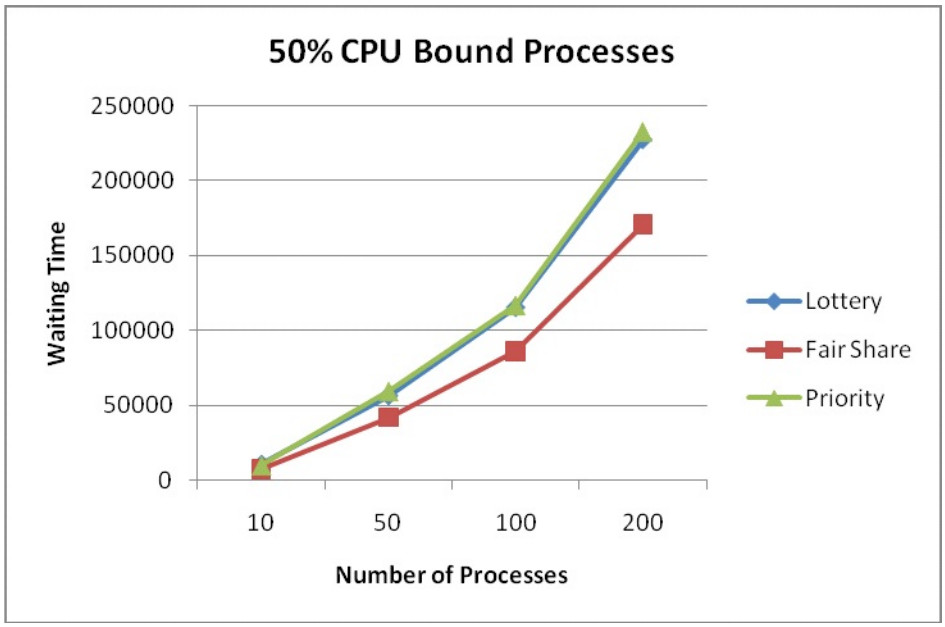


Figure 2: Waiting time with process mix 50% I/O bound and 50% CPU bound

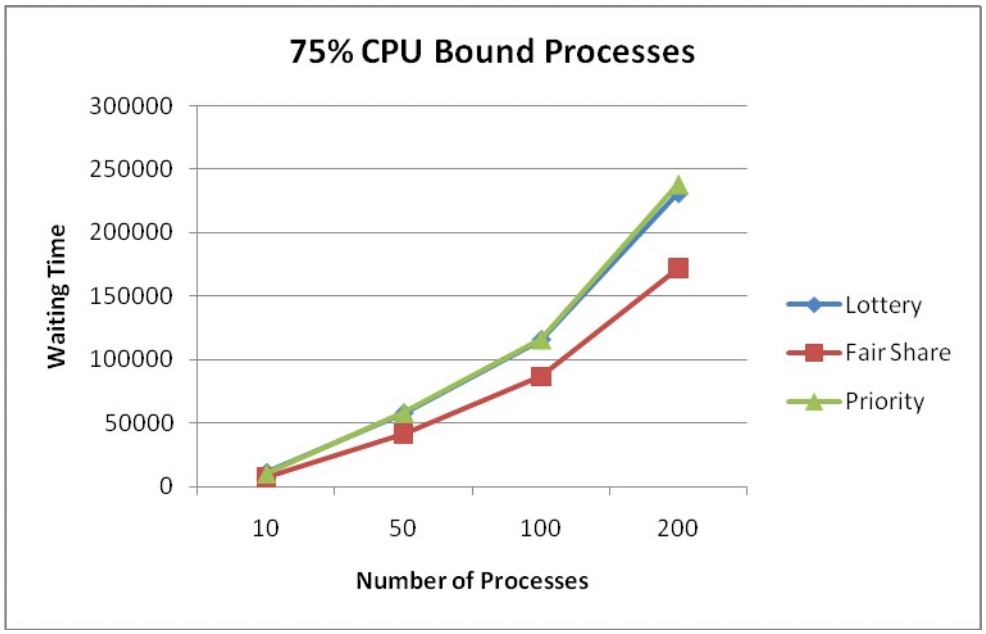


Figure 3: Waiting time with process mix 25% I/O bound and 75% CPU bound

RESULTS OF MULTI-CORE SIMULATIONS

Each scheduling technique was simulated to run 100 processes multiple times over a different number of processors. From Figures 4-6, one can notice that overall, Lottery Scheduling and Priority Scheduling performed better than Fair Share Scheduling in all cases, especially when the number of processors increased. As shown in Figures 4-6, the amount of time saved by using more processors seemed to increase almost linearly for each of the scheduling technique, although the MLFQ gained the greatest advantage by adding more processors. The percentage of CPU-I/O bound processes did not seem to affect the results too greatly, but having a 50-50 balance of the types of processes made lottery scheduling the most efficient technique when the number of processors was five or less. Otherwise, the processes' waiting times were always lowest while using the MLFQ.

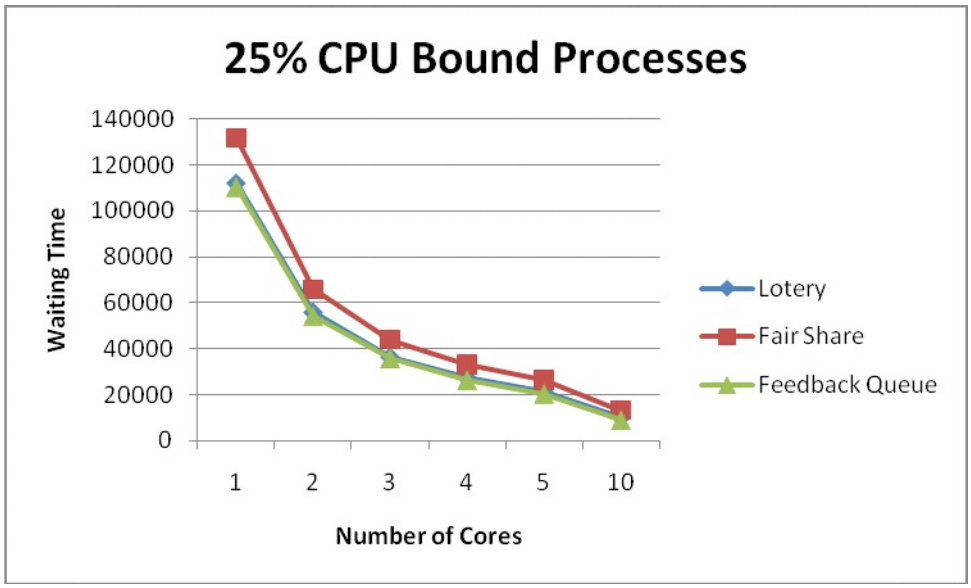


Figure 4: Waiting time vs. number of cores with process mix 75% I/O bound and 25% CPU bound

CONCLUSION

CPU scheduling is the basis of multi-programmed operating systems. Multi-core processors are becoming standard. In this paper, we presented a project that was proven to be beneficial for operating systems students to not only understand CPU scheduling but also study the effect of multi-core processors on different CPU scheduling techniques.

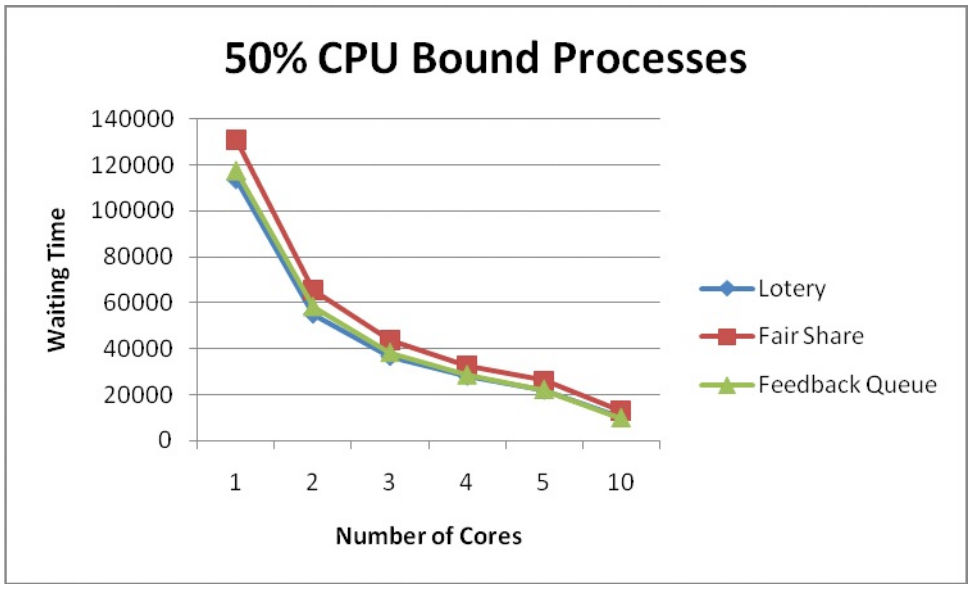


Figure 5: Waiting time vs. number of cores with process mix 50% I/O bound and 50% CPU bound

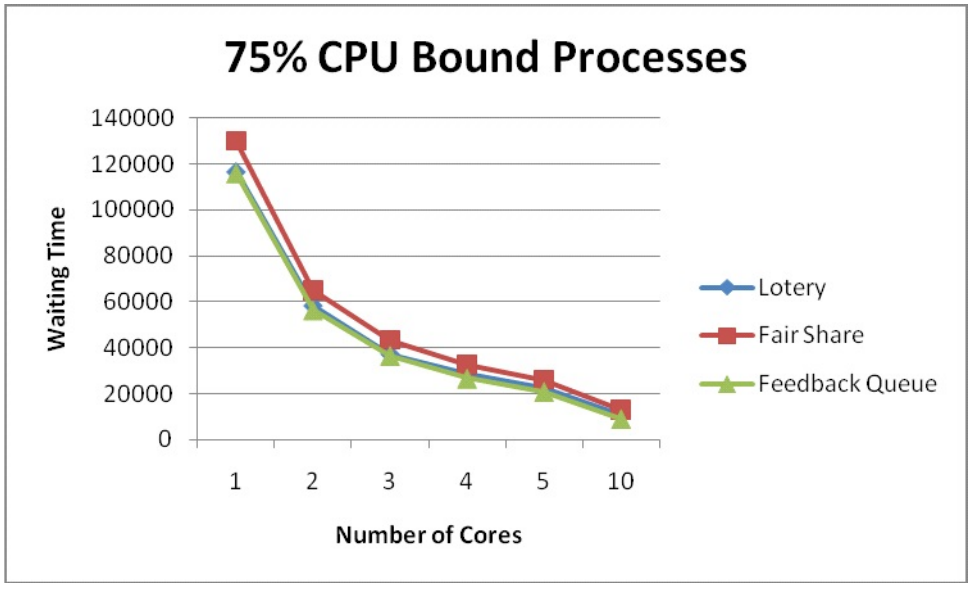


Figure 6: Waiting time vs. number of cores with process mix 25% I/O bound and 75% CPU bound

REFERENCES

[1] Love K., *Linux Kernel Development, 2nd Edition*, Developer's Library, 2005.

[2] Mauro J. and McDougall R., *Solaris Internals: Core Kernel Architecture*, Prentice Hall, 2007.

[3] Russinovich and Solomon, *Microsoft Windows Internal, 4th edition*, Microsoft Press, 2005.

[4] Silberschatz, Galvin and Gagne, *Operating System Concepts, 8th Edition*, Wiley, 2009.

[5] Tanenbaum, *Modern Operating Systems, 3rd edition*, Pearson Prentice Hall, 2008.

[6] Flynn and MacHoes, *Understanding Operating Systems, 3rd edition*, Brooks/Cole, 2000.

[7] Stallings, W., *Operating Systems Internal and Design Principles, 5th Edition*, Pearson Education, 2006.

[8] Hoganson K., Reducing MLFQ Scheduling Starvation with Feedback and Exponential Averaging, *The Journal of Computing Sciences in Colleges*, 25, (2), 196- 202, 2009.

[9] Waldspurger and Weihl, Lottery Scheduling: Flexible Proportional-Share Resource Management, *Proceedings of First Symposium on Operating System Design and Implementation*, 1-12, 1994.

[10] Henry G., The Fair Share Scheduler, *AT&T Bell Lab Technical Journal*, 63, (8), 1845-1857 1984.

TEACHING GAME PROGRAMMING USING XNA: WHAT WORKS AND WHAT DOESN'T*

James Harris
Georgia Southern University
Statesboro, GA 30460
912 478-7375
jkharris@GeorgiaSouthern.edu

ABSTRACT

In 2006, Microsoft became the first large gaming company to offer free of charge a professional development toolset for game programming called XNA. The pre-release version came out in March of 2006, and was followed by versions two, three, and four in 2007, 2008, and 2010 respectively [7]. XNA provides the ability to program directly on the graphics processing unit (GPU) using High Level Shader Language (HLSL). XNA also has a plug-in for Visual Studio that provides extensions to C# for developing games that run on the Xbox 360. In this paper, experiences learned while teaching game programming at the undergraduate junior/senior level in a CS Department using XNA 3.0 are related. This paper should be of interest to any computing instructors who are thinking about incorporating game programming into their curriculum.

INTRODUCTION

In fall 2010, a junior/senior level game programming course offered through Computer Sciences department at Georgia Southern University was scheduled to be taught for the first time. The catalog description [4] for this course is shown below:

CSCI 5439 Game Programming: 3-0-3

An introduction to game design and development including game physics, using game engines, using AI in games, creating multi-threaded games, and creating networked

---

games. Prerequisites: A minimum grade of "C" in CSCI 5332 and CSCI 5437 or permission of instructor.

Since Game Programming was a new course and fall 2010 was the first time it was being taught, the book(s) and development environment needed to be chosen. At that time, it was known that Microsoft had a development environment for the Xbox 360 named XNA. Microsoft XNA Game Studio is a toolkit for Visual Studio that creates a professional quality development environment for game programming and is distributed by Microsoft for free. Because our department was a member of the Microsoft Developers Network Academic Alliance (MSDNAA) and because of the lack of other low cost alternatives, XNA seemed like a logical choice to use as a development environment. We were not sure how teaching game programming using XNA would work because since XNA's inception, very little has been written on teaching game programming using XNA and there have been only a handful of academic publications that have focused on using XNA to teach game programming [3, 5, 6]. We found that all books on game programming using XNA that were available at that time were not academic in nature. They did not have many (if any at all) problem sets at the end of sections/chapters and they tended to focus on features rather than concepts. Most of them assumed an intimate knowledge of C#. This was a problem for us because as with most CS programs, our primary language is Java. The book that was finally chosen was XNA Game Studio Creators Guide (Second Edition) by Stephen Cawood and Pat McGee [2], primarily because of the good reviews on Amazon and the author's reputations. The fact that XNA with C# was a true game development environment outweighed the minuses and so XNA and C# were chosen as the game development environment for our game programming class.

COURSE DESIGN

The course was designed in two parts, 2-D game concepts and 3-D game concepts. The plan was that the concepts behind 2-D games would be simple enough that students could focus on the programming syntax of C# and by the time 3-D games were to be covered, they would be familiar enough with C# syntax that they could then focus on the more challenging concepts behind 3-D games. The course grading policy reflects this design and is shown in Table 1 below:

- Homework and quizzes 40%
- Midterm project (2-D game) 20%
- Final project (3-D game) 40%

Table 1: Grading Policy

Homework consisted mostly of small programming assignments to reinforce concepts and quizzes were used to test to see if students were doing their reading, showing up for class, and paying attention. For the midterm and final projects, students were required to submit a proposal for a project to create a game, produce a requirements document, design the game, implement the game, test the game, and then present the game. Requiring students to present the midterm and final projects turned out to be a very good idea. The midterm and final projects became an informal competition between

students to see who could produce the best game. Their motivation did not seem to come from their desire to achieve a high grade, but rather from a desire to have their game accepted among their peers. Since the projects were individual, students could not hide behind group members as many do in many group projects. As upperclassmen, the peer pressure was intense, since this was an important venue to showcase their talent. They even went so far as to try to hide programming secrets from one another so that they could have the best game.

Because the course was being taught for the first time, the course outline specified in the syllabus was not well defined. Students were told that the course outline would be followed as much as possible, but that there may be deviations from time to time and some of the course outline topics may not be covered. The course outline presented in the syllabus is shown below in Table 2.

- XNA Game Framework
- 2-D primitive objects
- 2-D vector algebra
- Animations in 2-D
- Shaders and High Level Shader Language
- Textures
- Backgrounds and z-buffers
- Collision detection in 2-D
- Midterm presentations
- 3-D modeling
- Camera, view, and projection
- 3-D vector algebra
- Animation in 3-D
- Light sources and shading
- Creating and using terrains
- Game physics
- Audio
- Networked Games
- Final presentation

Table 2: Course Outline

There were no dates given in the course outline because it was not known beforehand how long it would take to properly cover each topic.

COURSE IMPLENTATION

The first problem that surfaced before the class even started was the fact that the pre-requisite courses for Game Programming are Data Communications (CSCI 5332) and Computer Graphics (CSCI 5437). Both are junior/senior level CS courses. The Data Communications and Computer Graphics courses were designed as pre-requisites so that it could be assumed that students would have already had exposure to many of the concepts needed for game programming. Because of the fact that there were two upper division courses as pre-requisites, many students who wanted to take the game

programming class did not have the appropriate pre-requisites. In order to have enough students in the class for the class to "make", many students were given overrides to enroll without having the appropriate pre-requisites.

The actual topics taught in class along with the dates they were taught are shown below in Table 3.

Date	**Topic covered**
8/16	XNA Framework
8/18	XNA Framework (part II)
8/20	A sample 2-D game
8/23	A sample 2-D game with objects
8/25	2D Primitive objects
8/27	Rotation, translation, and scaling in 2-D
8/30	Animating 2-D objects with timers
9/1	Animating 2-D objects using parametric equations
9/3	Introduction to shaders
9/8	Changing colors using vertex shaders
9/10	Programming the GPU using High Level Shader Language
9/13	Texturing 2-D objects
9/15	2-D backgrounds and z-buffers
9/17	Collision detection in 2-D
9/20	Exporting games to the Xbox 360
9/22	Introduction to 3-D games
9/24	A sample 3-D game
9/27	A sample 3-D game (part II)
9/29	Camera, view, projection
10/1	Lines, planes and normal vectors in 3D
10/4	Rotations in 3-D
10/6	Translation and scaling in 3-D
10/8	Animation using parametric equations
10/11	Animation using parametric equations (Part II)
10/13	Light sources in 3-D
10/15	Phong shading with HLSL
10/18	Creating 3-D models
10/20	Midterm presentations

10/22	Midterm presentations
10/25	Midterm presentations
10/27	Midterm presentations
10/29	How to import an .fbx model into XNA.
11/1	Introduction to Blender
11/3	Modifying a 3-D model with Blender
11/5	Lab – Creating a 3-D model using Blender
11/8	Using armatures, and bones
11/10	Creating an animation using Blender
11/12	Animations with Blender (part II)
11/15	Exporting a Blender animation to XNA
11/17	Texturing 3-D models
11/19	Altering Blender animations in C#
11/29	Collision detection in 3-D
12/1	Creating and using terrains
12/3	Incorporating game physics into 3-D games
12/8	Final Exam – Final presentations

Table 3: Course Schedule

One thing that became apparent early on was that students learned much faster in game programming by example. Showing and discussing sample programs and then discussing the concepts needed drew much more interest than discussing the concepts first and then showing how they were applied using examples. For example, when you demonstrate a spaceship spiraling out of control, students become intensely interested in how that is done. If you cover parametric equations first and then show the spaceship spiraling out of control, their reaction would be "Neat! How did you do that?" Showing examples first provides a motivational opening necessary for student to remember concepts. Game programming is a wonderful venue for this, since most concepts in game programming can be viewed graphically. Mistakes become immediately apparent and once a mistake displays itself graphically through the improper movement of a necessary game object, students are highly motivated to try to get the object to animate correctly.

One problem that surfaced early on was the need to understand linear algebra and vector algebra. XNA uses matrix transformations to animate models and the XNA shaders use Phong shading and therefore require knowledge of vector algebra. The math background of students taking this class was clearly not sufficient. Too much time was spent having to teach the mathematics behind the code.

Another problem was that of creating models using primitive objects. Primitive objects in XNA consist of triangle strips, triangle lists, line strips, line lists, and point lists. To draw a primitive object, the type of primitive object and the corresponding

coordinates of the vertices as an array are specified as arguments in the *DrawUserPrimitives* method. To create a 3-D model, primitive objects must be pieced together. Needless to say, it is very tedious to create models using this method and the models generally do not turn out very well. The models are also very difficult to texture and animate. To overcome this in 2-D, C# PictureBoxes were used to store images. The 2-D models are then animated by moving the PictureBoxes around the screen, altering the images in the PictureBoxes, or using animated gifs as images. This works well for 2-D models. However, for 3-D games, modeling software is needed to create 3-D models. Most 3-D modeling software, such as Maya and 3D Studio Max are too expensive, so we chose Blender [1], an open source (GPL license) 3D modeling package that runs under Windows. One nice thing about Blender is that it does not require the Windows registry, so it can be installed on a jump drive. It is a fairly complete, albeit a little buggy, 3D modeling environment. It is easy to create, texture, and animate 3D models in Blender. You can also input a variety of 3D modeling formats. This is very useful because there is a plethora of free 3D models available online. The main problem with Blender is the time that is needed for students to become comfortable using it. The entire course could easily have been devoted to creating 3-D models using Blender. We did not starting using Blender until after the 2-D presentations were complete, which was a mistake. Blender should have been introduced at the start of the semester to give students time to become comfortable with its many features.

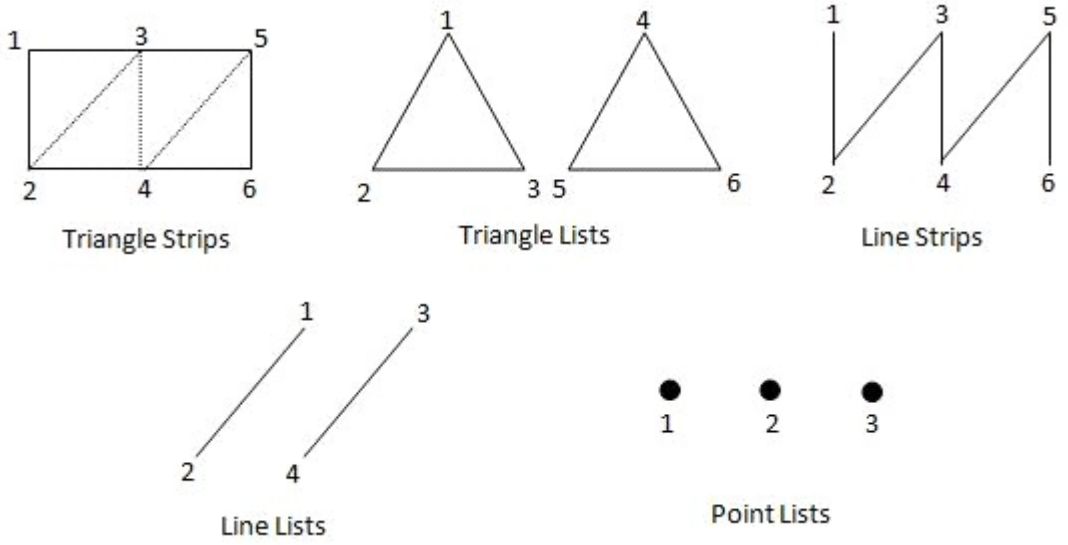


Figure 1: Primitive Objects in XNA

One of the major shortcomings of XNA is that the only 3D modeling format that it currently supports is the Autodesk fbx format. It is possible to import a model in a variety

of 3-D formats into Blender, export it to fbx format, and then import the fbx file into XNA. This is easy to do for 3-D models that are not animated. For animated models, this is a problem. XNA does not provide support for skeleton animation data, which is what fbx format uses. There is third party open source code that provides support for skeleton animation [8]; however, it requires that each vertex in the model be painted with something called "heat". Heat determines which bones are associated with which armatures in the model, so when you move a bone, the associated bones connected by armatures will move in unison. If one vertex is left unpainted, the model cannot be used. There was, at the time, no way to tell in Blender if a vertex was left unpainted and there were usually thousands of vertices in a model. At the end of the semester we found a Python script (Blender is based on Python) that would indicate if any vertices were not painted. However, there was not much time left in the semester for students to use Blender to produce animated models, so almost all animations in the final projects were hard coded in C#. This required much more detailed code and also requires students to have a much greater understanding of mathematics, so most of the animations in the final 3-D game projects were not as good as I had hoped they would be.

CONCLUSIONS

To summarize, I have several suggestions as a result of my experiences. First, students who take a course in game programming using XNA need to minimally have a background in computer graphics and since XNA depends on matrix transformations and vector algebra with analytical geometry for animation and shading, a math background in those topics also. Actually, a background in calculus in three dimensions would also be nice, but this is a balancing act because you always run the risk of having too many pre-requisites and nobody signs up for the class. Second, examples (with sample code) should be presented before their associated concepts. Third, 3-D modeling software should be introduced early in the semester and students should be using the modeling software throughout the semester. Fourth, there should be a grading policy with heavy weights on individual gaming projects and a requirement to have students present those projects.

REFERENCES

[1] Blender Homepage, http://www.Blender.org, retrieved January 5, 2011.

[2] Cawood, S., McGee, P., *Microsoft XNA Game Studio Creator's Guide, Second Edition* McGraw Hill, 2009.

[3] Ferguson, E., Rockhold, B., Heck, B., Video game development using XNA game studio and C#.Net, *Journal of Computing Sciences in Colleges*, 23 (4), 186-188, 2008.

[4] Georgia Southern University 2010-2011 Academic Catalog, http://students.GeorgiaSouthern.edu/registrar/2010-2011_GeorgiaSouthernCatalog.doc, 499, retrieved January 15, 2011.

[5] Linhoff, J., Settle A., Teaching game programming using XNA, *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education, 40 (3)*, 250-254, 2008.

[6] Shen, Y. Teaching Game Development Using Microsoft XNA Game Studio, *2009 Spring Simulation Multiconference (SpringSim'09* San Diego, CA), March 2009.

[7] Wikipedia, Microsoft XNA, 2011, http://en.wikipedia.org/wiki/Microsoft_XNA, retrieved January 9, 2011.

[8] XNA Animation Library, 2010, http://xnanimation.codeplex.com/, retrieved January 17, 2011.

A BINARY COUNTING OF RATIONAL NUMBERS*

Samuel C. Hsieh and C. Van Nelson
Computer Science Department, Ball State University, Muncie, Indiana, U.S.A

ABSTRACT

A one-to-one correspondence between positive binary numbers and positive rational numbers is defined and studied. Efficient algorithms to compute the rational number for a given binary ordinal number and to compute the binary ordinal number for a given rational number are presented and analyzed. This one-to-one correspondence and the related algori
thms provide a link between the well-known problem of counting rational numbers and a key topic in computer science: binary numbers. The binary ordinal number of a rational and that of its reciprocal are shown to be related in a simple manner.

1. INTRODUCTION

It is well known that positive rational numbers form a countable set. In discrete mathematics or computing theory classes, this is usually shown by using a table [e.g., 1, 2]. Fig. 1 shows part of such a table. The fraction in row *a* and column *b* of the table is $\frac{a}{b}$. The proof lists the fractions on the diagonals of the table, starting from the fraction $\frac{1}{1}$ located on the upper left corner. As shown in Fig.1, the first 11 fractions in the list are $\frac{1}{1},\frac{2}{1},\frac{1}{2},\frac{1}{3},\frac{3}{1},\frac{4}{1},\frac{3}{2},\frac{2}{3},\frac{1}{4},\frac{1}{5},\frac{5}{1}$. It should be noted that the listing skips those fractions that cause double counting, so that each rational number is listed only once. For example, in Fig. 1 the fractions $\frac{2}{2},\frac{4}{2},\frac{3}{3},$ $\frac{2}{4}$ and are skipped to avoid repetition in the list. The skipped fractions are alternative representations of some rational numbers that have already been listed.

This proof shows a way to enumerate positive rational numbers but does not provide an explicit formula for the one-to-one correspondence between integers and rational numbers. As this way of listing rational numbers skips certain fractions in the table, an

---

interesting question concerns how to efficiently compute the rational number for a given ordinal number and to compute the ordinal number for a given rational number in this list.

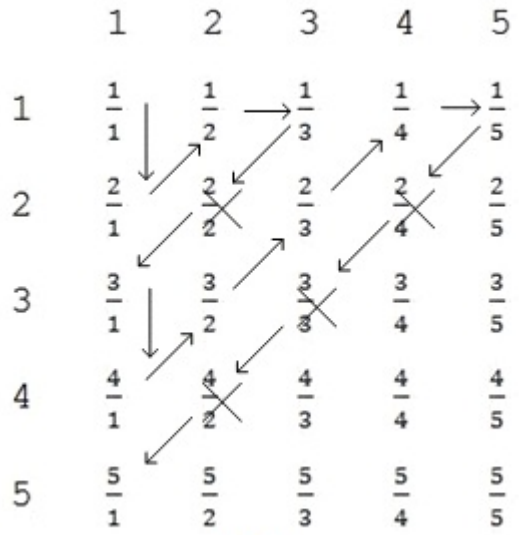


Fig. 1

In this article, we will present a one-to-one correspondence between positive rational numbers and positive binary numbers. As this mapping involves only reduced fractions, each rational number is uniquely represented and double counting due to alternative representations of a rational number is impossible. Moreover, computations of the nth rational number for a given binary ordinal number *n* and of the binary ordinal number for a given rational number can be efficiently performed. Efficient algorithms for both of these tasks will be presented.

A way of counting rational numbers based on a class of resistor networks is given in a related work [3]. The mapping presented in this article is based on binary numbers. This provides a direct link between the well-known problem of enumerating rational numbers and a key topic in computer science: binary numbers.

2. MAPPING BINARY NUMBERS TO RATIONAL NUMBERS

In order that each integer has a unique binary representation, the positive binary numbers used in this article have no leading zeroes - that is, the leftmost bit of a positive binary number is always 1. We will also adopt the following notations regarding binary numbers. For any binary number b, b.0 and b.1 represent the binary numbers formed by appending 0 and 1, respectively, to b. For example, if b is 110, then b.0 is 1100 and b.1 is 1101. In addition, the *tail* of a binary number b is the bit string formed by removing the left most bit from b. For example, the tail of the binary number 10011 is 0011.

Technically, the tail of a binary number is not necessarily a positive binary number as defined for this article because the tail may have leading zeroes, as demonstrated by the example.

We now present a function F from the set of positive binary numbers to the set of positive rational numbers. The function F is defined recursively as follows

a) F(1) is the fraction $\frac{1}{1}$.

b) For any positive binary number b, if F(b) is $\frac{x}{y}$, then F(b.0) i$\frac{x}{x+y}$ and F(b.1) is $\frac{x+y}{y}$.

Fig. 2 shows the binary numbers 1 through 111 and their images F(1) through F(111).

b	1	10	11	100	101	110	111
F(b)	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{2}{1}$	$\frac{1}{3}$	$\frac{3}{2}$	$\frac{2}{3}$	$\frac{3}{1}$

Fig. 2

It is known that if two positive integers x and y are relatively prime, then the integers x and $x+y$ are also relatively prime, and so are y and $x+y$. Thus, if $\frac{x}{y}$ is a reduced fraction, then both $\frac{x}{x+y}$ and $\frac{x+y}{y}$ are also reduced fractions. Since F(1) is a reduced fraction, and all other fractions in the image (range) of F are generated from F(1) by applying part b of the definition of F, possibly a multiple number of times, all fractions in the image of F are reduced fractions.

Suppose F(b) = $\frac{x}{y}$. From the definition of F, it should be obvious that $x = y$ if and only if b = 1 and that, for any b > 1, $x < y$ if and only if the rightmost (least significant) bit of b is 0, and $x > y$ if and only if the rightmost bit of b is 1.

3. ONE-TO-ONE CORRESPONDENCE

We now present a contradiction proof that the function F is a one-to-one correspondence, that is, F is both one-to-one and onto.

Suppose F is *not* one-to-one. There are binary numbers that share an image. Such binary numbers must have multiple bits, since the binary number 1 does not share its image $\frac{1}{1}$ with others. From the definition of F, the binary numbers having the same image must have the same value in their rightmost bit. Let two different binary numbers b and c share the image $\frac{x}{y}$. The binary numbers b' and c' obtained from b and c by truncating their common rightmost bit have the same image; that is, by the definition of

F, either $\frac{x-y}{y}$ $\frac{x}{y-x}$ depending on whether the truncated bit is 1 or 0. reasoning, the binary numbers b'' and c'' obtained from b' and c' in the same way share their image, and by the same reasoning b''' and c''' obtained from b'' and c'' in the same way also share their image, and so forth. This process eventually leads to the contradiction that the binary number 1 shares its image with another binary number.

Now suppose F is *not* onto. At least one rational number is not in the image of F. Let the reduced fraction $\frac{x}{y}$ represent one such rational number. Since $\frac{1}{1}$ is in the image of F, x and y cannot be equal. The fraction $\frac{x'}{y'}$, which is either $\frac{x-y}{y}$ $\frac{x}{y-x}$ r depending on whether $x > y$ or $x < y$, cannot be in the image of F (otherwise, by the definition of F, $\frac{x}{y}$ would be in the image of F). By the same reasoning, the fraction $\frac{x''}{y''}$, which is obtained fracti$\frac{x'}{y'}$ in the same way, cannot be in the image of F, and by the same reasoning, the fraction $\frac{x'''}{y'''}$, which is obtained fracti$\frac{x''}{y''}$ in the same way, cannot be in the image of F, and so forth. This process carries out the Euclidean algorithm to find the greatest common divisor of x and y. Since $\frac{x}{y}$ is a reduced fraction, x and y are relatively prime and this process eventually leads to the contradiction that $\frac{1}{1}$ is not in the image of F.

4. ALGORITMS

An algorithm to compute the rational number that is the image of a given binary number is presented as a C++ function `findRational` below. The value parameter `b` is an integer representing a given binary number. Within the function, `b` is treated as a binary number and manipulated with binary operations. The reference parameters `x` and `y` represent the rational number $\frac{x}{y}$ to be computed from the binary number `b`.
For example, suppose `x` and `y` are `int` variables, the function call `findRational(6,x,y)` will set `x` to 2 and `y` to 3. This result is correct because 6 converted to a binary number is 110 and F(110) is $\frac{2}{3}$.

```
void findRational(int b, int& x, int& y)
{
  if (b==1)
         x=y=1;
  else {
         int RMBit=b&1;
         findRational(b>>1, x, y);
         if (RMBit) x=x+y; else y=x+y;
       }
}
```

The algorithm closely follows the definition of F. When the given binary number b is 1, x and y are set to 1 because and F(1) is $\frac{1}{1}$. When the binary number b is greater than 1, the algorithm saves the rightmost bit, recursively computes the rational for the binary ordinal number obtained by b>>1, which is b with the rightmost bit truncated, and then adjusts the result by setting x or y to x+y depending on whether the saved rightmost bit is 1 or 0. Obviously, in terms of the number of bits in the binary ordinal number b, the running time of the algorithm is linear. However, in terms of the value *n* of the ordinal number, the running time is O(log *n*).

An algorithm to compute the binary ordinal number for a given rational number is given as a C++ function `findBinary` below. The value parameters x and y represent a given reduced fraction $\frac{x}{y}$, and the integer reference parameter b represents the corresponding binary ordinal number. In the function `findBinary`, b is treated as a binary number and manipulated with binary operations. As an example, suppose b is an integer variable, the function call `findBinary(2, 3, b)` will set b to 6. This result is correct because $\frac{2}{3}$ is F(110) and the binary number 110 converted a decimal number is 6.

```
void findBinary(int x, int y, int& b)
{
  if (x==y) b=1;
  else if (x > y)
    { findBinary(x-y, y, b);
      b= b<<1 | 1;
    }
  else
    { findBinary(x, y-x, b);
      b= b<<1;
    }
}
```

The function follows the definition of F in implementing the inverse of F. For the case where x and y are equal, that is, the fraction $\frac{x}{y}$ is $\frac{1}{1}$, the binary ordinal number b is set to 1. For other cases, the function recursively finds the ordinal number for $\frac{x-y}{y}$ or $\frac{x}{y-x}$ depending on whether x is greater than y, and then adjusts the binary number b accordingly: by appending a 1 to b if x is greater than y and appending a 0 otherwise. Clearly, the function is a recursive implementation of the Euclidean algorithm for computing the greatest common divisor of x and y, augmented with statements to construct the binary ordinal number b. Since the values of the parameters x and y are relatively prime (because they represent a reduced fraction), their greatest common divisor is 1. This guarantees that the recursive function will eventually terminate with x and y both having the value 1. In terms of the parameters x and y, the worst-case running time

of the algorithm is O(max(x, y)), but in terms of the value *n* of the binary ordinal number b, the running time is O(log *n*).

5. RECIPROCALS

It is interesting to note that the binary ordinal number of a rational number and that of its reciprocal are related in a simple manner. Specifically, if F(b) is the reciprocal of F(c) then the tail of b is the bitwise complement of the tail of c. This can be shown by induction. Since the reciprocal of F(1) is itself and the tail of 1 is null, the basis case is true. If F(b)= $\frac{x}{y}$ and F(c) $\frac{y}{x}$ and the tail of b is the bitwise complement of the tail of c (inductive hypothesis), then F(b.0) = $\frac{x}{x+y}$, which is the reciprocal of F(c. $\frac{x+y}{x}$, and F(b.1) = $\frac{x+y}{y}$, which is the reciprocal of F(c $\frac{y}{x+y}$. The tails of b.0 is the bitwise complement of the tail of c.1, and in the same way are b.1 and c.0 related.

The converse case, i.e., if the tail of b is the bitwise complement of the tail of c then F(b) is the reciprocal of F(c), can also be shown similarly.

6. SUMMARY

A one-to-one correspondence between positive binary numbers and positive rational numbers is defined and studied. Efficient algorithms to compute the rational number for a given binary ordinal number and to compute the binary ordinal number for a given rational number are presented and analyzed. The one-to-one correspondence and the related algorithms provide a link between the well-known problem of counting rational numbers and a key topic in computer science: binary numbers. The binary ordinal number of a rational and that of its reciprocal are shown to be related in a simple manner.

REFERENCES

[1] Rosen, K. H., Discrete Mathematics and Its Applications, McGraw-Hill, 6th ed., 2007.

[2] Sipser, M., Introduction to the Theory of Computation, Thomson Course Technology, 2nd ed., 2006.

[3] Samuel C. Hsieh, C. Van Nelson, and Logeshbabu Sampath, "Resistor Network-based Algorithms for Enumerating Rational Numbers," *Journal of Computing Sciences in Colleges*, 25, 5 (2010), 287-293.

COST-EFFICIENT DESKTOP VIRTUALIZATION EXPERIENCE AT GEORGIA SOUTHWESTERN STATE UNIVERSITY*

Dr. Simon Baev, Casey Weaver, and Cody Weaver
Department of Computer Science
Georgia Southwestern State University
800 GSW State University Drive
Americus, GA 31709
sbaev@canes.gsw.edu, cweaver4@radar.gsw.edu, cweaver3@radar.gsw.edu

ABSTRACT

This paper describes the pilot project started at the School of Computing and Mathematics of the Georgia Southwestern State University (GSW) in Fall 2009 and targeted towards setting up a cost-effective desktop virtualization environment in one of the computer laboratories. Following the initial setup it was decided to prepare a variety of project-oriented courses for GSW students, including those who enrolled through the exchange program with North Gujarat University, India. The pilot was built on the Citrix XenServer platform running on HP ProLiant ML350 server and Wyse V10L thin clients allowing for complete elimination of ordinary desktop stations and achieving delivery of virtual Windows XP desktops via RDP protocol.

INTRODUCTION

In modern computing environments, especially in those where cost-effective solutions are preferable because of the budgetary restrictions, it is very important to find a balance among needs of end-users (software they are about to run), scalability of the system, maintenance complexity, and efficiency in terms of minimizing costs and efforts associated with the replacement of obsolete technological units. Accounting to all these moving forces and requirements, *virtualization* appears to be one of the most suitable solutions. The formal definition of virtualization refers to physical abstraction of computing resources [1]. From the hardware perspective it appears that all available

resources are distributed among all installed *virtual machines* (VMs), where each VM is seen as an application by the *host* computer. On the other hand, each VM appears to be a standalone computing platform for the software it runs. Such a standalone platform has all crucial parts of a computing system existing in virtual form, e.g. virtual HDD is mapped to a file or files on the hosting HDD, virtual NIC delivers corresponding network traffic to the *mapped* physical network adapter, and so on.

Once the physical computing infrastructure is *virtualized*, it is desired to achieve the same user experience in a variety of ways including but not limited to the availability of the same software products, desktop environment, etc. Such an important set of user expectations can be addressed either by means of network delivery of the entire Desktop environment, e.g. Windows Terminal Server, or by means of allowing remote invocation of pre-selected pieces of software installed on the virtualized server. It is worth mentioning that the majority of nowadays desktop users prefer to deal with MS Windows products which results in focusing of most valued technological achievements into this area. In turn, such a practice assumes that end users not only are responsible for paying off the Windows (virtualized *server* and virtual *workstations*) licenses, but also all the accompanying licenses, for example the ones allowing use of Windows Terminal Server. Moreover, the technology used for remote desktop delivery through the network to *thin terminal stations* requires another investment associated with all described infrastructure being virtualized using *hypervisor* technology: single hardware server not only hosts guest VMs but also its own operating system (OS) which appears to be yet another virtual machine running under control of the hypervisor software which is in turn installed to the *bare metal*. There exist two major players on the hypervisor software market: Citrix with XenServer and VMware with ESXi. Both products are available free of charge but require additional software, i.e. Citrix XenDesktop, being installed and licensed to enable support of virtual desktops delivery. The overall investment can be seen as consisting from the following portions: powerful hardware server; Microsoft Windows 2008 Server OS, Windows Terminal Services (per-user price); Citrix XenDesktop (per-user price), and *thin terminals* (user-end computers with limited functionality used to present virtual desktops to end-users) which appear to be replacement of regular desktop computers. The price considerations are out of the scope of this paper but can be roughly estimated as following: $6k for the hardware server, $4k for MS Windows 2008 Server OS including 25 access licenses [2], $3k for 25 concurrent licenses for XenDesktop, and finally $5k for 25 thin clients (approx. $200 each) resulting in overall investment of $18,000. This solution is pretty much standard and appears to be employed by many organizations who wish to receive out-of-the-box working infrastructure with technical support by software vendors and other related services.

The *goal of this paper* is to introduce an *alternative* approach to virtualization allowing for the reduction of the total cost down to $11,000 assuming on-site existence of *Volume License Key* (VLK) for either Windows XP or Windows 7 OS (such an assumption can be considered valid for many organizations running hundreds of traditional desktop computers).

METHODOLOGY

The proposed methodology can be outlined as follows. The hardware server is used to run Citrix XenServer hypervisor hosting as many Windows XP (or Windows 7) virtual machines as the *maximum* number of *concurrent* end-users who are willing to receive unique desktop experience through dedicated thin terminals. More specifically, this number is set to 25 to be in consistence with the price estimate given above and to reflect overall capacity of the selected hardware platform (HP ProLiant ML350 G5 server). Along with the mentioned 25 Windows VMs, the hypervisor also hosts a Linux virtual machine, i.e. Debian Lenny, playing the role of *network gateway* by means of providing the following services: NAT [3], DNS [4], DHCP [5], and optionally WebCache for all installed Windows VMs. The hardware server is expected to have *two network adapters*: one to support connectivity to the Internet, and one to allow thin clients to communicate with dedicated VMs by means of RDP protocol [6]. More details on the the proposed network connectivity are shown in Figure 1 and explained in the following subsection.

Network Connectivity

The XenServer hypervisor is connected to two (matching the number of installed *network interface cards*) networks: *public* and *private* ones. The former has connectivity to the Internet through the GSW LAN and the latter serves the needs of virtual desktop delivery with no effect to the rest of the world. Each virtual machine may have connectivity to *any* number of configured networks, which makes possible to isolate Windows VMs from being accessed directly from GSW network and therefore from the Internet. On the other hand, the Debian Linux gateway, connected to both networks at the same time, allows for traffic *forwarding* between them and provides *on-demand access to the Internet* for all Windows VMs by means of running *network address translation* (NAT) service. Moreover, Linux VM is responsible for providing DHCP and DNS services to the private network which are in dynamical assignment of IP addresses (DHCP) and their direct and reversed resolution (DNS) to and from hostnames of Both services tightly communicate to each other making possible of dynamic update of corresponding configuration files in response to the change of private network structure. The dynamic re-configuration of both DNS and DHCP services especially useful accounting to the fact that thin terminals are also connected to the private network. At the boot time each thin terminal is trying to locate dedicated VM to initiate the RDP session, but as long as each Windows VM can be identified by means of *unique and permanently assigned* hostname (in opposite to non-permanent IP address), the configuration of thin terminals becomes a straightforward task of binding particular hostname to particular thin terminal. involved Windows VMs.

Thin Terminals

A variety of thin terminals exists nowadays, while per-unit price starts from $100. Depending on the manufacturer, thin terminals could have different support in terms of implemented desktop delivery solutions. The majority of them *do* support Microsoft RDP protocol. Particularly, those terminals which are used in the virtual infrastructure under

consideration, i.e. Wyse V10L, operate under control of Wyse ThinOS operating system making them ready to be used in conjunction with Citrix XenDesktop software or any other desktop delivery solution relying either on Citrix ICA or on Microsoft RDP protocol. Such model was selected due to several reasons: it doesn't have any moving parts (HDD), it has 3-years repair warranty, and one of the mostly important aspect - it is ready to be used in the virtual infrastructure almost out-of-the-box (a single configuration parameter needs to be updated)

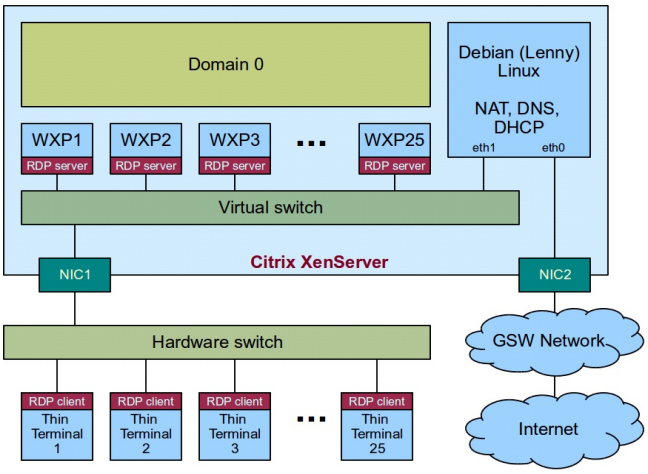


Figure 1. Network connectivity diagram

Automated Deployment of Windows XP VMs

It was mentioned earlier that each Windows VM is considered to have a unique *hostname*. It doesn't appear to be a problem when there exist limited number of VMs, say up to 5, but when the size of the infrastructure turns out into more than 20 hosts, the process of automated hostname assignment becomes a prioritized task. The following paragraph briefly explains technique, which is currently employed for 10 Windows XP VMs, reinstating from a pre-configured template. It is important to note that the number of VMs is limited only by hardware resources of the physical server used to host the hypervisor. The number 10 was selected, while 30 was estimated as the top boundary, because of the limited number of sits in the classroom, used for the project.

At first, the Windows XP template with all available security updates and required software is getting prepared. It is possible to create more than one template to allow relatively fast switching of the virtual infrastructure context. On top of the mentioned software, the template needs to be equipped with a set of Visual Basic (VB) script used for the automated hostname assignment. Responsibilities of such a VB script include the following: at the time of the first boot of a VM, instantiated from the template, read a *predefined file*, containing *desired hostname*, update the Windows registry to honor such a hostname, shut down the VM. Having this mechanism introduced the only question left is *how to create the mentioned file and make it readable by the VM once the instantiation took place?* To address the question a shell script, running on the Dom0 [1] virtual

machine needs to be introduced. The Dom0 VM appears to be a unique virtual machine which is also hosted by the hypervisor, but, in opposite to all other VMs, it is *automatically created* during the hypervisor installation process. From the system administrator's perspective it appears as yet another Linux box allowing to control all other VMs via command line interface. The shell script, executed by Dom0 needs to instantiate a Windows VM from the template, plug in a *virtual hard drive* which appears as a *removable storage* from inside the VM, create a file containing unique hostname on this virtual hard drive, power on the VM and wait for it to shut down in accordance to the algorithm implemented in the introduced VB script. This procedure can be repeated as many times as the desired number of Windows VMs to be deployed. Finally all VMs are about to be started up and become ready for virtual desktop experience to end users working on thin terminals.

The proposed solution is built for the Windows XP operating system which is going to lose support from Microsoft in several years. The following section presents tremendous results, achieved by the group of senior students, working on the migration from Windows XP to Windows 7 operating system in the scope of their Capstone Project Course.

CASE STUDY: UPGRADE FROM WINDOWS XP TO WINDOWS 7

The major problem associated with the movement of the current virtual lab infrastructure from Windows XP to Windows 7 is resulted from the dramatic changes made by Microsoft to its flagship OS - Windows 7 behaves in a much different way than Windows XP does. One of the issues that had to be solved was the new method of deployment. Windows XP did not as heavily rely on the unique G/UUID (Global/Universally Unique Identifier) that was generated at the install of the OS; therefore, it was relatively simple to create one copy of the OS, set it as a Citrix Template, and then replicate them on the XenServer, just activating them with the new key. Windows 7, on the hand, required a unique GUID in order to be activated against a Key Management Server (KMS) running on Windows Server that would activate it against the VLK (Volume License Key) given to academic institutions. If the GUID wasn't reset before an attempted join, the system would attempt and fail to activate against a KMS server that saw identical GUID's on its network.

The next problem, in a way related to the first, was the massive shift in deployment technologies used by Microsoft. Before Microsoft leveraged their Hyper-V virtualization products, templating was allowed to a blind-eye extent. Now templating is explicitly frowned upon by Microsoft, and the only recommended options are full image deployment or installations, both of which have the downfall of requiring the OOBE (Out Of Box Experience) to run. OOBE is the setup that runs when a new system is turned on, gathering the user's name, desired username, password, desktop background, etc. Most of this is controlled by Sysprep, a function of the OS that allows the setup to be automated via an xml script. Another hurdle we came across was the need for modularity in the system. At times these VM's would be used for research reasons by the student. We needed the ability to kill a corrupted VM and redeploy it rapidly with minimal or no loss of data. This was going to require centralizing data storage and user profiles so that the

OS itself was independent of needed data. Finally, the last problem to overcome was efficiency in the methods. We needed to make sure that network and disk access were kept to a minimum. Also RAM, at its high cost, needed to be used as efficiently as possible. Wasted RAM, or worse maxed out RAM, would create a slow lab that would possibly start to crash VM's. Things from data storage, VM startup, VM idle, to network usage all needed to be tested and prioritized so that the limited and expensive resources would be used to their fullest extent. This will include a short conversation on the different combinations of DNS and services offered via the Debian Lenny gateway, versus what can be run on Windows Server 2008.

The Solution

The first explored solution was Microsoft's recommendation of WDS, or Windows Deployment Services. This method requires setting up a Windows 7 setup with a sysprep script, and then deploying that collected image through PXE booting. The downfall of this method was the heavy amount of user interaction required to manage the install process, and worse yet, the fact that it was a full install. On a test rack of 3 IBM x336 servers, a deployment of 30 Windows 7 systems fully automated (after the laborious task of starting the install process for each one) took well over 9 hours of 100% server usage. The main bottle neck was the job being spread over 6 146GB 15k RPM hard drives. If server density was increased over more high performance drives, the speed increases in accordance. This would result, however, in wasted server overhead when the VM's were not being setup all at once. Therefore, with further research, it was found that Windows 7 could be made to generate a new GUID when the name of the system was changed during Audit Mode (sysprep OOBE stage). The sysprep script was changed so that several custom written *powershell scripts* would execute at the proper times to first un-restrict powershell's usage for the default administrator account, then rename the computer, restart the computer, and re-restrict powershell for security. On top of this, XenServer has software, known as XenTools that can be installed into the virtual OS. With these tools, the name of the VM can be called into the OS and output as plain text [7]. Calling this and piping it into the renaming powershell made it simple to rename the Windows 7 system after the name of freshly deployed VM. On this note, if a VM was `W7_VMcw21406` (A Windows 7 installation, virtualized in Crawford Wheatly Hall, Room 214 computer 6), by calling XenTools in a batch script during sysprep.

```
# Save the VM name into a file
cmd.exe C:\Program Files\Citrix\Xen\xenstore_client.exe read name >
name.txt
# Read the content of the file into a variable in powershell
$Name = Get-Content name.txt
# Rename the computer
(gwmi win32_ComputerSystem).Rename($Name)
# Restart the computer, allowing Windows 7 re-generate its GUID off
of the new name
Restart-Computer
```

The system will have renamed itself according to the VM name, and on reboot the GUID will be re-created. The system is now unique and sysprep can finish its job. This solution allows the avoidance of from scratch installs. Instead, a Windows 7 VM must be

manually instantiated once from the general template having all the mentioned scripts installed, got customized by means of installing applications, backgrounds, extras, etc and finally got converted back to the template. Such a template is used to instantiate all required Windows 7 VMs followed by activating the renaming script and generating new GUID after reboot.

To solve the issue of modularity, it was decided that domains (Windows Server Active Directory) would be utilized. With this we could give ever student a unique account with a roaming profile (an initiative already begun at the university to synchronize the accounts across labs). With the roaming profile on the server, students could save their school files in a share controlled by Windows Server Distributed File System (DFS). With the students having their own private storage, files can easily be moved along with the profile to any current Windows 7 system, be it a VM or a physical computer in the domain running Windows 7. This meant that VM's could be created, destroyed, and modified to the needs of the clientele without having to change login procedures. It also means students can take their specific permissions that they need to whatever system they need to use them on. This was nothing new compared to a physical environment, but one of the benefits brought to the system will be discussed in the efficiency section.

Finally, the problems of efficiency come to play. Disk access is the most precious resource of the setup. As everyone has seen their OS run off the hard drive, most are aware of how accessing programs causes disk access. But what happens when this is escalated to 30-50 systems running from one system? Disk access goes through the roof and the system slows to a crawl as the mechanical hard drives try to do work for 30 OS's at once. The main way of fixing this is making sure caching is heavy on the server side, so that most of the OS is laying in memory. Next is a technology known as de-duplication. It allows like blocks of memory and disk access (such as the same program installed on all the W7 systems) to be cached together. With this joint cache, one copy of the file exists in high speed memory and is spread amongst the systems, keeping them from all hitting the hard drive at the same time for what equates to identical files. On the network side, almost all communications happens between VM's, and doesn't load down a physical network. Because of this more high speed RAM and processor bear the wait of communications, instead of having to build a network to support it. The highest usage will be if the Roaming Profile storing Windows Server is outsourced away from a VM to a physical box. The physical box will need a fairly potent network tie to the VM server in order to support that communication of profiles.

CONCLUSIONS

The paper covers yet another approach to virtualization, which appears to be a cost-efficient alternative to the majority of existing commercial solutions. It is different in the subject of virtual desktop delivery: the commercial implementations rely on Microsoft Terminal Service, requiring per concurrent user licensing, while the proposed solution virtualizes individual Windows workstations, whose desktops are getting delivered to thin terminals by means of RDP protocol. Two generations of Windows OS, i.e. Windows XP and Windows 7 have been considered against proposed methodology.

A template based deployment process, allowing for unique hostname (both cases) and GUID (latter case) assignment at the time of the first boot, was used.

REFERENCES

[1] Hess, K., Newman, A., Practical Virtualization Solutions: Virtualization from the Trenches, Boston, MA: Prentice Hall, 2009.

[2] Windows Server 2008 R2 Pricing, http://www.microsoft.com/windowsserver2008/en/us/pricing.aspx, retrieved April 8th, 2011.

[3] RFC 1631: The IP Network Address Translator (NAT), 1994, http://www.ietf.org/rfc/rfc1631.txt, retrieved April 8th, 2011.

[4] RFC 1034: Domain Names - Concepts and Facilities, 1987, http://www.ietf.org/rfc/rfc1034.txt, retrieved April 8th, 2011.

[5] RFC 2131: Dynamic Host Configuration Protocol (DHCP), 1997, http://www.ietf.org/rfc/rfc2131.txt, retrieved April 8th, 2011.

[6] Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification, http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-RDPBCGR].pdf, retrieved April 8th, 2011.

[7] Citrix XenServer API Documentation, http://support.citrix.com/article/CTX127586, retrieved April 8th, 2011.

DEVELOPING MECHANISMS FOR SUPPORTING FACULTY USE OF CLASSROOM TECHNOLOGY*

TUTORIAL PRESENTATION

Bob Harbort, Ph.D., P.E.
Professor of Computer Science
School of Computing and Software Engineering
Southern Polytechnic State University
J-330 -- 1100 S. Marietta Pkwy.
Marietta, GA 30060
678.915.7405
bharbort@spsu.edu

David Edwin Stone
Director of Educational Support and Online Learning
Office of Faculty Support and Development
Southern Polytechnic State University
dstone@spsu.edu

Greg Scott
Laboratory Manager
School of Computing and Software Engineering
Southern Polytechnic State University
gscott3@spsu.edu

ABSTRACT

This workshop focuses on the ways faculty use classroom technology in teaching every day. We examine the entire spectrum of activities, from preparation for class through post-class processing activities such as posting lecture notes or captured class sessions to the Web. Most smaller schools lack budget and personnel to support these activities as comprehensively as large universities do.

The purpose of this workshop is to examine creative solutions for providing support for faculty in their classroom-related activities. We have identified a number of approaches, and will engage workshop participants in a brainstorming session to share their ideas and experiences.

INTENDED AUDIENCE

Post-secondary CS educators, support personnel, and departmental administrators in departments making significant use of technology in classroom settings.

AGENDA

Topic one:

Analysis of the typical range of pre-class, classroom, and post-class technology-related activities found in a variety of college and university settings. We will give examples from our experience, and make a comprehensive list that will include input from workshop participants.

Topic two:

Discussion of possible configurations of support for teaching-related faculty activities. This will include such things as shortcuts for configuring display computers for specific class needs, use of student volunteers for classroom setup and calibration, and pre-planning for low-budget tech support to be used when things go wrong during a class.

Topic three:

Open discussion among workshop participants to aid in mutual evaluation of various scenarios for classroom technology support.

Topic four:

Wrap up and discussion, approximately 15 minutes.

Prospective participants should come prepared to discuss the following things regarding your institution:

- typical classroom technology configurations -- computers, projectors, cameras, other equipment, software
- current support strategies -- centralized versus decentralized, hours of availability
- level of satisfaction with current support -- success stories, problem areas

PRESENTER BIOGRAPHIES

Bob Harbort is a professor in the School of Computing and Software Engineering at Southern Polytechnic State University in Marietta, Georgia. A licensed professional engineer, he also holds certifications in data processing and computer systems. His PhD is from the Graduate Institute of the Liberal Arts at Emory University, where he studied theories of interpretation and how they may be applied to both human and machine information processing systems.

David Edwin Stone is Director of Educational Support and Online Learning in the Office of Faculty Support and Development at Southern Polytechnic State University in Marietta, Georgia. He has a BSCS from SPSU, an MS in Instructional Technology from Georgia State University, and is finishing his PhD in Instructional Technology from GSU.

Greg Scott is the Laboratory Manager for the School of Computing and Software Engineering at Southern Polytechnic State University in Marietta, Georgia. He was previously the computer support manager for the Department of Continuing Education in the Extended University at SPSU.

SCRIPTING REPETITIVE RESEARCH TASKS*

Keith R. Nelms
Walker School of Business
Piedmont College
P.O. Box 10
Demorest, GA 30535
706-778-3000 x1289
knelms@piedmont.edu

ABSTRACT

Cloud computing services and web-based learning management systems place an abundance of useful data "at web's length." Information is available via web browser and can sometimes be downloaded as text or spreadsheet files, but not always in a format amenable for research and analysis. This paper describes Macintosh OS X scripting techniques developed for harvesting large volumes of information from web-based resources not able to provide data in useable formats. In addition to minimizing opportunities for human error due to tedious, repetitive manual processing, these techniques open opportunities for research and instructional activities not normally viable for faculty in small college settings and with limited resources.

PROBLEM STATEMENT

The scripting techniques described below grew out of projects requiring repetitive data downloading from web-based sources. In both projects, the web-based systems involved could not provide data in a readily usable format and would require many hours of tedious manual data manipulation.

- For a management information systems course, the author wishes to show publication trends for technology-related terms (e.g., "application service provider," "cloud computing," "virtualization") over a number of years. Terms must be searched in bibliographic databases over consecutive small units of time (e.g., monthly over 15 years). Each term searched would require manually entering 180

Boolean search phrases then require manually entering each search result into a spreadsheet for analysis and graphing. The project requires searching dozens of terms.

- In an introductory computer skills and concepts course, the author wishes to harvest detailed student performance data from a web-based instructional simulation program and from a separate web-based learning management system then combine the data into one structured dataset for assessing patterns of student performance. If completed manually, 15,000 individual data items would have to be downloaded and appropriately entered into a spreadsheet.

Historically, such data collection and entry tasks have been the Augean labors of graduate research assistants. With budget cutbacks and concerns over student privacy, however, graduate assistants are often neither available nor appropriate.

Fortunately, with a modest investment of time and minor investment of treasure, computer-based scripting tools can automate repetitive data collection and processing tasks - even tasks that require manipulating the computer operating system, operating system graphical user interface controls, multiple application programs, and data displayed only in webpages. Although custom-developed scripts for automating research tasks are, by definition, highly specific to the individual processing task, general strategies are developed below to guide script development.

SCRIPTING ALTERNATIVES

Numerous scripting languages are available to the personal computer user. Many common computer programs have an internal scripting or macro language that can manipulate information within the program (macro or scripting languages in various versions of Microsoft Office products, for example). A programming solution that automates the tasks described above, however, must control operating system functions (launching programs, switching programs, controlling graphical user interface elements), complete and submit web-based forms within a web browser, and manipulate a spreadsheet program. Such broad control is beyond the capability of an individual application program's embedded scripting language. While there are many scripting languages that can manipulate the operating system, not all of these can control the internal workings of individual application programs such as a web browser or spreadsheet.

The Apple Macintosh OS X operating system currently has two native scripting systems that can be deployed to address these automation needs. These scripting languages are part of the operating system and are thus available to any Macintosh user at no additional cost.

- **AppleScript**. Macintosh operating systems since 1993 have included AppleScript [1]. AppleScript is an outgrowth of the HyperTalk programming language embedded in Apple's popular HyperCard multimedia environment of the late 1980s [5]. Despite marketing materials touting AppleScript's easy-to-understand English-like commands, AppleScript is a verbose, complex language (see Figure 1) that can

challenge experienced programmers [6]. Its syntax is easier to read and understand than it is to write.

- **Automator** first appeared with Mac OS X 10.4 (Tiger) in 2005 [2]. Automator empowers non-technical computer users with the ability to automate some repetitive tasks. It features a point-and-click graphical interface (see Figure 1) that allows users to build "workflows" by dragging graphical tiles and setting a limited number of options in each tile [5]. Automator's "Watch Me Do" mode creates workflows by recording user actions, greatly simplifying some automation processes.

Both AppleScript and Automator are able to manipulate the operating system (to open and close files and programs, save files, manipulate operating system GUI elements, etc.) and application programs (like word processors, spreadsheets, web browsers, graphics programs, etc.). Automator actually creates AppleScript code internally; Automator workflows can be translated into AppleScript code. [5]

Figure 1 is a screen capture of two computer windows. The left-side window shows the AppleScript interface. AppleScript is a traditional text-based scripting language with structured syntax. The right-side window shows the Automator interface. Automator workflows are created by drag-and-dropping a limited set of pre-defined command tiles. Note the sixth line of code in the AppleScript window is running "TermTimePeerInstance," which is the Automator workflow shown in the right side window. As illustrated in the fourth tile displayed in the Automator window, some Automator tiles can be expanded to show parameters and events.

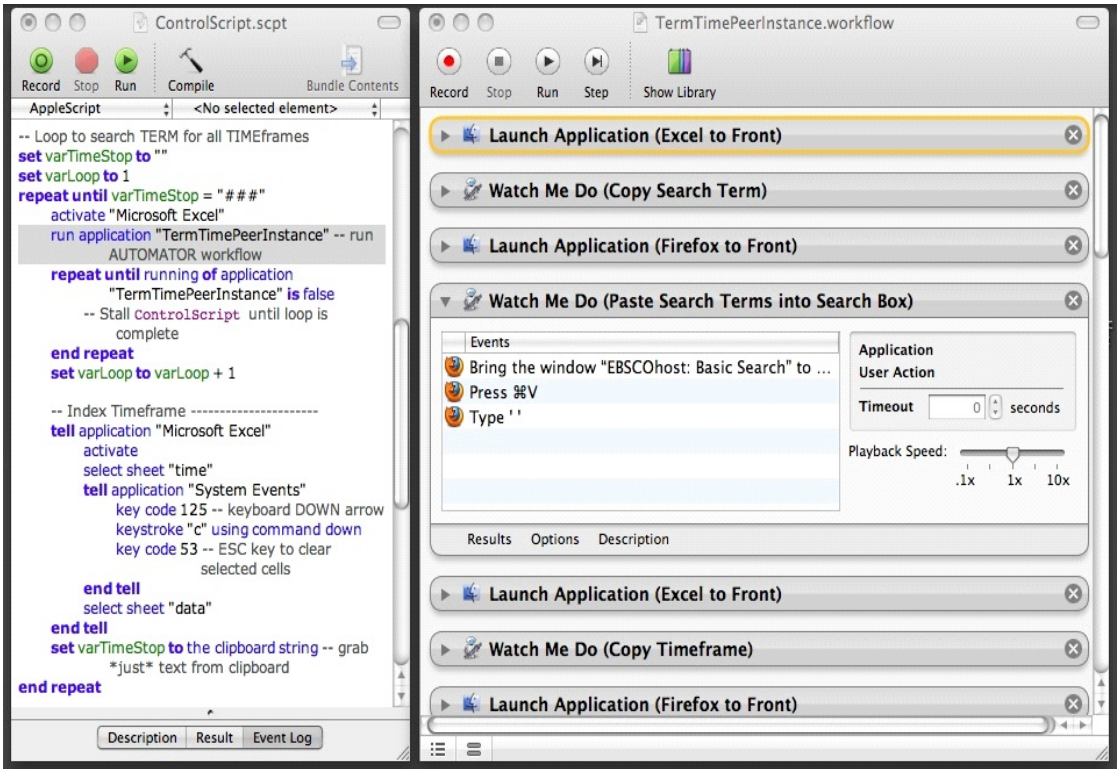


Figure 1. AppleScript and Automator Interfaces

To understand how AppleScript and Automator work, it is necessary to know how program automation takes place. Consider the task of closing an application program window, like a webpage or a spreadsheet. There are three ways a script could accomplish this task:

- **Script command**. The scripting language could tell the application program to close the window. The only visible sign of this to the end user would be the disappearance of the designated window. This approach only works when the appropriate scriptable command is built into the application program.
- **Keyboard equivalent**. The scripting language could also send the "close window" keyboard command through the operating system to the program. To the program, it would appear the user had typed the close window keyboard equivalent (command-W in Apple OS X). Again, the user would only see the window disappear.
- **User interface script**. The scripting language might manipulate the pointer on screen so it selects "Close" on the program's File menu. The user would see a "ghost in the machine" moving the mouse and manipulating the computer in a step-by-step manner.

All three of these techniques accomplish the same result. However, coding for each is different. Many tasks cannot be accomplished with all three techniques. For example, some menu items do not have keyboard or scriptable command equivalents; many mouse-and-pointer interface manipulations (like selecting a field in a web form) do not have scriptable commands or keyboard equivalents. For some tasks, it is far easier to implement one technique over another.

All application programs are not equally scriptable. Some application programs have many internal commands that can be manipulate directly with script commands. Microsoft Excel, for example, has almost 600 commands that can be manipulated directly in AppleScript [3]. Other programs have very limited script command support. Firefox, for example, has only 26 scriptable commands.

THE RIGHT TOOL FOR THE RIGHT JOB

Automator is designed to make automation accessible to the non-technical user. Some tasks might be automated with Automator alone. More complex tasks, however, require the sophistication of AppleScript. Since AppleScript can invoke Automator scripts, it is possible - indeed advantageous - to use the two scripting systems together. The challenge is to choose the right tool for each part of the automation task.

If script or keyboard commands are not available for a task, then programmatically simulating mouse movements and clicks may be the only practical automation solution. This is often necessary when retrieving information from web-based interfaces to commercial databases, which typically do not provide adequate documentation for scripting searches via the URL. Although it is technically possible for AppleScript to manipulate the interface, Automator's "Watch Me Do" command tile is a much simpler and faster approach.

	AppleScript	Automator
Strengths	• Powerful programming capabilities • Sophisticated loops and control structures	• Relatively simple programming • "Watch Me Do" records interface interface actions
Weaknesses	• Complex syntax • Difficult to program user interface actions	• Limited functionality • Limited looping • No branching or control structures

Figure 2. Critical Differences between AppleScript and Automator

Automator has two significant limitations that will push researchers into using the more complex AppleScript:

- **Looping limitations**. By definition, automating repetitive research processes means creating programmatic loops that can repeat the same process many times. Beginning with the Mac OS X 10.5 (Leopard) version, Automator can perform simple loops [4]. However, it can only loop a predetermined number of iterations; Automator cannot detect the number of iterations needed then stop when done. If the automation task requires that level of loop control, then AppleScript is required.
- **Control structures**. Similarly, Automator lacks control capabilities such as IF-THEN-ELSE commands. There is no way in Automator to specify conditional action. Again, AppleScript is required.

As noted above, AppleScript and Automator can communicate with one another, providing ways to productively combine Automator's simplicity and user interface capabilities with AppleScript's power and sophistication.

Figure 3 shows how the two scripting environments were deployed to conduct automated research with a web-based bibliographic database. Prior to triggering the automation, the list of keywords to be searched is placed in an Excel worksheet. The Boolean search phrase for each time period of interest is placed in another worksheet. The bibliographic database is launched in a web browser and the database's search textbox is shown. The AppleScript controlling script is then launched. One-by-one, the script selects a search term and searches each time period for that term. The number of "hits" for each search is copied from the web browser back to an Excel spreadsheet. Due to the complexity of the database vendor's website, the script must switch to the webpage source code to locate the number of hits and copy them. Necessary hygiene steps (like clearing the search box between searches) are performed. The automation continues until all terms are search for all specified time periods. The spreadsheet contains the number of hits for each term for each time period, allowing the researcher to compare trends across time. In Figure 3, icons indicate which scripting tool was used in each step.

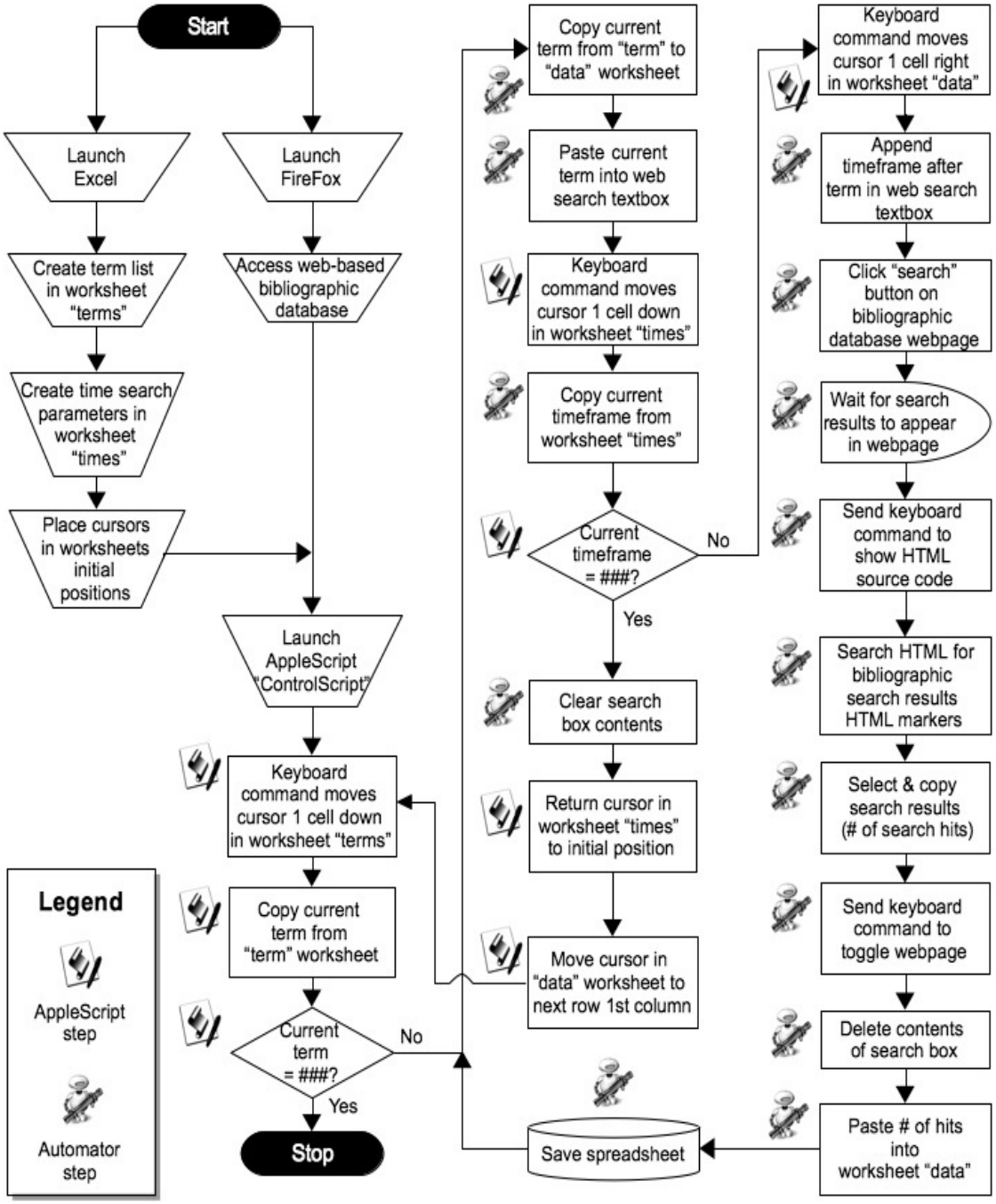


Figure 3. Flowchart Showing Combined Use of AppleScript and Automator

RECOMMENDATIONS

In evaluating options for automating research tasks, it is important to remember that automation is a means to an end, not the end itself. The goal is not to create a totally automated process but to speed processing of research data and reduce human error through automation. It is perfectly appropriate to mix automation and manual processing if that strategy is most productive.

In all cases, Automator should be the first tool explored. If significant automation gains can be obtained with just Automator, then it may be the only tool needed. Adding AppleScript to the project increases the complexity of the programming task and might be beyond the technical skills of some researchers who could effectively develop an Automator-only solution. It should also be noted that Apple continues to develop Automator with each release of their operating system. Future versions of Automator may be less limited in looping and control. Anytime Automator is insufficient for an automation task, however, AppleScript will likely fill the need. As illustrated in Figure 3, a working solution will likely consist of Automator workflows under the control of an AppleScript master script.

REFERENCES

[1] Apple ships first common script language entry, *PC Week*, 10(14), 13, 1993.

[2] Cline, C. Riding the Tiger, *The Seybold Report*, 5(4), 15-17, 2005.

[3] *Excel 2004 AppleScript Reference*, Redmond, WA: Microsoft Corporation, 2004.

[4] Langer, M., *Visual Quickstart Guide: Mac OS X 10.5 Leopard*, Berkeley, CA: PeachPit Press, 2008.

[5] Myer, T., *Apple Automator with AppleScript Bible*, Indianapolis, IN: Wiley Publishing, Inc., 2010.

[6] Sanderson, H., Rosenthal, H., *Learn AppleScript, 3rd Edition*, New York, NY: Apress, 2010.

COMPARISON OF SATISFACTION AND SUCCESS OF TRADITIONAL AND ONLINE STUDENTS IN AN INTRODUCTORY COMPUTER LITERACY COURSE IN A SMALL LIBERAL ARTS UNIVERSITY*

Gail Miles
School of Mathematics and Computer Science
Lenoir-Rhyne University
Hickory, NC 28603
828 328-3268
milesg@lr.edu

ABSTRACT

The purpose of this research study was to compare the satisfaction and success of two student groups: a traditional face-to-face (f2f) computer programming class and an online computer programming class. In spring, 2010, two sections of an introductory general education computer course were taught: one as a traditional lab class and the other totally online. The only difference between the two classes was the pedagogical delivery. Using t-tests, the results showed no significant difference in the satisfaction level of the two groups, but a significant difference in the grades –online students scored significantly lower than students in the traditional classes.

INTRODUCTION

In the past ten years, online learning in higher education has seen significant changes. Very few institutions offered online courses or curriculum before 2000. Instruction in early online education was considered by most to be substandard. These institutions were often assessed lower in terms of reputation and quality. This is no longer

---

the case. Most institutions offer as least some of their courses in an online format. Assessment of these courses and programs has become critical in order to standardize and guarantee a high level of quality.

Significant research has been published on the impact of online course delivery to student learning. A recurring problem in many of the studies is the inability to extract the pedagogy variable to be assessed in the comparison. These comparison studies have had mixed results. Part of the reason is the inability to control extraneous variables. MacGregor [1] published one of the first studies. He found little difference between online and f2f classes in terms of their perceived learning, but reported a significant difference in their perception of the amount of work they were expected to do and the comfort level with the technology. The structure of the research study was good but lacked homogeneity in the courses chosen for comparison. The classes in the study were in different disciplines using very different online and face-to-face(f2f) approaches to teaching, but the study is important as a rubric for comparing these two types of pedagogy and laid a foundation for future studies.

Cooper [2] specifically studied students in computer applications classes. Her findings analyzed the differences between traditional and non-traditional students in both online and f2f classes. Her results supported MacGregor's findings that students did not think they learned as much in an online class as with an instructor. At the time of her study, most students did not see online courses replacing traditional classes, but today most institutions of higher education have many different delivery choices. These two studies were chosen to highlight the evolution of student perceptions in the last 10 years.

A later study done by Shelley, Swartz, and Cole [3] compared two courses in business law, one f2f and one online. All aspects of the class were the same except the delivery. They found no significant difference in student success between the online and f2f courses. Student satisfaction was not assessed.

The goals of any research comparison of traditional and online courses are to determine if students, in fact, do learn as well in the online format as the f2f classes, and are they satisfied with their experience.

Purpose of the Study

The purpose of this research study was to compare the satisfaction and success of two student groups: a traditional face-to-face (f2f) computer programming class and an online computer programming class. In spring, 2010, two sections of an introductory general education computer course were taught: one as a traditional lab class and the other totally online. The same professor taught both sections. All assignments and schedules were identical. The submission of work was also the same. The only difference was the pedagogical delivery.

Environment

The study was conducted at a small, private, liberal arts institution with a student body of about 1900 students. The Math and CS department has approximately 50 majors

in CS, IT, and Math. The department is also responsible for teaching the required general education computing courses (CSC) for the institution. Students can satisfy this requirement with a number of different courses ranging from introduction to computing concepts to programming with Java, depending on their major requirements. There are, however, a significant number of majors who do not specify a required course. For these majors, the department offers a course for general education credit, CSC 115. This course has changed over the years, driven by technological changes and student needs. The configuration of the course at the time of this study was an 8week course using the Alice programming language to build simple algorithmic programs. In a given year, about 250 students take this course.

The trend toward online learning is now affecting traditional institutions. There has been significant resistance toward offering courses in the online format at the university. Three years ago, a task force was created by the Provost to study the feasibility of online courses and to develop rubrics for implementation. These rubrics for class approval of online courses were accepted by the curriculum committee, and online courses were offered for the first time in the summer session of that year. Online courses are new at the university, which is, historically, a traditional residential liberal arts institution. Online courses are slowly being created at the institution, and the faculty members are steadily building a number of online courses. Two of the general education CSC courses were the first to have an online option.

METHODOLOGY

Overview of the course

About 60% of the students at the university choose CSC 115 to meet the core requirement. By the end of this course, students are able to develop appropriate algorithms, develop program code that is logical and efficient, and develop simple 3D programs modeling the real world. The Alice programming language was used to meet these objectives. Students were exposed to ethical issues relating to computer technology and software. Discussion Question requirements were used to meet this objective.

Course Design

To meet the learning objectives, several assessments were used. Both online and f2f students were assigned weekly reading assignments, participated in weekly online discussions (for both online and f2f), completed Alice projects, completed 3 quizzes and one final exam. A Sakai Learning Management System (LMS) was used by both courses to post the syllabus, assignments, discussion forums, and grades. The online course received all the instructions and course material online. The f2f class received instructions and course material in the classroom lab. They also completed most of their program projects during the lab class period. The instructor was available throughout the lab for questions.

Traditional Class. Students attended class two days a week for 8 weeks. The class met in a Computer Lab on Tuesdays and Thursdays for 50 minutes. The delivery system

was a combination of lecture and lab assignments. Exams were proctored in the class. There was a discussion component each week. A discussion question (DQ) was posted on the LMS at the beginning of each week. Students followed a published rubric in answering the DQ.

Online Class. Students received all their instruction and material online in the LMS environment. The delivery system consisted of weekly assignments, lectures, and videos posted to the LMS. Exams were proctored in a f2f environment. A discussion question was posted on the LMS each week. Students followed a published rubric in answering the DQ.

The LMS interface was set up with a user-friendly GUI where students could find any resource within one or two clicks. Both classes had the same GUI.

Basic guidelines for the online course included the following

- Timely Weekly Instructor Feedback was given to each student.
- Class started on Sunday night at Midnight to Sunday night at midnight. All assignments were required to be posted by the deadline.
- Discussion Question Rubrics were used to guarantee engaging discussions. (The same rubric was used for both the online and the f2f students).
- Quizzes and exams were proctored and f2f in some format.

Sample

Table 1 illustrates that the majority of students were females with the traditional class having about 37% males and 63% and the online class having about 21% males and 79% females.

	Males	Females
Face-to-face	7	12
Online	5	19

Table 1. Gender

Table 2 shows that most of the students in the f2f class were sophomores and juniors while most of the students in the online were a little older with the majority of juniors and seniors.

	Freshmen	Sophomore	Junior	Senior
Face-to-face	2	10	7	0
Online	2	4	10	8

Table 2. Year in college

Data collection procedures.

Instruments: Two instruments were used to collect satisfaction data – the End-of-Course Student Evaluations and a Satisfaction Survey collecting data on specific issues related to these courses. They were administered at the end of both courses. The

university's course evaluations were distributed and data anonymously collected using the university's Learning Management System and analyzed for both classes. The Satisfaction Survey was also anonymously administered to both groups (all students) via the LMS.

For the Student Evaluation, students answered 16 questions related to course satisfaction, instructor satisfaction, and learning satisfaction. Students were asked to rate their response to the questions as *Strongly Agree, Agree, Disagree,* and *Strongly Disagree*. Out of 19 students in the f2f class, 10 completed the survey. Out of 24 in the online class, 17 completed the survey.

In the Student Satisfaction Survey, students answered 13 questions related to course satisfaction, instructor satisfaction, and learning satisfaction. Students rated their response as *Agree* or *Disagree* based on their satisfaction level. The surveys were posted online for both classes. In the f2f class, 14 out of 19 completed the survey in the f2f class. . Seventeen out of 24 On-line students completed the survey.

RESULTS

Paired t-tests were used to assess significant differences in the student responses. The Student Evaluation data is represented in Table 3. For the online course, the mean was 2.9 out of a possible 4.0. The Standard Deviation was .3056. For the f2f class, the mean was 2.7. The Standard Deviation was .4256 (Table 3). T value is 1.3358 with 38 degrees of freedom. Although the mean for online classes was higher, it was not significant. With a confidence level set at 90%, the t-test produced 81.04% which is not significant.

	N	Mean	Standard Deviation
Online class	17	2.9	.3056
F2f class	10	2.7	.4256

Table 3. Results of the student evaluations.

A paired t-test was used to assess significant differences in the student responses of the Satisfaction Survey. For the online course, the mean was 1.84 out of a possible 2.0. The Standard Deviation was .13. For the f2f class, the mean was 1.82. The Standard Deviation was .11 (Table 4). T value was .4563 with 29 degrees of freedom. Although the mean for online classes was higher, it was not significant. With a confidence level set at 90%, the t-test produced 34.84% which is not significant.

	N	Mean	Standard Deviation
Online class	17	1.84	.13
F2f class	14	1.82	.11

Table 4. Results of the Student Satisfaction Survey.

Student success was assessed using the final grades in the course. Table 5 shows the breakdown of the grades for both the online and the f2f courses. As can be seen in Table 5, students in the online class scored lower on every assessment. A paired t-test was used to assess significant differences in the students' final grades.

For the online course, the mean course grade was 79 out of a possible 100. The Standard Deviation was . 201502. For the f2f class, the mean was 86.22. The Standard Deviation was .2239 (Table 4). T value was 115.8486 with 45 degrees of freedom. With a confidence level set at 90%, the t-test produced 100% which is a significant difference between the two results.

Assessment	Online	F2F
Assignments and Projects (40% of the grade)	93	96
Discussion Questions (15% of the grade)	85	93
Quizzes (25% of the grade)	77	86
Final Exam (20% of the grade)	80	83
Final Course Grade Average	79	86

Table 5. Student Grades

	N	Mean	Standard Deviation
Online class	24	79	.201502
F2f class	19	86	.2239

Table 6. Results of the student satisfaction survey.

The grade results were a little surprising. The result which was expected to be lower with the online class was the assignments and projects because online students received no one-on-one help with the programming. Although a little lower, both results on this assessment were very close. Where the two diverge are with the Discussion Questions and the Quizzes. The online students did significantly worse than the traditional classes. This is even more surprising because the students in the online classes were more mature learners (juniors and seniors) as opposed to sophomores and juniors in the traditional class.

CONCLUSIONS

This study compared student learning and satisfaction in an introductory programming class for non-majors which was offered as both a traditional class and an online class. The goal of the research study was to assess whether students are meeting the learning outcomes of the course and to assess students' satisfaction based on the method of delivery.

Factors which may explain the results include the following. Students taking these courses had little or no background in technology. The idea of programming a computer was new to them. None of the students had programmed in Alice or any other language. Online courses are new to the university, and as such, the students in the online class, did not fully understand what it takes to be successful as an online student. Instructional and motivational videos were included in the course but were not universally viewed.

The findings of this study support other research which measured satisfaction of online students. There is little difference to students' satisfaction, whether they take the course online or in a traditional classroom. There were no significant differences of student satisfaction for course structure and course material comparing the online students and traditional students. In this sample, students who were enrolled in the online course

were more likely to have lower grades than those students in the traditional class. This may be due to their inexperience in taking classes online since this format is new to most. Most of the students in the courses were also traditional age students which may have added to the motivational issues. One limitation of this study is the sample size. The institution has a low faculty-to-student ratio as part of its mission so class size is small. Further adding to the limited size is the fact that the evaluations and surveys were submitted online, and thus, there was not 100% participation. There is an expectation of further revisiting this in the future.

REFERENCES

[1] Cooper, L. A comparison of online and traditional computer applications classes. *T.H.E. Journal* 28 (8) 2001 WN: 0106000462013.

[2] MacGregor, C.J. A comparison of student perceptions in traditional and online classes. *Academic Exchange Quarterly*, 5, 143-148, 2001.

[3] Shelley, D, Swartz, L, and Cole, M. A comparative Analysis of Online and Traditional Undergraduate Business Law Class. *International Journal of Information and Communication Technology Education*, 3, Issue (1), 2007.

PLANNING FOR CO-EXISTENCE*

Marsha Zaidman, Gail Brooks
University of Mary Washington
Fredericksburg, VA 22401-5300
540-654-1319, 540-286-8021
marsha@umw.edu, gbrooks@umw.edu

ABSTRACT

Many institutions offer Management Information Systems (MIS) programs from their Schools of Business. Similarly, Computer Science Departments in the Schools of Liberal Arts and Sciences offer both Computer Information Systems (CIS) and the traditional Computer Science (CPSC) major. How similar are MIS and CIS? What distinguishes them? Can both programs be offered without robbing majors from each other? Two instructors at The University of Mary Washington, one in Computer Science and the other in Management Information Systems, chose to explore these questions within the context of two existing courses. One course is offered by the computer science department as part of the computer information systems program that has existed for four years. The other course is offered by the College of Business as part of the business administration major, which has existed for many years. Both courses are required in their corresponding major. This paper summarizes the comparison of BUAD 152 Management Information Systems from the business administration major and CPSC 310 Computer Information Systems from the computer information systems major.

INTRODUCTION

Traditional Computer Science (CPSC), Computer Information Systems (CIS), and Management Information Systems (MIS) are three related but distinct areas of study in the undergraduate curriculum. Supporting these related but distinct areas is challenging for smaller institutions. Typically, these multiple tracks are supported by overlapping courses between the computer science and business departments [6]. Because

these courses are primarily designed to serve different audiences, integrating them presents difficult challenges. Unlike courses designed to be studies in a particular progression within a single discipline, these courses have mixed audiences and students may complete them in varying sequences. The authors realized that the boundaries of two particular courses, CPSC 310--Computer Information Systems and BUAD 152 Management Information Systems had become blurry, so they chose to engage in a dialog that would improve both courses. Our approach was: compare the status of these courses with each other; compare the contents of these courses with the recommendations of the Association for Information Systems (AIS) and the Association of Computing Machinery (ACM); and reduce the overlap between the two. The challenge was still to provide necessary content for both an MIS course within the School of Business and a CIS course within the Computer Science Department of the School of Liberal Arts and Sciences. This paper will compare and contrast the two courses, one taught in a computer science department and the other taught in a business department.

ROLE OF BUAD 152 VERSUS CPSC 310

BUAD 152 (Management Information Systems) was introduced in the early 1990s to provide instruction in personal productivity tools like word processing and spreadsheet applications. As noted in the AIS-ACM joint recommendations for curriculum guidelines, such a course is no longer appropriate and should be removed [8]. Fortunately at the University of Mary Washington, the contents of this course evolved to incorporate other topics and meet other objectives, see Table 1 below. This course serves our academic community as an introduction to MIS and as a pre-requisite to BUAD 353 – Decision Analysis. According to its catalog description, BUAD 353 introduces a variety of Management Science models for use in analysis of business problems. A computer software package provides the computational basics for case analysis of problems in linear programming, inventory, waiting lines, PERT/CPM, and simulation. CIS majors may select BUAD 353 to fulfill a requirement in their CIS program. Hence, CIS majors must complete BUAD 152 Management Information Systems as well as CPSC 310 Computer Information Systems.

To complicate matters further, traditional computer science majors may use CPSC 310 to fulfill a major elective. Such students are unlikely to enroll in BUAD 152 and BUAD 353. Additionally, Business majors are unlikely to enroll in CPSC 310 because of its pre-requisite requirement. Figure 1 illustrates the relationship of these courses to the different majors. Thus the challenge: to provide two distinct yet related courses with a minimum of overlap and yet be capable of standing alone in support of the business administration major, the computer science major and the computer information systems track students.

Figure 1: Relationship of These Courses to the Different Majors

LEARNING OBJECTIVES

As illustrated in Table 1, the two courses have different learning objectives and course descriptions. However, the topical content of these courses are very similar. The differences lie in the emphasis and focus. BUAD 152 approaches these topics from a business perspective and CPSC 310 emphasizes the computing aspects of these topics.

CONTENT

Emphasis in BUAD 152: Information Systems in the Digital Age, Global E-Business and Collaboration, Porter's Competitive Forces, Building Competitive Advantage with Information Systems, IT Infrastructure, Telecommunications, Emerging Technologies and Foundations of Business Intelligence: Spreadsheet Analysis and Database Applications.

Emphasis in CPSC 310: Information Systems in the Digital Age, Global E-Business and Collaboration, Achieving Competitive Advantage with Information Systems, Securing Information Systems, Risk Management, Continuity Of Operations Planning and Disaster Recovery planning, Software as a Service, E-commerce: Business and Technology, Computer Programming Projects as appropriate. Table 1 compares and contrasts several aspects of both courses.

BUAD 152	CPSC 310
Pre-requisite: NONE Credits: 3	Pre-requisite: CPSC 220¡XComp Sci I Credits: 4

Catalog Description: Management Information Systems-- The purpose of this course is to examine the technical, business and management aspects of management information systems through the study of MIS theory and concepts. Emphasis is placed on how and why different types of information systems have become an essential part of organizations. Students gain experience solving real world business problems using different information systems applications throughout the course.	**Catalog Description:** Computer Information Systems-- This course introduces the student to the use and implications of information technology in the business environment. This course covers such topics as data management, networks, analysis and design, computer hardware and software, decision support systems, database management systems, transaction processing systems, executive information systems, and expert systems. It also provides activity with computer based and non-computer-based problems/cases and includes real-world programming projects that are implemented using a high-level programming language.
Learning Objectives: • To learn how technology and information systems can be utilized by organizations • To gain technical skills necessary to analyze and use information systems to solve common business problems • To understand managerial strategies and methods related to information systems • To analyze decisions faced by information system professionals. • To enhance communication and critical thinking skills	**Learning Objectives:** • To understand the role of information systems in a business environment • To understand how information systems can improve decision making and support management of data and knowledge • To understand how to apply information systems to create a competitive advantage • To understand how the global marketplace affects the competitive forces of a business • To understand current computing issues in business like Cloud Computing, Information Security, and E-commerce • To practice the design and implementation of original Visual Basic Programs within the Visual Studio.NET 2010 environment • To improve oral communication skills through class presentations • To improve critical thinking and research skills

Technical Skills: Database and spreadsheet applications show students how information systems are used for making informed decisions and creating information that can help analyze business processes for efficiency and effectiveness. Students learn to design databases, create forms and queries, generate reports, use mathematical and conditional formulas for calculations, create visual representations of data, and execute different scenarios to assist in decision making.	**Technical Skills:** Visual Basic .NET Windows application programming assignments that involve the design and implementation of a GUI, windows form processing, event handling, and data processing. Students explore menus, toolbars, dialog boxes, exception handling, text files, string processing, objects, modules, and collections. Projects introduce the student to Database integration using the ADO.NET interface.
Student Audience: This course has no prerequisite and students from any major may take it.	**Student Audience:** Computer science and/or Computer Information Systems majors.

Table 1: Overview of BUAD 152 and CPSC 310 at the University of Mary Washington

These topics are appropriate and many of them correspond to suggested topics for a Foundations of Information Systems and a Computer Information Systems course [3,4,8]. Even though many of the general topic areas are the same for both courses, the concepts and applications that are emphasized in each area differ. For example, the MIS course emphasizes the business process and management perspectives of each topic and how decision makers could be supported by information system technologies [2]. The CIS course emphasizes the role that information systems play in a business environment and instructs students on programming development tools commonly found in professional environments.

Both courses emphasize oral and written communication skills, group work, critical thinking and problem solving throughout the majority of the course activities. Research by Becerra-Fernandez, Elam and Clemmons supports the current ways in which both of these courses are taught [2]. They suggest using technologies in MIS courses that could serve many business process areas while also providing technology skills that could be useful in other majors. Mansour and Reynolds' study of students' interest in IT subject areas supports many of topics and technology skills taught in both the MIS and CIS courses [7]. The study included students from many different majors and at all levels of undergraduate standing. This information supports the importance of the MIS course being offered as a freshman level course to students in any major. Gorman's case study indicates today's businesses need leaders with knowledge of business areas and technology skills [5]. Technical skills are taught in both courses to assist with analyzing and solving business problems.

Based on current research, the topics generally taught in these courses are beneficial to all three audiences (MIS, CIS, and traditional Computer Science). In the future, additional topics for CPSC 310 may be drawn from Introduction to Problem Solving with

ArcGIS, Information Assurance, Software as a Service, E-commerce: Building an E-Commerce Web Site, and Ethical and Social Issues in Information Systems.

COMMUNICATION SKILLS

Both courses provide opportunities for practicing written and oral communication skills. Students in CPSC 310 practice their written communication skills through the preparation of an annotated bibliography on an assigned company and three detailed outlines of oral classroom presentations. The three individual classroom presentations involve case studies of different topics related to the students' assigned companies. Students must prepare a detailed outlines of each presentation, a properly formatted bibliography of references, and a supporting electronic slide presentation (usually MS-PowerPoint).

Students in BUAD 152 are assigned written assignments to discuss selected MIS topics such as information security and privacy. Research references are required for each assignment. Students are asked to collaborate on different MIS topics and present the information to the class, both verbally and in presentation format.

High quality written and oral communication are mandatory skills in all professions. Khalid Aziz reports that 63% of company directors believe presentation skills are more important for career success than intelligence or financial aptitude [1]. Clear communication engenders client confidence and adds value to companies [1]. Students benefit from having resources like the Speaking Center and Writing Center to support the preparation of their papers and presentations as well as having opportunities to practice these skills in the classroom where they receive peer and instructor feedback.

CONCLUSION

These authors conclude that the CIS concentration within Computer Science and the MIS concentration within Business can peacefully co-exist. The content of both courses serve as a foundation for future course work for all audiences. Because of the different technical skills covered and the difference in emphasis, both authors agree that students would benefit from taking both courses, even if not required. According to Hoganson, there will be some overlap in courses offered in MIS and CIS, but he also suggests there may be opportunities to share courses and course content between the two areas [6]. These authors and instructors foresee more courses that can be shared in the future. Candidates for sharing include fundamental programming courses, database fundamentals, information assurance, and network security. These authors agree that continuing dialog is essential to the most effective use of university resources. In a parallel study, the authors looked more specifically at course assessment. The authors will be presenting these findings at CCSC Eastern, October 2011.

These authors look forward to continuing our dialog and collaboration. For sure, the ongoing conversation and relationship building was one of the most valuable outcomes of this investigation. Continuing dialog is also crucial to the minimization of overlap among the programs and the most advantageous use of institutional resources.

REFERENCES

[1] Aziz, Khalid. The Kay to Perfect Presentations. Industrial and Commercial Training, 30 (6), 214-217, 1998.

[2] Becerra-Fernandez, I., Elam, J., & Clemmons, S. Reversing the Landslide in ComputerRelated Degree Programs. Communications of the ACM , *53* (2), 127 – 133. February 2010.

[3] Computing Curricula 2005- The Overview Report. September 30, 2005.

[4] Ducrot, Joelle, Steven Miller and Paul S. Goodman. Learning outcomes for a business information systems undergraduate program. *Communications of the Association for Information Systems, 23* (6) 95-122, June 2008.

[5] Gorman, Michael. A Case Study in Effectively Bridging the Business Skills Gap for the Information Technology Professional. *Journal of Education for Business, 86,* 17-24, 2011

[6] Hoganson, K. Alternative Curriculum Models for integrating Computer Science and Information Systems Analysis, Recommendations, Pitfalls, Opportunities, Accreditations, and Trends. *Journal of Computing Sciences in College, 17* (2), 313-325, February 2001.

[7] Mansour, S. S., & Reynolds, J. Development of a Baccalaureate Major in Information Technology: Adding a Third Dimension to a Comprehensive Computing Program. SIGITE '09: Proceedings of the 10th ACM conference on SIG-information technology education, 108-114, Fairfax: ACM, 2009.

[8] Topi, H., Valacich, J. S., Wright, R. T., Kaiser, K., Nunamaker, J. J., Sipior, J. C., et al. IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. *Communications for the Association of Information Systems, 26* (18), 359-428, 2010.

HOW CAN WE REACH THE NON-CS MAJOR?*

PANEL DISCUSSION

Julia Benson-Slaughter, Perimeter College, Julia.Benson-Slaughter@gpc.edu
Susan G. Glenn, Gordon College, Barnesville, GA, sglenn@gdn.edu
Dee Medley, Augusta State University, Augusta, GA, dmedley@aug.edu
John Reece, Gordon College, Barnesville, GA, jreece@gdn.edu
Rebecca Rutherfoord, Southern Polytechnic State University, Marietta, GA, brutherf@spsu.edu

ABSTRACT

As the state of current technologies continue to change along with the demographics our students the challenge of reaching the non-CS major continues to be a compelling issue among college-level educators. As we strive to serve the needs of this population while also perhaps drawing some of the more interested students to the major, we must address certain questions:

- What topics, across the range of the computing discipline, does the non-major need to study?
- What are the most effective methods for introducing these topics to the class?
- Should we incorporate some of the latest gadgets such as tablet computers, or simple robots?
- To what extent should programming be discussed, especially the emerging field of parallel programming?
- Might there be extra-curricular activities that would prove beneficial to the non-major?

The panel will consist of a group of college professors ranging in specialty from electrical engineering to CS and IT education. The panel's participants come from colleges in rural two-year schools and universities located in metropolitan areas. The panel will conduct the presentation as a participatory one. The audience participation will allow new brainstorming ideas to be considered with ideas for future research.

CASE STUDIES FOR INTRODUCTON TO COMPUTATIONAL MODELING*

TUTORIAL PRESENTATION

Jose M. Garrido
Kennesaw State University
Kennesaw, GA 30144
jgarrido@kennesaw.edu

ABSTRACT

Computational thinking is an important component in today's computing education and in developing computational models. This tutorial provides a short overview of our approach to develop computational models at the pre-calculus level and presents a gentle introduction by developing computational models using several case studies. These are presented by applying computational thinking starting with a word problem, designing a conceptual model, a mathematical model, and finally a computational model.

The main software tools that are used for implementing the mathematical models are: Matlab and Octave. This material corresponds to the first course in computational modeling. Because this first course includes programming principles, this first course can replace a traditional first programming course (CS1), which is usually taught with a conventional programming language. More advanced courses on computational modeling include linear models that are formulated with sets of linear equations, linear optimization models that are formulated with linear inequations, and continuous models formulated with differential equations. These more advanced models are implemented with LP_solve and Scilab, in addition to Matlab and Octave, and will only be briefly discussed in this tutorial.

The case studies were designed and developed for teaching students of computing (Computer Science, Software Engineering, and related disciplines), mathematics, and the various sciences.

TUTORIAL OBJECTIVE:

To present the general concepts and techniques used for developing computational models via case studies. Provide a gentle introduction to the process of developing computational models. Part of the process is applying computational thinking. Discussion on implementing undergraduate courses in computing and to help strengthen the application areas of computer science and related disciplines.

LIST OF TOPICS:

(Topics marked (*) will only be briefly introduced due to time constraints)

1. Real-world systems and models. Introduction to computational models modeling.
2. The process of developing computational models. Computational thinking. Abstraction and decomposition. Conceptual model. Mathematical model. Algorithmic structures.
3. Introduction to basic Matlab and Octave commands. Difference equations. Functional equations.
4. Case study 1: Temperature conversion. Developing the computational model. Case study 2: computing area and circumference of a circle using algorithmic structures.
5. Case study 3: free falling object. Average and instantaneous rate of change of the vertical position and the velocity of the object.
6. Case study 4: model with arithmetic growth formulated with difference equations.
7. Using functional equations. Case study 5: model with quadratic growth.
8. Case study 6: model using a polynomial function.
9. Case study 7: Empirical model using curve fitting techniques.
10. Case study 8: Model with geometric growth.
11. Optional case studies used in second course in computational modeling: linear models formulated with systems of linear equations, linear optimization models (LP).

MATERIAL:

Handouts of the slides and instructions for the accessing the Web page with all the additional material on computational modeling with Matlab and Octave. These materials will be located on the following web site:

http://science.kennesaw.edu/~jgarrido/compmod

PRESENTER BIOGRAPHY

Jose M. Garrido is Professor of Computer Science, Department of Computer Science. He earned a Ph.D. in from George Mason University, an M.S. in Computer Science (George Mason University), an M.Sc. in Information Systems (University of London), and B.S. in Electrical Engineering (Universidad de Oriente).

His research and academic areas of interest are: Object-oriented modeling and simulation, multi-disciplinary computational modeling, formal specification of real-time

systems, specification and programming languages, performance modeling of operating systems. Dr. Garrido developed the Psim project. He has published several papers on these areas and has written a book on operating systems and three books on object oriented simulation.

FUNDING COMPUTING INSTRUCTION FROM WATER: COMBINING SWEAT EQUITY WITH OPEN SOURCE AND OTHER FREE TOOLS*

CONFERENCE WORKSHOP

Rich Halstead-Nussloch and Orlando Karam
School of Computing and Software Engineering
Southern Polytechnic State University
Marietta, GA 30060
678-915-5509, 678-915-5558
rhalstea@spsu.edu, okaram@spsu.edu

ABSTRACT

Over the past ten years, faculty at Southern Polytechnic State University (SPSU) have implemented instructional environments for computing using Free, Open Source and low-cost tools and resources. (Although not completely accurate, for convenience "Open Source" will be used here to refer to Free, Open Source and low-cost tools.) This has resulted in a set of initiatives to trade "sweat equity" for capital equity investments to provide quality computing instruction on our campus. The faculty members have started and maintain multiple instructional development activities, which are built on a foundation of Open Source technology platforms and in the sum are in alignment with SPSU's polytechnic nature. This 90 minute workshop will describe and explain what worked and what did not work, prescribe and recommend methods, tools and actions for effective implementation on participants' campuses, allow for questions and discussion, and provide tool recommendations and resources for participants to take back home.

INTRODUCTION

In today's funding and financial climate, universities and many other organizations are seeking less expensive and more cost-effective ways to provide instruction. Paying nothing for software and/or having a license for software with no restrictions are attractive in this environment. This ninety-minute workshop's purpose is to aid computing and education professionals in establishing and implementing campus initiatives to enhance cost-effective computing instruction within the university's community. The intended audience includes university faculty, administration and students with an interest in utilizing Open Source tools to facilitate computing instruction. The workshop requires no prerequisite knowledge from the CCSC attendee. Participants are recommended to bring a notebook computer to this workshop.

With respect to the workshop objectives, participants will be able to: 1) State and discuss the major concepts, tools and methods related to utilizing Open Source software, especially as related to a computing instructional environment. 2) Identify and pursue internal and potentially external funding and support for customizing and deploying Open Source software in a campus environment with an eye on synergism and partnering. 3) Take steps to engage students meaningfully in such an instructional environment. 4) Find opportunities to innovate appropriately and effectively within this instructional environment. 5) Plan for the coordination, governance and assessment of the utilization of Open Source on the campus.

METHODOLOGY

The workshop is designed to present short tutorials and vignettes that capture the experience with Open Source at SPSU. The intentions are to a) demonstrate the lessons learned about what to do and what not to do; b) raise discussion questions from the attendees; and c) provide resources and ideas for using Open Source on the participants' home campuses.

Planned Workshop Topics, Resources and Timing

Topic	**Description and Take-Away**	**Timing**
Introduction	Description: Introduce Open Source and how it aligns with SPSU's vision, mission, and guiding principles. Describe the polymorphic approach to campus instructional technology. Quick round-robin introductions of all participants to identify their goals and questions. Take-Away: A template charter for a campus' use of Open Source and a link to the workshop website for Open Source [1].	15 minutes

Funding	Description: Overview Open Source and the "leverage existing dollars" approach taken at SPSU to fund and provide resources for using Open Source software on campus. Provide for participant questions and discussion. Take-Away: A checklist of typical campus activities that are already funded with example descriptions and justifications that can be leveraged for a campus Open Source initiative	10 minutes
Open Source Contribution to Student Engagement and Innovation	Description: Present two or three vignettes of student engagement in and use of an Open Source instructional environment for computing and how it supports synergistic innovation. Overview the current use of Open Source and the reach of classes covered with these cost-effective environments as well as the innovations seen. Provide for participant questions and discussion. Take-Away: A checklist for typical campus activities that can be easily set up and implemented using Open Source tools	45 minutes
Coordination and Governance	Description: Present one or two vignettes of governance and coordination. Provide a list of the multiple issues and opportunities for coordination and governance (e.g., property and accountability) that have emerged so far in SPSU's use of Open Source and at other campuses. Overview our ideas for addressing issues and using opportunities. Provide for participant questions and discussion. Take-Away: A list of campus Open Source governance issues.	10 minutes
Wrap-up, Discussion, Q&A and Evaluation	Description: Keep a "parking lot" of questions and items as the workshop progresses. Show the scoreboard for where SPSU is on expanding cost-effective, quality computing instruction via Open Source. Quick round-robin of all participants for reactions to application on their campuses. Participants will also complete a short workshop evaluation. Take-Away: Recommended measurements, evaluation and assessment steps for a campus' use of Open Source.	10 minutes

REFERENCES

[1] Halstead-Nussloch, R. and Karam, O. *Toolkit for utilizing Open Source software on campus.* http://cse.spsu.edu/rhalstea/free, retrieved on 12 June 2011.

INDEX OF AUTHORS

JCSC Volume 27 Number 2 December 2011

Muhlenberg College
2400 Chew Street
Allentown, PA 18104-5586