

The Journal of Computing Sciences in Colleges



*The Consortium for Computing
Sciences in Colleges*

Volume 26, Number 5

May 2011

The Journal of Computing Sciences in Colleges

Papers of the Ninth Annual CCSC Mid-South Conference

**April 1-2, 2011
University of Central Arkansas
Conway, Arkansas**

Papers of the Seventeenth Annual CCSC Central Plains Conference

**April 8-9, 2011
University of Central Missouri
Warrensburg, Missouri**

**John Meinke, Editor
UMUC — Europe**

**George Benjamin, Associate Editor
Muhlenberg College**

**Susan T. Dean, Associate Editor
UMUC — Europe**

**Dan Brandon, Contributing Editor
Christian Brothers University**

**Dean Sanders, Contributing Editor
Northwest Missouri State University**

Volume 26, Number 5

May 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 Jim Aman, CCSC Membership Chair, Saint Xavier University, Chicago, IL 60655.

Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

TABLE OF CONTENTS

THE CONSORTIUM FOR COMPUTING SCIENCES IN COLLEGES BOARD OF DIRECTORS.....	ix
CCSC NATIONAL PARTNERS.	x
FOREWORD.....	x
John Meinke, UMUC – Europe	
PAPERS OF THE NINTH ANNUAL CCSC MID-SOUTH CONFERENCE.....	1
WELCOME TO THE 2011 CCSC MID-SOUTH CONFERENCE.....	2
Larry Morell	
2011 MID-SOUTH CONFERENCE COMMITTEE.	2
REGIONAL BOARD — CCSC MID-SOUTH REGION	3
REVIEWERS — 2011 CCSC MID-SOUTH CONFERENCE.....	4
AN INTRODUCTION TO OPENMP — PRE-CONFERENCE WORKSHOP.....	5
David Naugler, Southeast Missouri State University	
A WEB-BASED GRAPHICAL SET THEORY PROOF ASSISTANCE SYSTEM	6
Cong-Cong Xing, Nicholls State University	
EXPERIENCES IN COMPUTER SCIENCE WONDERLAND: A SUCCESS STORY WITH ALICE.	16
Jose Cordova, Virginia Eaton, Kimberly Taylor, University of Louisiana at Monroe	
MINIMIZING TO MAXIMIZE: AN INITIAL ATTEMPT AT TEACHING INTRODUCTORY PROGRAMMING USING ALICE.	23
Tebring Daly, Collin College	
HOW TO DELIVER A GENTLE INTRODUCTION TO PARSING — CONFERENCE TUTORIAL.	31
David Middleton, Arkansas Tech University	

THE FIRST WEBPAGE DESIGN — NIFTY ASSIGNMENT.....	34
Valerie Chu, LeMoyne-Owen College	
GENERATING PRONOUNCEABLE NONSENSE WORDS — NIFTY ASSIGNMENT.....	36
Mark Goadrich, Centenary College of Louisiana	
ELECTRIC BUGLE — NIFTY ASSIGNMENT	38
Carl Burch, Hendrix College	
CREATING A PREDICTIVE MODEL FOR PARKING ON CAMPUS — NIFTY ASSIGNMENT.....	40
Janet Renwick, University of Arkansas - Fort Smith	
GAMERS CONVENTION MODEL — NIFTY ASSIGNMENT.	42
Donna Wright, University of Arkansas-Fort Smith	
SNOWFALL TO INTRODUCE SYSTEM SIMULATIONS — NIFTY ASSIGNMENT	44
Matt Brown, Arkansas Tech University	
COLLABORATION PROBLEMS IN CONDUCTING A GROUP PROJECT IN A SOFTWARE ENGINEERING COURSE.....	45
Priyatham Anisetty, Paul Young, University of Central Arkansas	
FORMAL THEORY FOR SOFTWARE ENGINEERING STUDENTS.....	53
Ken Riggs, Florida A&M University	
ANDROID APPLICATION PROGRAMMING — CONFERENCE TUTORIAL..	60
Frank McCown, Harding University	
HTML 5 PROGRAMMING — CONFERENCE TUTORIAL.....	61
Dan Brandon, Christian Brothers University	
CS1 STUDENTS' UNDERSTANDING OF COMPUTATIONAL THINKING CONCEPTS.....	62
Jake A. Qualls, Michael M. Grant, Linda B. Sherrell, University of Memphis	
TEACHING COMPUTER SCIENCE IN A BIG QUESTIONS FRAMEWORK. . .	72
Steve Donaldson, Samford University	
USING EARLY INSTRUCTION SETS TO INTRODUCE COMPUTER ARCHITECTURE.....	80
David L. Tarnoff, East Tennessee State University	

STAGING A REALISTIC ENTITY RESOLUTION CHALLENGE FOR STUDENTS.....	88
Yinle Zhou and John Talburt, University of Arkansas at Little Rock	
SENSITIVITY OF DIFFERENT MACHINE LEARNING ALGORITHMS TO NOISE.....	96
Abhinav Atla, Victor Sheng, Rahul Tada, Naveen Singireddy, University of Central Arkansas	
IMPROVING NON-SMALL CELL LUNG CANCER CLASSIFICATION IN DATA MINING COURSES.....	104
Quoc-Nam Tran, Lamar University	
PROGRAMMING CONTEST HOSTING — CONFERENCE TUTORIAL.....	113
Steve Baber, Harding University, Becky Cunningham, David Hoelzeman, Arkansas Tech University, Rick Massengale, University of Arkansas Fort Smith	
CHALLENGES AND PROFESSIONAL TOOLS USED WHEN TEACHING WEB PROGRAMMING.....	116
Yi Liu and Gita Phelps, Georgia College & State University	
ROOTKIT ATTACKS AND PROTECTION - A CASE STUDY OF TEACHING NETWORK SECURITY.....	122
Thomas Arnold and T. Andrew Yang, University of Houston - Clear Lake	
DESIGN OF AN EXTENSIBLE INTERPRETER USING INFORMATION HIDING	130
Larry Morell, Arkansas Tech University	
MANAGING VENDOR ALLIANCES — PANEL DISCUSSION.....	137
Janet S. Renwick, University of Arkansas - Fort Smith, Stephen Baber, Harding University, Brian Henehan, Bruce Thigpen, University of Arkansas - Fort Smith	
AN INTRODUCTION TO PROGRAMMING IN HASKELL — CONFERENCE WORKSHOP.....	139
Gabriel J. Ferrer, Hendrix College	
SHORT MOBILE GAME DEVELOPMENT PROJECTS FOR INTRODUCTORY CS COURSES — CONFERENCE WORKSHOP.....	141
Delvin Defoe, Rose-Hulman Institute of Technology, Stan Kurkovsky, Central Connecticut State University, Emily Graetz, Rose-Hulman Institute of Technology	
AMICABLE PAIRS IN PARALLEL — NIFTY ASSIGNMENT.....	144
Gabriel Foust, Harding University	

COMMUNICATION AND ORIGAMI — NIFTY ASSIGNMENT.....	146
Mark Goadrich, Centenary College of Louisiana	
TopSpin — NIFTY ASSIGNMENT.....	148
Mark Goadrich, Centenary College of Louisiana	
OBJECT-ORIENTED SPACE PHYSICS — NIFTY ASSIGNMENT.....	150
Carl Burch, Hendrix College	
DEMONSTRATING THE NEED FOR PRECISION TO FRESHMEN — NIFTY ASSIGNMENT.....	152
David Middleton, Arkansas Tech University	
FUNCTION COMPOSITION IS A GOOD PRACTICE — NIFTY ASSIGNMENT.....	154
Cong-Cong Xing, Nicholls State University	
PAPERS OF THE SEVENTEENTH ANNUAL CCSC CENTRAL PLAINS CONFERENCE.....	157
WELCOME — 2011 CCSC: CENTRAL PLAINS CONFERENCE	158
Mahmoud Yousef, University of Central Missouri	
2011 CENTRAL PLAINS CCSC REGIONAL BOARD MEMBERS.....	159
2011 CENTRAL PLAINS CCSC CONFERENCE STEERING COMMITTEE ..	159
REVIEWERS — 2011 CCSC CENTRAL PLAINS CONFERENCE	160
DEVELOPING A WAREHOUSE CONTROL SYSTEM IN C# — KEYNOTE ADDRESS.....	162
Stuart A. Thomas, Hallmark Cards, Inc.	
HOW TO TRAIN YOUR COMPUTER: THE NEED FOR QUALITY EMBEDDED SYSTEMS EDUCATION — BANQUET ADDRESS.....	163
John W. McCormick, University of Northern Iowa	
THREE HOURS, TWO APPS, NO PROBLEM! A QUICK INTRODUCTION TO IPHONE/IPAD DEVELOPMENT — PRE-CONFERENCE WORKSHOP..	165
Michael Rogers, Northwest Missouri State University	
AN EFFECTIVE WAY OF INTRODUCING IPHONE APPLICATION DEVELOPMENT TO UNDERGRADUATE STUDENTS.....	166
Baoqiang Yan, Deborah Becker, Connie Hecker, Missouri Western State University	

THERE AND BACK AGAIN: LEVERAGING iOS DEVELOPMENT ON MAC OS X.....	174
Michael P. Rogers, Northwest Missouri State University	
A TOOL FOR TEACHING PETRI NETS.....	181
Chao Mei, Xiaoyang Zhang, Wenfei Zhao, Kasi Periyasamy and Mark Headington, University of Wisconsin-La Crosse	
LESSONS LEARNED IN THE DESIGN OF AN UNDERGRADUATE DATA MINING COURSE.....	189
Cecil Schmidt, Washburn University	
AN INTRODUCTION TO DISTRIBUTED VERSION CONTROL SYSTEMS — CONFERENCE WORKSHOP.....	196
John Cigas, Park University	
CRITICAL THINKING AND MODELING IN CS0: THE PRISONER'S DILEMMA	197
David Reed, Creighton University	
REQUIRING OUTREACH FROM A CS0-LEVEL ROBOTICS COURSE.....	205
Janice L. Pearce, Berea College	
A NEEDS AND REQUIREMENTS ELICITATION SIMULATION — NIFTY ASSIGNMENT.....	213
Edward J. Mirielli, Westminster College	
BOID SWARMS — NIFTY ASSIGNMENT.....	215
Andrew Mertz and Nancy Van Cleave, Eastern Illinois University	
SKIP-3 SOLITAIRE — NIFTY ASSIGNMENT.....	216
David Reed, Creighton University	
USING SAP ERP SOFTWARE IN ONLINE DISTANCE EDUCATION.	218
John L. Wilson and Ed Lindoo, Regis University	
INFORMATICS IN THE CLASSROOM: AN EVALUATION OF INNOVATIVE AND EFFECTIVE USE OF TECHNOLOGY IN ONLINE COURSES	225
John P. Buerck, Patricia G. Bagsby, Stephanie E. Mooshegian, and Lisa M. Willoughby, Saint Louis University	
ORGANIZING AND HOSTING A STUDENT PROGRAMMING CONTEST — PANEL DISCUSSION.	233
Brian Hare, Moderator, University of Missouri-Kansas City, James Cain, Southwest Baptist University, John Cigas, Park University	

SCRATCH THE WORKSHOP AND ITS IMPLICATIONS ON OUR WORLD OF COMPUTING.....	235
Judy Clark, Michael Rogers, Carol Spradling, Northwest Missouri State University	
MULTI-STEP PROBLEM SOLVING USING SCRATCH: A PRELIMINARY REPORT.....	244
William Siever, Linda Heeler, Phillip Heeler, Northwest Missouri State University	
EMBEDDED AND REAL-TIME PROGRAMMING WITH ADA — CONFERENCE WORKSHOP.....	249
John W. McCormick, University of Northern Iowa	
FIVE FOCUSED STRATEGIES FOR INCREASING RETENTION IN COMPUTER SCIENCE 1.....	252
Tim DeClue, Jeff Kimball, Baochuan Lu, James Cain, Southwest Baptist University	
WORKING TO CHANGE PERCEPTIONS.....	259
Ken T. N. Hartness, Sam Houston State University	
BUILDING A THRIVING CS PROGRAM AT A SMALL LIBERAL ARTS COLLEGE.....	268
Timothy Urness and Eric Manley, Drake University	
STUDENT ENGAGEMENT THROUGH APPLIED LEARNING: THE "LIVE" BUSINESS PARTNERSHIP MODEL.....	275
Deborah Becker, Connie Hecker, Missouri Western State University	
FRANCES-A: A TOOL FOR ARCHITECTURE LEVEL PROGRAM VISUALIZATION.	283
Tyler Sondag, Iowa State University, Kian L. Pokorny, McKendree University, Hridesh Rajan, Iowa State University,	
TEACHING AGILE METHODOLOGY IN A SOFTWARE ENGINEERING CAPSTONE COURSE.	293
Baochuan Lu, Tim DeClue, Southwest Baptist University	
INDEX OF AUTHORS.....	300

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@mcperson.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.

Jim Aman, Membership Chair (2013), Assoc. Professor, Computer Science, Saint Xavier University, Chicago, IL 60655, 773-298-3454 (O), 630-728-2949 (cell), aman@sxu.edu.

Bill Myers, Treasurer (2011), 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 3668, APO AE 09063; (H) Werderstr 8, D-68723 Oftersheim, Germany, 011-49-6202-5 77 79 16 (H), meinkej@acm.org.

Kim P. Kihlstrom, Southwestern Representative (2011), Associate Professor of Computer Science, Westmont College, 955 La Paz Road, Santa Barbara, CA 93108, 805-565-6864 (O), 805-570-6722 (cell), 805-565-7036 (fax), kimkihls@westmont.edu.

Elizabeth S. Adams, Eastern Representative (2011), Associate Professor Emeritus, James Madison University - Mail Stop 4103, CISAT - Department of Computer Science, Harrisonburg, VA 22807, adamses@jmu.edu.

Deborah Hwang, Midwestern Representative (2011), Dept. of Electrical Engineering and Computer Science, University of Evansville, 1800 Lincoln Avenue, Evansville, IN 47722, 812-488-2193 (O), 812-488-2780 (fax), hwang@evansville.edu.

Scott Sigman, Central Plains Representative (2011), Associate Professor of Computer Science, Drury University, Springfield, MO 65802, 417-873-6831, scott.sigman58@gmail.com..

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 Greenvue 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 3669, 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.

Paul D. Wiedemeier, National Partners Chair, 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), wiedemeier@ulm.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, MI 48801, 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.

Microsoft Corporation

Turing's Craft

National Science Foundation

Panasonic Solutions Company

FOREWORD

And, once again two conferences have produced great programs which I'm proud to be a part of. As I look at the programs I really wish that I could attend both!

Getting this manuscript out is always a challenge. We have the conference committees putting programs together and then refining them, but we also have printing deadlines. My thanks to the Dan Brandon and Dean Sanders for their hard work feeding me manuscript that was carefully laid out by the respective conference committees. My thanks also to Dan and Dean for their responsiveness when I was "harassing" them that we had printing and shipping deadlines to worry about. The programs for both conferences are excellent, and I heartily commend the papers in this issue. Those able to attend the conferences will benefit greatly from the presentations and we will all benefit greatly from having the copy of the papers from both conferences available to us. This is particularly true for those of us unable to attend either conference.

This year printing deadlines were particularly tight. Authors need the time in January to prepare their final manuscripts, and then we need to do the reformatting for the final copy. The first conference in this issue is the first weekend in April. Normally we try to allow ten weeks from the author's final manuscript submission so that we have adequate time to reformat, get the final manuscript to the printer, receive the "blue line" back for final proofing and making a few minor corrections (hopefully), giving the printer the go-ahead to print, get the printing accomplished, and the final document to the respective conferences in time for the beginning of the conference. (Shipping alone takes a week!) Backing up ten weeks from that first conference placed us in mid-January, which was not really adequate time for authors to receive acceptance notifications and prepare their final manuscripts. In addition, we always have some "stragglers" whose manuscripts are late in arriving. Hopefully we have found all of the minor errors and the final copy is what the individual authors expected.

That brings up the pre-acceptance activities. I have heard people questioning why the initial manuscript submission is so far ahead of the conference. Every paper submission undergoes a double-blind reviewing process. Once the reviewing process is complete the conference committee determines acceptances. Sometimes a perfectly good

paper that received high reviews is not accepted because it does not fit into the conference program. The work of the conference committee is not easy – I have served on conference committees and can vouch for that. One wants only the best papers but must also balance that with the conference program. I commend the respective conference committees for the two conferences represented in this issue of the *Journal* for their work in bringing a quality conference to fruition. Each conference welcome indicates the acceptance rate for submissions for the respective conferences.

I would also call your attention to six Nifty Assignments that appear at the end of the Mid South conference. Those were not received in time for publication in last year's papers. There will not be a presentation of those six since the presentation took place in 2010.

Let me take a moment here to call your attention to our National Partners listed above. Their financial support enables us to continue to offer the CCSC regional conferences at an inexpensive registration rate, allowing those of us with minimal travel budgets to attend professional conferences. Their generosity is very much appreciated.

Let me also recognize Upsilon Pi Epsilon as a supporter of student programs at our regional conferences.

My thanks to all the supporters and facilitators, and it's a pleasure to present two more sets of excellent conference papers.

John Meinke
UMUC – Europe
CCSC Publications Chair

Papers of the Ninth Annual CCSC Mid-South Conference

**April 1-2, 2011
University of Central Arkansas
Conway, Arkansas**

WELCOME TO THE 2011 CCSC MID-SOUTH CONFERENCE

On behalf of the conference committee, I welcome you to the Ninth Annual Consortium for Computing Sciences (CCSC) Conference for the Mid-South region. We are pleased to be hosted this year by the University of Central Arkansas in Conway, Arkansas.

The conference program includes fifteen papers, five tutorials, five workshops, two panels, and two sessions of oral presentations of undergraduate student research. The program also contains the Nifty Assignments session began last year and continues to host our annual undergraduate student programming competition. Sessions range across a broad spectrum of computing topics.

The twenty-nine papers submitted were reviewed using a double-blind process with at least three reviewers per paper, one of which was on the program committee. Fifteen papers were accepted for publication in the Journal of Computing Sciences in Colleges and for presentation in this year's conference. We thank all of the authors for their hard work and their high-quality submission and all the reviewers for their careful reviews.

A special word of thanks is extended Henry Walker who graciously supported our conference in past years by allowing us to use the paper submission system he developed and maintains at Grinnell College.

The Mid-South conference committee, the regional board, and the faculty and staff of the University of Central Arkansas Department of Computer Science worked hard organizing this conference. I thank all of you for your varied contributions and the many hours of hard work that make this conference possible.

We hope you enjoy the conference and find it to be a valuable opportunity for professional development. We also look forward to seeing you at next year's CCSC Mid-South Conference hosted by Union University in Jackson Tennessee.

Larry Morell
CCSC-MS 2011 Chair

2011 MID-SOUTH CONFERENCE COMMITTEE

Larry Morell, Conference Chair	Arkansas Tech University, Russellville, AR
Vamsi Paruchuri, Site Chair	University of Central Arkansas, Conway, AR
David Naugler, Papers Co-Chair	Southeast Missouri State University, Cape Girardeau, MO
Janet Renwick, Papers Co-Chair	University of Arkansas at Fort Smith, Fort Smith, AR

CCSC: Mid-South Conference

Otha Britton, Panels/Workshops/Tutorials Co-Chairs	University of Tennessee - Martin, Martin, TN
Conrad Cunningham, Panels/Workshops/Tutorials Co-Chair.	University of Mississippi Oxford, MS
Carl Burch, Nifty Assignments Chair	Hendrix College, Conway, AR
Steve Baber, Student Programming Contest Co-Chair	Harding University, Searcy, AR
Max Li, Student Programming Contest Co-Chair . . .	Union University, Jackson, TN
Michael Noonan, Student Programming Contest Co-Chair.	University of Central Arkansas, Conway, AR
Mark Goadrich, Student Papers Co-Chairs.	Centenary College of Louisiana, Shreveport, LA
David Middleton, Student Papers Co-Chair.	Arkansas Tech University, Russellville, AR
Linda Sherrell, Publicity Chair.	University of Memphis, Memphis TN
Gabriel Ferrer, Past Conference Chair.	Hendrix College, Conway, AR

REGIONAL BOARD — CCSC MID-SOUTH REGION

Gabriel Ferrer, Regional Board Chair and Past Site Chair (2010).	Hendrix College, College, Conway, AR
Linda Sherrell, CCSC National Board Representative and Treasurer.	University of Memphis, Memphis, TN
Paul D. Wiedemeier, Membership and Registration Chair.	University of Louisiana at Monroe, Monroe, TN
Larry Morell, Conference Chair.	Arkansas Tech University, Russellville, AR
Daniel Brandon, Editor.	Christian Brothers University, Memphis, TN
David Hoelzeman, Webmaster.	Arkansas Tech University, Russellville, AR
Vamsi Paruchuri, Site Chair.	University of Central Arkansas, Conway, AR
Max Li, Next Site Chair (2012).	Union University, Jackson, TN

REVIEWERS — 2011 CCSC MID-SOUTH CONFERENCE

David Hoelzeman.....	Arkansas Tech University, Russellville, AR
Larry, Morell.....	Arkansas Tech University, Russellville, AR
Steve, Baber.....	Harding University, Searcy, AR
Carl, Burch.....	Hendrix College, Conway, AR
Janet, Renwick.....	University of Arkansas at Fort Smith, Fort Smith, AR
Gabriel, Ferrer.....	Hendrix College, Conway, AR
Otha, Britton.....	University of Tennessee at Martin, Martin, TN
Conrad, Cunningham.....	University of Mississippi, Oxford, MS
David, Middleton.....	Arkansas Tech University, Russellville, AR
Mark, Goadrich.....	Centenary College of Louisiana, Shreveport, LA
Paul, Wiedemeier.....	University of Louisiana at Monroe, Monroe, LA
Dan, Brandon.....	Christian Brothers University, Memphis, TN
David, Naugler.....	Southeast Missouri State University, Memphis, TN
Vamsi, Paruchuri.....	University of Central Arkansas, Conway, AR
Sarah, North.....	Southern Polytechnic State University, Marietta, GA
Thomas, Arnold.....	University of Houston - Clear Lake, Clear Lake, TX
Kuo-pao, Yang.....	Southeastern Louisiana University, Hammond, LA
Jenq-Foung, Yao.....	Georgia College & State University, Milledgeville, GA
Tsu-Ming, Chiang.....	Georgia College & State University, Milledgeville, GA
Samar, Swaid.....	Philander Smith College, Little Rock, AR
David, Tarnoff.....	East Tennessee State University, Johnson City, TN
Jake, Qualls.....	University of Memphis, Memphis, TN
Michael, Grant.....	University of Memphis, Memphis, TN
Steve, Donaldson.....	Samford University, Birmingham, AL
Jessie, Walker.....	Philander Smith College, Little Rock, AR
Jose, Cordova.....	University of Louisiana at Monroe, Monroe, LA
Virginia, Eaton.....	University of Louisiana at Monroe, Monroe, LA
Yinle, Zhou.....	University of Arkansas at Little Rock, Little Rock, AR
Paul, Young.....	University of Central Arkansas, Conway, AR
Priyatham, Anisetty.....	University of Central Arkansas, Conway, AR
Victor, Sheng.....	University of Central Arkansas, Conway, AR
Cong-Cong, Xing.....	Nicholls State University, Thibodaux, LA
Quoc-Nam, Tran.....	Lamar University, Beaumont, TX
Iraj, Danesh.....	Alabama State University, Montgomery, AL
Tebring, Daly.....	Collin College, McKinney, TX
Theresa, Beaubouef.....	Southeastern Louisiana University, Hammond, LA
Ghassan, Alkadi.....	Southeastern Louisiana University, Hammond, LA
Patrick, McDowell.....	Southeastern Louisiana University, Hammond, LA
Yi, Liu.....	Georgia College & State University, Milledgeville, GA
Gita, Phelps.....	Georgia College & State University, Milledgeville, GA

AN INTRODUCTION TO OPENMP*

PRE-CONFERENCE WORKSHOP

*David Naugler
Southeast Missouri State University
One University Plaza
Cape Girardeau MO, 63701
(573)651-2787
dnaugler@semo.edu*

OpenMP is an API for shared memory parallel programming and is a straightforward way to write multithreaded programs on multi core computers. It is widely available and is supported for major C/C++ and Fortran compilers.

OpenMP uses compiler directives (pragmas in C/C++), environment variables and functions instead of additional programming language statements, so there is not a new language to learn. Using OpenMP a program can be incrementally parallelized and by using appropriate macros the same source code can be used for both the sequential and the parallel versions of a program. As a consequence OpenMP can be taught and learned incrementally.

Since MPI was designed for distributed memory parallel programming MPI and Open MP have different domains and can both be used in a single program for a cluster of multi core machines.

This workshop is a hands-on introduction which incrementally covers the main features of OpenMP using examples in C. A reasonable knowledge of C/C++ is assumed.

* Copyright is held by the author/owner.

A WEB-BASED GRAPHICAL SET THEORY PROOF

ASSISTANCE SYSTEM*

Cong-Cong Xing

Department of Mathematics and Computer Science

Nicholls State University

Thibodaux, LA 70310

cong-cong.xing@nicholls.edu

ABSTRACT

Learning set theory proofs is an important but frustrating task for students. This is partially due to the (traditional) way of writing proofs as English sentences mixed with mathematical notations. The structure of logical reasoning which constitutes the essence of a proof is buried in such writings and may not become explicitly clear to the student. As a result, unnecessary comprehension barriers may be created. Towards resolving this issue, in this paper, we present a web-based graphical set theory proof assistance software system. We first identify the problems that a traditional writing of proofs may cause, and then propose the proof assistance system by describing its specification and implementation. Next, we give an example to show how the system works, and finally, we conclude this paper with some remarks.

1 INTRODUCTION

It has been widely discussed and agreed upon that (discrete) mathematics is a vitally important component in computer science education (see e.g. [11, 10, 1]). Unfortunately, handling mathematics is a difficult and frustrating task for most college and university students, which has led to the so called “math anxiety” [9, 12, 8] for a long period of time. Among various topics of mathematics, mathematical proofs, which require rigorous thinking and logical reasoning, are one of the toughest subjects for the students. Within the subject of mathematical proofs, set theory proof is of fundamental importance in the sense that (1) set theory is the foundation of all other subjects in mathematics and computer science theory; (2) set theory proof is encountered at an early stage by the

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

student in their mathematical studies, a failure of handling set theory proofs successfully will have a direct impact on the confidence and effectiveness of their further theoretical studies.

There are many factors that contribute to the fact that set theory proof is difficult to grasp for the student. One of them is the way in which the proof is presented. The traditional way of writing proofs is to use the combination of English sentences and mathematical notations. While this textual presentation of proofs has been used for hundreds of years, we argue that the structure of logical reasoning, which constitutes the essence of a proof, is not immediately clear under this type of presentation of proofs; and that not every student can figure out the structure of logical reasoning by delving into such a textual writing of proofs. In light of this, we believe that a graphical representation of proofs would be better — the nodes, edges, and their connections in a graph can make the order and structure of logical reasoning *explicitly* clear, and thereby offer an improved presentation of proofs.

Based on this idea, we have implemented a web-based software tool which can be used to assist students in conducting set theory proofs. It is worthy of noting that although there are many mathematics software tools available on the market (e.g., Maple [2], Matlab [3], Mathematica [4], Coq [5], Lillypadz [6]), none of them, as far as we know, is able to perform a set theory proof in an interactive and graphical manner as our work does. We present this software tool in this paper.

The paper is organized as follows: Section 2 identifies some problems that a traditional style proof may cause; section 3 presents the proof assistance system; section 4 provides an example demonstrating how the system works; and section 5 concludes the paper.

2 THE PROBLEM

The problem with the traditional style of writing proofs is that it may inevitably create an unnecessary comprehension barrier for the student, especially for those students with insufficient mathematical background or training. For example, the traditional proof for $(A \cup B) - C = (A - C) \cup (B - C)$ may be something like the following. (The places that students may have trouble with are italicized.)

We first show $(A \cup B) - C \subseteq (A - C) \cup (B - C)$. Assume $x \in (A \cup B) - C$, then $x \in (A \cup B)$ but

$x \notin C$. $x \in (A \cup B)$ implies that either $x \in A$ or $x \in B$. If $x \in A$, considering $x \notin C$, we can see that $x \in A - C$ and hence $x \in (A - C) \cup (B - C)$, which indicates $(A \cup B) - C \subseteq (A - C) \cup (B - C)$. If $x \in B$, remembering $x \notin C$, then we have $x \in B - C$ and hence $x \in (A - C) \cup (B - C)$ which indicates $(A \cup B) - C \subseteq (A - C) \cup (B - C)$ again. Now, we show that $(A - C) \cup (B - C) \subseteq (A \cup B) - C$. Let $x \in (A - C) \cup (B - C)$. Then either $x \in A - C$ or $x \in B - C$. In the first case, we can see that $x \in A$ but $x \notin C$ which indicates $x \in A \cup B$ and $x \notin C$, and $x \in (A \cup B) - C$ follows immediately. Therefore, $(A \cup B) - C \subseteq (A - C) \cup (B - C)$. In the second case, we have $x \in B$ but $x \notin C$, and the same result can be obtained in

a *similar* fashion. Since $(A \cup B) - C \subseteq (A - C) \cup (B - C)$ holds in both cases, combining them gives us what desired.

For the proof above, we have found that students may have the following issues in terms of understanding the proof.

- The word “but” actually means “and” here in this proof. Most students miss this point.
- Students have trouble understanding that $x \notin C$ is true for the case $x \in A$ and also true for the case $x \in B$, and therefore cannot “remember” it when the proof reaches the case of $x \in B$.
- The exact logical meanings connoted by the words “same” and “similar” may not be clear to the student.
- Students may not be able to comprehend correctly why $x \in A \cup B$. Some students think it is so because $x \in A$ in the first case and $x \in B$ in the second case, and the combination of the two cases gives the result. They do not seem to realize that the reasoning processes of the two cases cannot be mixed.
- After being pointed out that the reasoning processes of the two cases cannot be mixed, some students may ask why the proof combines the two cases at the end of the proof. They have trouble following through the logical inferences in the proof and may be hampered by the static meaning of a few English words.

Towards resolving this kind of problems, we have developed a set theory proof assistance software system which can perform set theory proofs in a graphical and interactive manner.

3 THE PROOF ASSISTANCE SYSTEM

We now describe the proof assistance system. The operational specification of the system is given first which is followed by an exposition of the implementation of the system.

3.1 Operational Specification

We call what is initially known at the beginning of a proof problem a *fact*; anything that can be derived from it (in one or more steps) a *subfact*; what is originally to be proved a *goal*; and anything that needs to be proved in order to prove the goal (in one or more steps) a *subgoal*.

The system allows the user to choose any one of the following three proof strategies to conduct the proof:

- *Top down*. The proof starts with the fact and moves forward towards the goal by first deriving subfacts from the fact and then repeatedly deriving further subfacts from subfacts. This process continues until the goal is reached.
- *Bottom up*. The proof starts with the goal and moves backward towards the fact by first deriving subgoals (reversely) from the goal and then repeatedly deriving further

subgoals (reversely) from subgoals. The process continues until all subfacts are satisfied.

- *Hybrid.* This is a mix of the top down and the bottom up strategies. The proof starts with either the fact or the goal and moves forward and backward alternately. This process continues until all subgoals are matched by subfacts.

The behavior of the desired program should be as follows:

- The program prompts the user to enter what needs to be proved (e.g., $(A \cup B) - C \subseteq (A - C) \cup (B - C)$). To facilitate the entering of the symbols of union, intersection, subset, empty set, etc., buttons of these symbols should be provided so that internal representations of these symbols (e.g. standard L^AT_EXnotations) can be automatically generated by clicking the relevant buttons.
- The program prompts the user to enter one of the three proof strategies mentioned above.
- The program takes what entered by the user and initially displays the fact and the goal at the top and the bottom, respectively, of the proof diagram to be produced.
- The fact, subfacts, goal, and subgoals are to be displayed in color-coded rectangles.
 - Rectangles containing a fact or a subfact are filled with green color.
 - Rectangles containing a goal or a subgoal are filled with yellow color.
 - Rectangles containing a subfact or a subgoal which is not currently valid (see below for more details) are filled with blue color.
- Each time a fact/subfact (or goal/subgoal) rectangle is clicked, further subfacts (or subgoals) which can be inferred (or reversely inferred) from it are generated and displayed. If no subfacts (or subgoals) can be derived (or reversely derived), a warning message will be shown.
- When a subfact (or subgoal) rectangle is clicked and there are two possible results, the program prompts the user to enter a choice. The selected choice will be displayed in a green-colored (or yellow-colored) rectangle, and the unselected choice will be displayed in a blue-colored rectangle indicating that it is not currently valid.
 - At any point in the proof, any green-colored rectangle can be used in combination with other green-colored rectangles to derive further subfacts.
 - If a blue-colored rectangle is clicked, a warning message will be displayed stating that what contained in the rectangle is not currently valid (yet). (But, it may be valid at a later point or at an earlier point during the proof.)
 - A subfact rectangle and a subgoal rectangle can be checked to see if they match. This is performed in a “drag-and-drop” fashion — press and hold the mouse button on one of the rectangles and move the mouse pointer to the other rectangle, and then release the mouse button. A confirming message will be displayed if they match, and a line will be drawn to connect these two rectangles.

3.2 Implementation

Because of the ubiquity of the web, we implemented a web-based set theory proof assistance system so that it can be immediately integrated into the web and become available to the student. These web pages were written using HTML, JavaScript, and the newly developed (still under development) graphics-drawing `<canvas>`¹ element standardized by WHATWG [7]. While HTML provided a working environment for the implementation, JavaScript was actually the programming workforce, and all drawings were performed by using the `<canvas>` element.

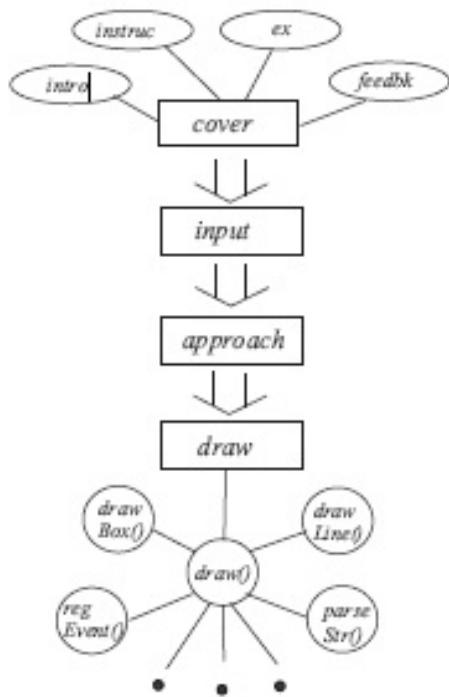


Figure 1: Structure of the Implementation. *Cover page* starts the set theory proof assistance system. *Introduction page* explains what this software tool is all about. *Instruction page* provides steps that need to be followed to use this proof assistance tool. *Example page* includes some screen shots of the problems worked out by this software tool. *Feedback page* collects comments and suggestions. *Input page* prompts the user to enter the input — what needs to be proved. *Approach page* asks the user to select one of the three proof approaches. *Draw page* performs the computation and displays the proof diagram in an interactive fashion. This page is the core of the system. About 50 JavaScript functions, including those that use the `<canvas>` elements, were written for this page to accomplish its work.

Figure 1 shows the implementation structure of the software system where primary HTML pages, auxiliary HTML pages, and JavaScript functions are signified by rectangle, oval, and circle, respectively. Due to the space limit of this paper, we only describe some of the most important JavaScript functions here.

- `draw()` is the entry point of the entire set of function calls, where the drawing context of the `<canvas>` element is set up and the input string entered on Input page is extracted and cleaned.
- `drawBox()` performs the drawing of the color-coded (green, yellow, blue) rectangles. Each of these rectangles is created dynamically as a `<canvas>` element using the DOM tree model.
- `drawLine()` performs the drawing of the straight lines that connect the color-coded rectangles. Similar to the rectangles created by `drawBox()`, these lines are also created dynamically.

¹ A web browser supporting the `<canvas>` element, such as Firefox, is needed to run the implementation.

- *parseStr()* parses an input string into a binary tree and makes preparation for the string to be displayed.
- *regEvent()* registers the `onmouseup` and `onmousedown` events to each of the color-coded rectangles preparing them for interactive actions.

4 AN EXAMPLE

We now provide an example to demonstrate how the proof system works.

Figures 2 and 3 contain the screen shots of the system when $(A - C) \cup (B - C) \subseteq (A \cup B) - C$ is being proved. Figure 2(a) shows the cover page. Figure 2(b) shows the input page where $(A - C) \cup (B - C) \subseteq (A \cup B) - C$ is entered. To enter the \cup , \cap , \subseteq , ... symbols, users can either click the corresponding buttons on the page or type their L^AT_EX commands directly. Figure 2(c) prompts the user to select a proof approach and the size of the proof diagram to be produced.

Figure 2(d) shows the initial configuration of the proof diagram where the fact and the goal are displayed at the top and the bottom, respectively. Figure 2(e) shows the situation when the fact rectangle is clicked. Since two possible subfacts can be derived from the fact, the system asks the user to enter a choice. The first choice was entered by the user. Figure 2(f) shows the result when the OK button in Figure 2(e) is clicked. The first subfact $x \in A - C$ is contained in a green rectangle because it was the choice selected by the user. The second subfact $x \in B - C$ is contained in a blue rectangle because it was not selected by the user. (But the user must come back at a later point in the proof to select this choice and proceed from there.) Remember, green rectangles represent valid fact/subfacts and can be clicked to produce further subfacts; blue rectangles represent unselected subfacts/subgoals and cannot be clicked to produce further subfacts/subgoals. Figure 2(g) shows the situation when the goal rectangle is clicked. Two subgoals $x \in A \cup B$ and $x \notin C$ are produced; both need to be satisfied in order to satisfy the goal. Figure 2(h) depicts the situation when the subfact rectangle $x \in (A - C)$ is clicked. Two valid subfacts $x \in A$ and $x \notin C$ are derived.

Figure 3(i) shows the situation when the subgoal $x \in A \cup B$ is clicked. There are two possible subgoals and the user is prompted to enter a choice. Note that *only* one of the two subgoals needs to be satisfied in order to satisfy the subgoal $x \in A \cup B$. The first choice was entered. Figure 3(j) shows the result when the OK button in Figure 3(i) is clicked. The first subgoal $x \in A$ is in a yellow rectangle since it was selected by the user, and needs to be satisfied; the second subgoal $x \in B$ is in a blue rectangle since it was not selected by the user, and does not need to be satisfied. Figure 3(k) shows the subgoal $x \in A$ is satisfied when it is checked with the subfact $x \in A$ by dragging one of the rectangles and dropping it onto the other. Figure 3(l) shows a similar situation when the subgoal $x \notin C$ is checked with the subfact $x \in A$. At this point, the satisfaction of the subgoal $x \in A$ satisfies the subgoal $x \in A \cup B$ which, in turn, together with the satisfied subgoal $x \notin C$, satisfy the original goal $x \in (A \cup B) - C$. Therefore, a proof is established (under the choices made by the user).

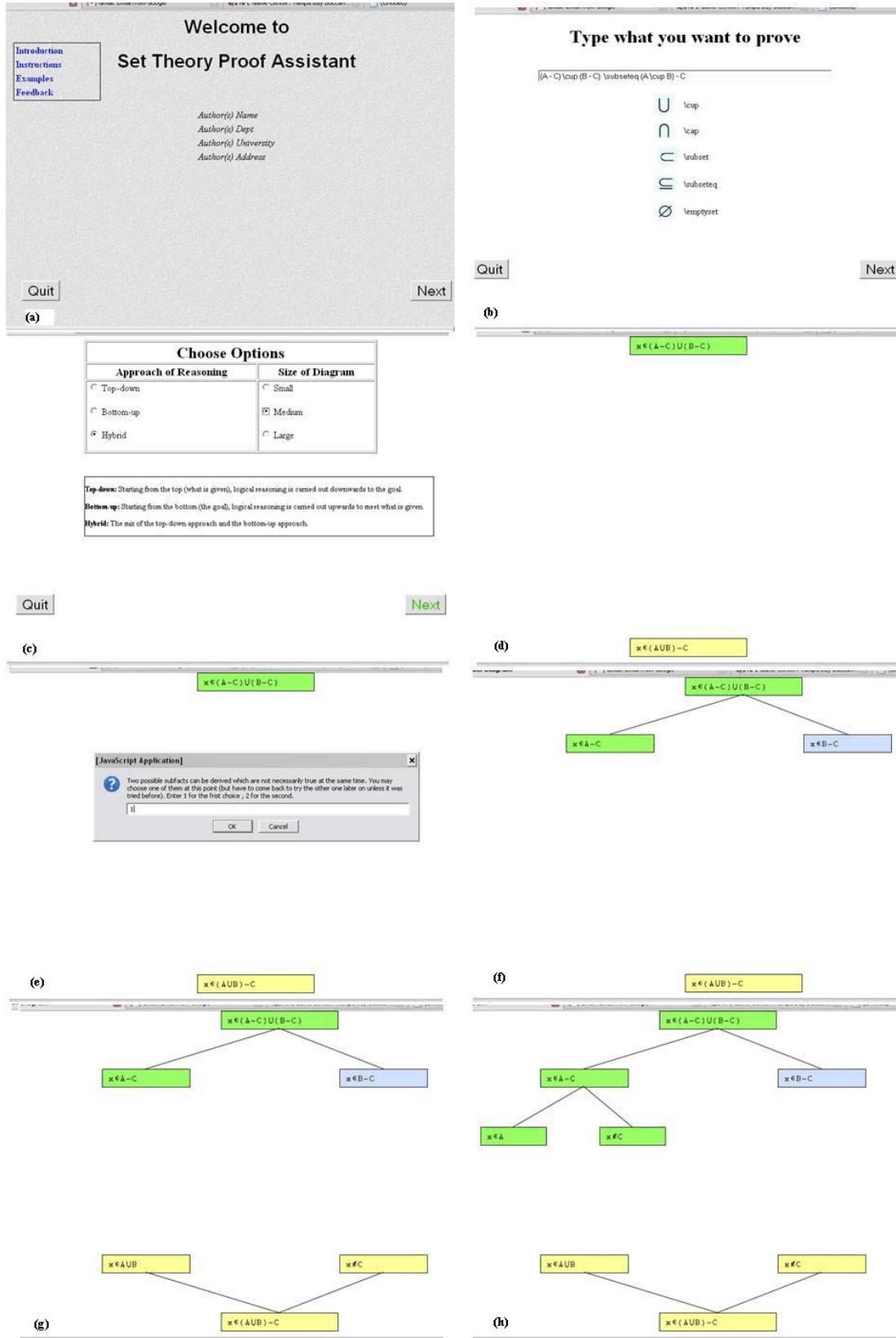
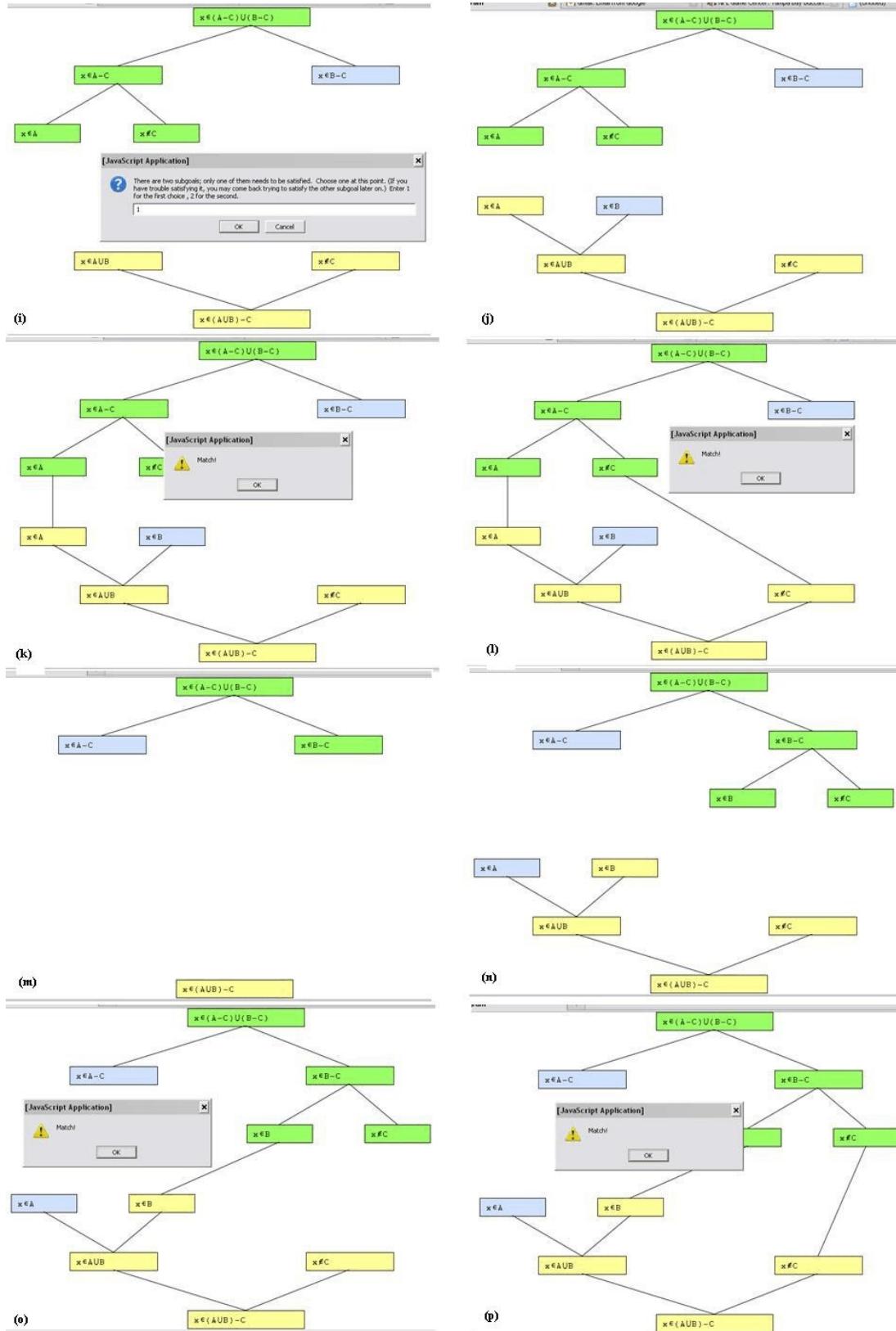


Figure 2: Proof of $(A - C) \cup (B - C) \subseteq (A \cup B) - C$

Figure 3: Proof of $(A - C) \cup (B - C) \subseteq (A \cup B) - C$ (continued).

As the prompt in Figure 2(e) indicates, a user must try the other choice of subfacts later on in a proof after choosing one of the two choices of subfacts, in order to have a complete proof. Figure 3(m) depicts the situation when the second choice is entered and OKed in Figure 2(e). Now, the first subfact $x \in (A - C)$ becomes invalid/unselected (in blue colored rectangle) and the second subfact $x \in (B - C)$ becomes valid (in green colored rectangle). Figure 3(n) shows the situation after the subfact $x \in (B - C)$ is clicked, the goals $x \in (A \cup B) - C$ is clicked, and the second choice is entered and OKed at the prompt when the subgoal $x \in (A \cup B)$ is clicked (see Figure 3(i) for the prompt). Figures 3(o) and 3(p) show the subsequent matchings of the subgoal $x \in B$ with the subfact $x \in B$, and the subgoal $x \notin C$ with the subfact $x \notin C$. At this point, the entire proof is completed.

5 CONCLUDING REMARKS

While learning set theory proofs is an important task for the students, we recognize that this task is a difficult and frustrating activity which is partially due to the way in which the proofs are presented. Typically, proofs are written in plain English sentences mixed with mathematical notations. This textual description of proofs may not be able to deliver the structure of logical reasoning as clearly as we would like, and consequently creates unnecessary comprehension hindrance. Towards resolving this issue, we have developed a web-based software tool which can construct set theory proofs in a graphical fashion and assist students in an interactive manner. We believe that this software tool makes the following contributions:

- It is *visual*. The structure of logical reasoning which defines a proof is displayed explicitly and clearly by this software tool.
- It is *as simple as possible*. English language writing skills which are required in traditional writing of proofs and may cause comprehension problems if not used carefully, are not a necessary part in this software system.
- It is *interactive*. By interacting with this software tool, students can see and “feel” how a proof is built up step-by-step, piece-by-piece, and in different ways.
- It is a *framework*. Students can learn and develop logical thinking skills by following the way in which a proof diagram is produced by the software tool.
- It is the *first* software system of its kind. As far as we know, our work represents the first effort in making a software system that can conduct set theory proofs graphically and interactively.

6 ACKNOWLEDGEMENTS

This work was made possible by a Nicholls State University Research Council grant.

REFERENCES

- [1] <http://www.math-in-cs.org/>.

- [2] [http://www.maplesoft.com/Products/Maple/.](http://www.maplesoft.com/Products/Maple/)
- [3] [http://www.mathworks.com/.](http://www.mathworks.com/)
- [4] [http://www.wolfram.com/.](http://www.wolfram.com/)
- [5] [http://coq.inria.fr/.](http://coq.inria.fr/)
- [6] [http://lillypadz.com/.](http://lillypadz.com/)
- [7] [http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html.](http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html)
- [8] Marilyn Burns. *MATH: Facing an American Phobia*. Math Solutions Publications, 1998.
- [9] Robert Hackworth. *Math Anxiety Reduction*. H. and H. Publishing, 1985.
- [10] Special Issue. Why universities require computer science students to take math. *CACM*, 46(9), 2003.
- [11] Jeff Kramer. Is abstraction the key to computing? *CACM*, 50(4), 2007.
- [12] Sheila Tobias. *Overcoming Math Anxiety*. W.W. Norton & Company Inc., 1994.

EXPERIENCES IN COMPUTER SCIENCE WONDERLAND: A SUCCESS STORY WITH ALICE*

Jose Cordova, Virginia Eaton, Kimberly Taylor

Department of Computer Science and Computer Information Systems

University of Louisiana at Monroe

(318) 342-1855, (318) 342-1842, (318) 342-1848

cordova@ulm.edu, eaton@ulm.edu, ktaylor@ulm.edu

ABSTRACT

Computer Science Wonderland, a non-residential camp involving 36 students and 12 teachers from area high schools, was held on the campus of our institution during the summer of 2010. The focus of the camp was to engage teachers and students in the development of graphical animations using the Alice programming environment, with the objective of introducing teachers and students to programming and logical problem solving. A secondary objective was to generate interest in the computing profession by presenting and completing activities in the areas of computing history, computing ethics, computing careers, Boolean logic, binary number representation, and cryptography. Throughout the camp, emphasis was placed on a few recurring themes: teamwork, creativity, and interdisciplinary relationships between computing and the arts/humanities. This paper offers a model for and evaluation of a computer science camp for middle/secondary school students and teachers.

PREVIOUS EXPERIENCES

It is widely recognized that, in order to attract a larger number of students into our profession, interest in computing must be fostered early. The student component of the Computer Science Wonderland Summer Camp was based on extensive prior experience working with middle and high school students [1, 2, 3, 4]. One of the authors taught computer science to middle and high school students in a summer residential program at

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Western Kentucky University (WKU) from 1985 to 1987. The WKU program was affiliated with the Duke Talent Identification Program (TIP), which targets gifted students. Subsequently, she received funding through the National Science Foundation's (NSF) Young Scholars Program from 1989 through 1997. The three-week summer residential Young Scholars Program targeted females and underrepresented minority students from low-income backgrounds. The primary teaching tool was LogoWriter and its subsequent iteration, MicroWorlds. In 2000, this author received an NSF grant from the Gender Equity program for Girls R.O.C., a summer residential program targeting low-income females. Girls R.O.C. also utilized MicroWorlds as its primary teaching tool. In the summers of 2005-2008, another one of the authors was the project director for one-week summer residential camps funded by LA GEAR UP (Louisiana Gaining Early Awareness and Readiness for Undergraduate Programs). The LA GEAR UP camps are targeted at middle and secondary students in low-income and academically challenged areas. Using Alice, the third author taught the computer science component of the LA GEAR UP camps. All of these programs were highly successful and included many of the components that were incorporated into the Computer Science Wonderland Summer Camp. Most important was that the primary teaching tool in all cases was a programming language that made it easy for beginning students to create reasonably complex projects that include graphics and music.

As with the student component, the teacher component of the Computer Science Wonderland Camp was based on the authors' prior experiences working with teachers [5]. In 1994-1995, one of the authors received two grants funded by NSF to develop a Rural Systemic Initiative grant proposal to improve K-12 education in Northeast Louisiana. From 1995 to 1999, she was the recipient of a Teacher Enhancement grant funded by NSF to train K-12 teachers to incorporate computers and the Internet into the teaching of mathematics and science. These teacher participants took a year's sabbatical and trained to be peer leaders in their school systems. In 2003, she was one of the founders of the Delta Regional Educators Academy (DREAM) at the university. DREAM was established to provide K-12 teachers with training to improve the teaching of mathematics and science. A particular focus of DREAM was the incorporation of technology, especially computers and the Internet into classroom teaching. In 2004, DREAM received an Innovative Technology Experiences for Students and Teachers (ITEST) grant from NSF; the Delta Agriculture Middle School Applied Life Science (DAMSALS 2) involved STEM teachers in a two-week summer program followed by a one-week program in which both teachers and students participated. In addition to DAMSALS, DREAM received state funding from the Louisiana Systemic Initiatives Program (LaSIP) and the Mathematics and Science Partnership (MSP) for training teachers. Evaluators from within and outside the state found all of these projects to be highly successful. The primary lessons learned were that teachers, like their students, learn best from hands-on activities and that languages like MicroWorlds and Alice are excellent teaching tools for beginners because of the ease with which they can create complex projects.

CAMP MODEL

Experiences from these projects led to the design and implementation of the CSCI Wonderland camp that brought both teachers and students at the same time to the

university campus. The camp lasted for five days. During the first four days, activities were scheduled from 8:30 am to approximately 3:30 pm, including an hour-long lunch break. On the last day, activities were scheduled from 8:30 am to 1:00 pm, including an awards banquet and presentation of projects. Each day of the camp included various types of activities:

- Mini Alice Projects: programming exercises involving the development of short animations, typically including concepts and features demonstrated by the instructor,
- Computer Science Challenges: short competitive exercises involving a particular Computer Science concept (binary-decimal conversion, Boolean logic, encryption), and
- Essay Writing: a short essay addressing a particular computing topic presented by one of the instructors.

Through the mini Alice projects, camp participants were introduced to basic concepts such as objects, methods, variables, sequences, repetition, conditional statements, and concurrency. All needed software was included in a USB drive given to all camp participants. In each of the projects, teams were given a sample Alice world and asked to augment or modify the world by including their own statements or method calls. Teams were given significant creative freedom in meeting the requirements of each project. Points were awarded based on effective problem solving, satisfaction of project requirements, artistic creativity, and overall project design. Once evaluated, feedback was given and the resulting animations were shown to all camp participants.

Each of the Computer Science challenges consisted of a short introduction to a computing topic followed by a timed exercise. Exercises included converting numbers from binary to decimal system, determining the output of simple combinational circuits given input values, and finding the shift value used when encoding messages using a Caesar cipher.

On each of the first three days of the camp, short presentations were given on topics such as a brief history of computing, careers in computing, and ethics of the computing profession. Teams were then given a topic, question, or theme to be used in the development of a team essay to be turned in the following day.

Each of the activities was part of an overall competition among the various teams. Points were awarded each day to the teams according to their relative success in the various activities. To ensure that all of the teams were aware of the relative standings, scores were uploaded each day into a publicly available spreadsheet linked from a Facebook group dedicated to the camp.

By the third day of the camp, once the students had been exposed to all the relevant programming concepts and Alice features, the teams were given the main Alice project assignment, which involved the development of an animated story to be presented during the luncheon on the final day of the camp. Although the camp participants were given a list of possible general topics and a minimum set of requirements for the project, each team had complete creative and artistic freedom regarding project design and development. Obviously, since the main project consumed the greatest amount of team effort during the week, it was given the largest point value in the overall competition. As a result, on the last day of the camp, all of the teams were within reach of the overall first

place team pending the result of the main project evaluations. Monetary awards and trophies were awarded to the teams which accumulated the top three point totals by the end of the week.

CAMP EVALUATION

Two evaluation instruments were developed and administered in order to obtain feedback from camp participants. A General Feedback Form was given to teachers and students at the end of the week. In the form, camp participants were asked to rate each of the activities on a scale from 1 to 5. They were also asked to give an opinion regarding funding for the camp, the residential vs. non-residential aspect, and general suggestions for improvement.

A second evaluation instrument, designed to measure students' attitudes regarding various aspects of Computer Science, asked them to express the extent to which they agreed with various statements on a Likert scale from 1 to 5. In order to measure whether participation in the camp had an impact on student attitudes, the survey was given to students during the introductory meeting on the first day of the camp and prior to the awards banquet on the last day of the camp. A summary of results is presented in Table 1.

1 - Strongly Disagree, 2 - Disagree, 3 - Neither agree nor disagree 4 - Agree, 5 - Strongly Agree	PreTest		PreTest	
	Mean	Std Dev	Mean	Std Dev
1. I feel I know what computer programming is all about.	2.25	0.80	3.47	0.88
2. I feel I know what Computer Science is all about.	2.22	0.87	3.34	0.75
3. Computer programming is a difficult process that involves coding complex instructions in a programming language.	3.13	1.01	3.03	1.26
4. People that major in Computer Science spend all of their time programming computers.	2.44	1.13	1.97	1.15
5. Computer Science has little or nothing to do with other areas, such as arts and humanities.	1.97	1.03	1.53	0.84
6. People that study Computer Science learn how computers work internally.	3.84	0.92	3.81	0.90
7. People that study Computer Science learn to solve problems in logical steps.	4.03	0.82	4.22	0.66

8. Computer security professionals must understand how computers work as well as the mathematical details of securing information.	4.03	0.90	4.25	0.62
9. There is a great need for professionals in Computer Science, particularly in the area of information security.	3.84	0.99	4.38	0.66
10. In college, I plan to major in Computer Science or a closely related field.	2.13	1.52	2.69	1.62

Table 1. Summary of PreTest and PostTest Mean Responses to Survey Items

It is worth noting that with the possible exception of item 6, the mean student response after the camp represents an improvement in students' attitudes. In retrospect, this is hardly surprising, since none of the camp activities directly involved the internal workings of a computer. A paired-sample t test was applied to each survey item in order to measure the statistical significance of any change in students' attitudes before and after the camp. Table 2 contains summary statistics for the mean and standard deviation of the individual differences in pretest and posttest scores, as well as the corresponding t values given the sample size of 32 students. Since the corresponding critical t value at the 95% confidence level is 1.6955, the mean difference in pretest vs. posttest ratings is significant for survey items numbered 1, 2, 4, 5, 9, and 10.

	Mean	N	Std Dev	t Stat
1. I feel I know what computer programming is all about.	1.219	32	1.008	6.84
2. I feel I know what Computer Science is all about.	1.125	32	0.976	6.52
3. Computer programming is a difficult process that involves coding complex instructions in a programming language.	-0.094	32	1.748	-0.30
4. People that major in Computer Science spend all of their time programming computers.	-0.469	32	1.319	-2.01
5. Computer Science has little or nothing to do with other areas, such as arts and humanities.	-0.438	32	0.914	-2.71
6. People that study Computer Science learn how computers work internally.	-0.031	32	0.999	-0.18
7. People that study Computer Science learn to solve problems in logical steps.	0.188	32	0.780	1.36

8. Computer security professionals must understand how computers work as well as the mathematical details of securing information.	1.36	32	0.975	1.27
9. There is a great need for professionals in Computer Science, particularly in the area of information security.	0.531	32	1.164	2.58
10. In college, I plan to major in Computer Science or a closely related field.	0.563	32	1.190	2.67

Table 2. Summary of Paired Sample t Test of PreTest vs. PostTest Differences

The significant improvements in student attitudes for items 1, 2, 4, 5, 9, and 10 are consistent with the camp's emphasis on programming, general Computer Science concepts, information security, as well as relationships with the arts and humanities.

SUMMARY AND CONCLUSIONS

A non-residential summer camp was held at our institution in July 2010. Participants included six students and two teachers from each of six area high schools. Students and teachers were introduced to the Alice programming environment and asked to complete a number of small assignments as well as a major project involving an animated story of their choice. In addition, participants were exposed to introductory topics from various areas of Computer Science. Each school team was asked to complete competitive exercises related to each of these topics, as well as short essays on selected topics. Throughout the week, points were awarded to the various teams based on their relative performance in each assignment, with the major Alice project representing the majority of the points.

An analysis of the students' responses reflects a statistically significant improvement in attitudes in six of the areas surveyed. Therefore we conclude the model for a successful computer science camp would include having teachers and students work together, using a beginner-friendly language such as Alice, incorporating a competitive environment, and integrating arts and humanities with computer science concepts.

REFERENCES

- [1] Eaton, V. Magoun, D., Owens, C., IT and the attitudes of middle school girls: A follow-up study, *Proceedings of the National Educational Computing Conference (NECC)*, 2002.
- [2] Eaton, V., Magoun, D., Owens, C., A follow-up study: The attitudes of middle school girls toward technology-related skills, *Proceedings of Site 02 Technology and Teacher Education Conference*, 2002.
- [3] Eaton, V., Magoun, D., Owens, C., Taylor, K., A gender issue: Effecting change in the attitudes toward technology for middle school girls, *Proceedings of the World Conference on Computers in Education (WCCE)*, 2001.

- [4] Eaton, V., Magoun, D., Owens, C., Taylor, K., Scientific and technology learning: Enhancing the attitudes toward technology for middle school girls, *Proceedings of Site 01 Technology and Teacher Education Conference*, 2001.
- [5] Eaton, V., Magoun, D., Owens, C., Smith, B., Bringing teachers into the twenty-first century: A collaboration between a university and its surrounding school systems, *Professional And Organizational Development Network in Higher Education (POD)*, 1998.

MINIMIZING TO MAXIMIZE: AN INITIAL ATTEMPT AT TEACHING INTRODUCTORY PROGRAMMING USING ALICE*

Tebriq Daly
Department of Computer Information Systems
Collin College
Plano, TX 75074
972-881-5838
TDaly@collin.edu

ABSTRACT

Although learning to program can be a daunting task for many students, it doesn't have to be. This paper discusses the possibility of teaching programming concepts in a learning environment that is less threatening. Learning abstract programming concepts and programming in an environment where users are prone to make mistakes can cause students to become frustrated, lose confidence, and refrain from taking another programming course. The purpose of this paper is to compare two online introductory programming courses: one using Alice as a precursor to Java programming and one using pure Java programming.

INTRODUCTION

Coding the syntax of today's popular programming languages can be frustrating for students who are new to programming. To write a simple program in Java for example, you must add extra code just to get the program to print the famous "Hello World" to the screen. The program must be typed with the proper capitalization, pristine spelling, and the exact amount of curly braces, quotes, semicolons, brackets, and parenthesis in the correct order. How can we expect students to gain confidence with programming, if they are constantly receiving compiler errors that are preventing them from executing their programs?

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

BUILDING CONFIDENCE

Bandura [1], in his social cognitive theory, proposed that self-efficacy plays a role in all facets of life. Confidence has the power to affect our motivation, interest, and achievement towards specific tasks [2]. Keller [6] created a powerful problem solving model that was designed to arouse and maintain motivation toward learning. This model is known as the ARCs model and it is composed of four distinct parts: attention, relevance, confidence, and satisfaction [6]. We must gain and uphold the students' attention throughout the learning process, provide students with relevant activities which are clearly identified, raise self-confidence so that students are motivated to continue, and provide students with a means of sense of accomplishment or enjoyment [6]. This paper focuses on the confidence factor of the ARCSs model and how it affects learning.

Since confidence affects motivation, we need to be sure that students are challenged, but that they are not overwhelmed. Students who are motivated become more involved in their learning and therefore are more likely to finish a course [7].

ALICE PROGRAMMING ENVIRONMENT

The Alice environment eliminates the frustration and focus on the syntax of the language, making it easier on students to create animations and/or games. Alice allows the programmer to create code without worrying about semicolons, curly braces, etc. and allows the students to focus on the concepts. The software not only immerses students into a programming rich multimedia environment in which they can feel comfortable manipulating and programming objects, but it truly enforces the object-oriented concepts needed if students are thinking about taking their programming to the next level.

Alice is programmed in Java and designed to give students the opportunity to experiment with objects, classes, inheritance, expressions, conditions, loops, variables, arrays, events, recursion, data structures, and to begin thinking algorithmically. The students are able to create simple animations and interactive environments/games by dragging and dropping graphical segments of code onto an editing area.

Research has shown that Alice provides security, comfort, and motivation towards programming; raises performance rates; and improves retention rates for programming courses [3, 4, 5, 8].

Moskal et al. [8] found that 88% of the at-risk undergraduate computer science students who were introduced to programming via the Alice environment moved on to a second programming course compared with the 47% who did not use Alice prior to programming. Cooper et al. [4] conducted a similar study which showed 91% of the students in the Alice-based CS1, level-one computer science, course continued on to CS2, a second level computer science course, whereas only 10% of the students in the non-Alice based CS1 course continued on to CS2.

To measure the impact of Alice on student attitudes toward computer programming, Courte et al. [5] surveyed one hundred non-computer science majors before and after using Alice; the results indicated that Alice "increased student enjoyment and promoted positive attitudes toward programming" (p. 1). Although Moskal et al.[8] found no

significant difference in students' confidence levels; they did find the creativity levels of the students using Alice to be significantly higher than those not using Alice.

Bishop-Clark et al. [3] found that using Alice for a brief period of 2.5 weeks yielded significant results among 154 students in understanding of programming ($p < .001$). Eighty-nine percent of the students "believed that they had learned the programming concepts and had gained a better appreciation of the complexity of programming" [3] (p. 205). To measure the effectiveness of the Alice environment, Skyes [9] asked students enrolled in a programming course using Alice and a programming course without Alice a series of programming questions; the results showed that the students using Alice scored significantly higher than those not using Alice.

RESEARCH QUESTIONS

The purpose of this study was to compare two online introductory programming courses: one using Alice as a precursor to Java programming and one using pure Java programming. The research questions were:

1. Does Alice improve students' confidence in programming?
2. Is there a difference between using Alice and not using Alice in student confidence levels in programming? If there are differences, what are they?
3. Does confidence affect enjoyment and retention?

PARTICIPANTS

There were a total of 11 participants from the online introductory programming course using Alice as a precursor to Java programming (Treatment Group) and 18 participants from the online introductory programming course using pure Java programming (Control Group). Three of the 11 treatment group participants were female and 4 of the 18 participants from the control group were female. All of the students were undergraduate students attending community college. Some of the participants enrolled in the courses were planning on majoring in computer science, but some were taking the courses as an elective or to meet the requirements of their degree plan. The two courses participating in this study had different instructors, but utilized the same Java textbook and same Java assignments throughout the semester.

METHODOLOGY

The students in the Alice/Java course (treatment group) received six weeks of Alice instruction before moving onto the Java component of the course and the other course (control group) worked with Java the entire length of the course. The treatment group completed three online surveys: one at the beginning of the course, one after completing the Alice portion of the course, and one after completion of the course. Although the control group did not use Alice in the course, they still completed the same surveys at the same time intervals as the treatment group using Alice. The purpose of the surveys was to gauge student confidence levels with basic programming concepts as they progressed through the course. Each of the surveys consisted of ten questions. Each of the questions required a ranking answer from 1 (Strongly Disagree) to 5 (Strongly Agree). The

questions measured confidence levels for each of the following programming concepts: objects, classes, methods, parameters, conditionals, repetition, events, arrays, variables, and arithmetic expressions. The end of the course survey contained both quantitative and qualitative questions relating to student retention (i.e. - I plan to take another programming course after this course). Please see the appendix for exact questions.

RESULTS

The students in the Alice/Java course had a higher level of confidence overall when compared to the pure Java course (Table 1). The students in the Alice/Java course also had a higher confidence level at the end of the course in all of the programming concept categories (Table 4). Table 6 indicates that the Alice/Java group has a significantly greater improvement gain from pre-test to post-test than the pure Java group in the following areas: objects, classes, methods, parameters, arrays, and variables. Table 5 indicates that the students in the Alice/Java course are more likely than the students in the pure Java course to take a programming course in the future even though there were fewer Computer Science majors in the Alice/Java course. Overall the students enjoyed using Alice and found it to be helpful.

	Pre-test		End of Course		Gain
	Mean	SD	Mean	SD	
Alice/Java Course	2.49	0.30	4.38	0.16	1.85
Pure Java Course	2.88	0.29	3.57	0.36	0.69

Table 1 - Total Confidence Levels

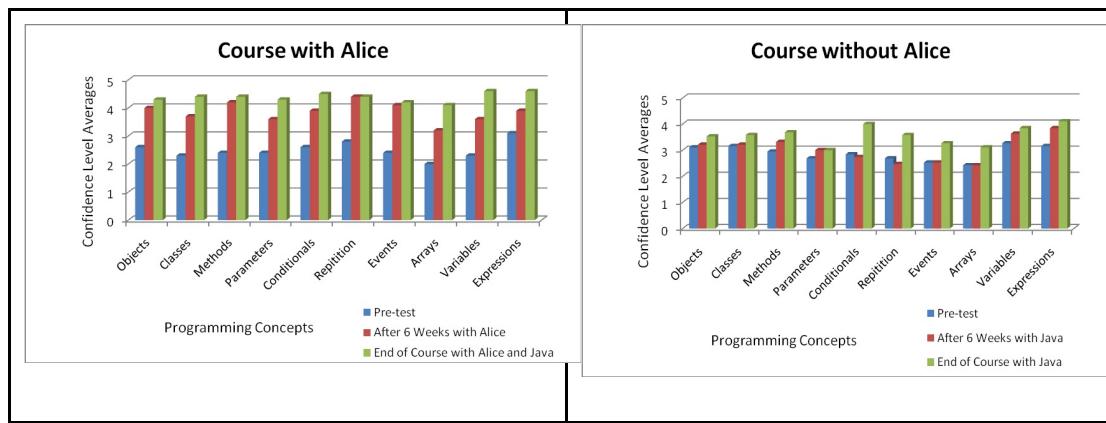
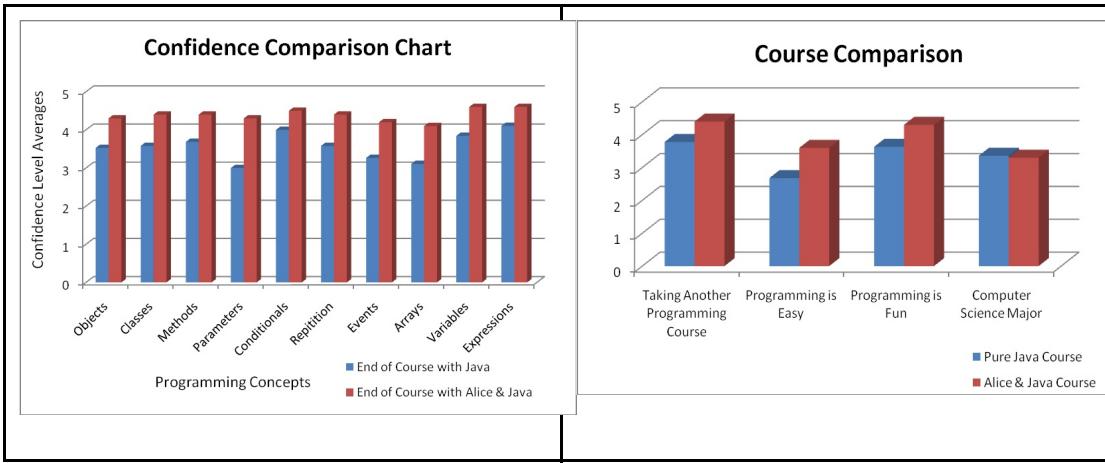


Table 2 - Confidence Levels in Alice/Java Course

Table 3 - Confidence Levels Java Course

**Table 4 - Comparison of Overall Confidence Level****Table 5 - Attitudes Toward Programming**

ANOVA					
		Sum of Squares	df	Mean Square	F
Objects	Between Groups	10.625	1	10.625	5.049
	Within Groups	56.823	27	2.105	
	Total	67.448	28		
Classes	Between Groups	17.722	1	17.722	11.318
	Within Groups	42.278	27	1.588	
	Total	60.000	28		
Methods	Between Groups	9.618	1	9.618	7.111
	Within Groups	36.520	27	1.353	
	Total	46.138	28		
Parameters	Between Groups	13.266	1	13.266	7.434
	Within Groups	48.182	27	1.785	
	Total	61.448	28		
Conditionals	Between Groups	1.742	1	1.742	.954
	Within Groups	49.283	27	1.826	
	Total	51.034	28		
Repetition	Between Groups	1.777	1	1.777	1.006
	Within Groups	47.672	27	1.766	
	Total	49.448	28		
Events	Between Groups	5.033	1	5.033	3.811
	Within Groups	35.657	27	1.321	
	Total	40.690	28		
Arrays	Between Groups	9.618	1	9.618	5.582
	Within Groups	46.520	27	1.723	
	Total	56.138	28		
Variables	Between Groups	14.951	1	14.951	7.886
	Within Groups	51.187	27	1.898	
	Total	66.138	28		
Expressions	Between Groups	.903	1	.903	.416
	Within Groups	58.545	27	2.168	
	Total	59.448	28		

Table 6 - ANOVA using gains from pre to post

CONCLUSION

The goal of Alice is to provide users with a comfortable programming environment that is motivating and improves confidence with programming concepts. Other studies have proved Alice to be enjoyable, to promote positive attitudes toward programming, to raise motivation, and to improve retention (4, 5, 8). This study supports the above claims. It seems that Alice does improve students' confidence levels toward

programming. The students in the Alice/Java course had higher levels of confidence at the end of the course in all of the categories when compared to the course using pure Java programming. Even though this study had a fairly small sample size, the results showed the Alice/Java group to have a significantly greater improvement gain from pre-test to post-test than the pure Java group in the following areas: objects, classes, methods, parameters, arrays, and variables. This study was an initial attempt to measure the effect of Alice on confidence levels. The following comments were made by students in the Alice/Java course.

"I thought that Alice made OOP easier to understand with its visual 3D format. This further assisted me in learning Java."

"It was really fun. It was a good way to teach concepts before the actual coding."

"I think it definitely helped the transition into Java, would recommend it to anyone starting out with programming."

REFERENCES

- [1] Bandura, A. (1993). Perceived self-efficacy in cognitive development and functioning. *Educational Psychology*, 28:117-148.
- [2] Bandura, A.; Barbaranelli, C.; Caprara, G.; Pastorelli, C. (1996). Multifaceted impact of self-efficacy beliefs on academic functioning. *Child Development*, 67:1206-1222.
- [3] Bishop-Clark, C., Courte, J., Evans, D., & Howard, E. (2007). A quantitative and qualitative investigation of using Alice programming to improve confidence, enjoyment, and achievement among non-majors. *Journal of Educational Computing Research*, 37 (2), 193-207.
- [4] Cooper, S., Dann, W., & Pausch, R. (2003). Teaching objects-first in introductory computer science. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, USA, 191-195.
- [5] Courte, J., Howard, E., & Bishop-Clark, C. (2006). Using Alice in a computer science survey course. *Information Systems Education Journal*, 4 (87), 1-7.
- [6] Keller, J. M. (1987). Development and use of the ARCS model of motivational design. *Journal of Instructional Development*, 10(3), 2-10.
- [7] Militiadou, M., & Savenye, W. (2003). Applying social cognitive constructs of motivation to enhance student success in online distance education. *AACE Journal*, 11(1), 78-95.
- [8] Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, USA, 75-79.
- [9] Sykes, E. (2007). Determining the effectiveness of the 3D Alice programming environment at the Computer Science I level. *Journal of Educational Computing Research*, 36(2), 223-244.

APPENDIX**Survey 1, 2 (Pre-test, After 6 weeks)**

The goal of this survey is to measure confidence levels of various programming concepts throughout the semester. This is an anonymous survey and will have no impact on your grade for the course. Please answer the following questions honestly so that this survey can be used to improve this course for the future.

Gender _____ Age _____

How much programming experience did you have before taking this class? _____

		Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree
1	I feel confident using object to program.	1	2	3	4	5
2	I feel confident using classes when programming.	1	2	3	4	5
3	I feel confident using methods to program.	1	2	3	4	5
4	I feel confident using parameters to program.	1	2	3	4	5
5	I feel confident using conditionals to program.	1	2	3	4	5
6	I feel confident using repetition to program.	1	2	3	4	5
7	I feel confident programming events.	1	2	3	4	5
8	I feel confident using arrays.	1	2	3	4	5
9	I feel confident using variables.	1	2	3	4	5
10	I feel confident building arithmetic expressions.	1	2	3	4	5

Survey 3 (End of Course) *Note: The pure Java course did not receive the Alice questions.*

The goal of this survey is to measure confidence levels of various programming concepts throughout the semester. This is an anonymous survey and will have no impact on your grade for the course. Please answer the following questions honestly so that this survey can be used to improve this course for the future.

Gender _____ Age _____

How much programming experience did you have before taking this class? _____

		Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree
1	I feel confident using object to program.	1	2	3	4	5
2	I feel confident using classes when programming.	1	2	3	4	5
3	I feel confident using methods to program.	1	2	3	4	5
4	I feel confident using parameters to program.	1	2	3	4	5
5	I feel confident using conditionals to program.	1	2	3	4	5
6	I feel confident using repetition to program.	1	2	3	4	5
7	I feel confident programming events.	1	2	3	4	5
8	I feel confident using arrays.	1	2	3	4	5
9	I feel confident using variables.	1	2	3	4	5
10	I feel confident building arithmetic expressions.	1	2	3	4	5

		Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree
1	I plan to take another programming course.	1	2	3	4	5
2	Computer programming is easy.	1	2	3	4	5
3	Computer programming is fun.	1	2	3	4	5
4	I am planning to major in Computer Science.	1	2	3	4	5
5	I found Alice to be helpful.	1	2	3	4	5
6	I enjoyed using Alice.	1	2	3	4	5
7	Alice helped me to understand programming concepts.	1	2	3	4	5

Comments:

HOW TO DELIVER A GENTLE INTRODUCTION TO PARSING*

CONFERENCE TUTORIAL

David Middleton

*Dept. of Computer and Information Science,
Arkansas Tech University,
201C Corley Building,
1811 N Boulder Ave. Russellville, AR 72801
(1 479) 968 0628
dmiddle@cs.atu.edu*

JUSTIFICATION FOR TEACHING PARSING

This tutorial shows how to introduce parsing to students, so that they can apply well-known tools to build systems that will organize a sequence of input elements into an appropriate structure.

'Why teach parsing, since no-one builds compilers anymore?' Two particular reasons stand out. One reason is pedagogical. Midway through their degree, students take a data structures course where they encounter a variety of intricate algorithms and data structures - tree balancing, hashing, graph algorithms - but then most or all of their upper courses use only simple ones. Parsing inevitably requires intricate algorithms, motivating the students' appreciation for these techniques.

Second, while the complexity of processing data exceeds that of acquiring it, during their early experiences, students eventually reach a point where some problems are essentially reading input in multiple forms, such as for legacy systems. In these cases, parsing, finding patterns in input provides the basis for a properly designed, robust, solution.

The challenge for computing faculty is to present this intricate material in a fashion that the students can understand well enough, that they can apply tools to build their parsing system. The purpose of this tutorial is to show how standard parsing techniques (specifically, for LL(1), SLR(1), LALR(1), and LR(1) grammars) can be presented, so that they are easily grasped by reasonable students.

The following description of the proposed tutorial presumes the audience is vaguely familiar with parsing techniques, and presents the order in which material would be covered in a classroom setting. This presentation has been used for more than a decade, as the start of our compulsory Compiler Design course, to motivate the students for subsequent, more detailed, learning. The tutorial will not only show how the material can be presented to students, but will also provide a refresher course for instructors who

* Copyright is held by the author/owner.

attend. As an additional bonus if time allows, clearer algorithms for calculating several important grammar properties (most notably, First and Follows) will be presented [1].

THE STAGES IN PRESENTING PARSING

Explaining the constructing of parsers is broken into three broad stages: for us to tell the computer how to find structures, we must already know how to do it ourselves, and for us to find structures ourselves, we need a precise definition of how the structures may be formed.

Hence, presenting parsing to students involves this series of steps, showing how grammars define structure, showing how derivations determine the structure of a particular (legal) input, and finally how we form an algorithm to automate such a structure-discovering activity.

With loose examples from natural language, grammars can be demonstrated by example. The second step, manual discovery of structure, is based on the idea of derivations, with emphasis on making judicious choices among alternative grammar rules, so that the desired input will be found (assuming it is legal).

The third step, automating the derivation building activity, is further divided into stages.

First, top-down and bottom-up parsing are viewed as problem solving activities. The problem is to find a certain structure in the input. In top-down parsing, finding a larger structure, such as the sentence, is decomposed into finding smaller parts, until at the lowest level, the system is finding individual tokens. Along the way, we record (in a stack) those component structures, large and small, that are yet to be found. Top-down parsing initially needs to find one complete sentence, and eventually needs to find nothing more.

Bottom-up parsing combines previously-found smaller components into new larger components. Along the way, we record those components that have already been found. Bottom-up parsing initially has found no components, and eventually will have found one complete sentence (assuming it was working with legal input).

We use a specific nonsense grammar that helps illuminate several issues that arise in the running of these parsing activities; for example, left recursive grammar rules prevent (typical, non-backtracking) top-down parsing from being feasible.

After this, we use this grammar to illustrate how we would build a machine for bottom-up parsing (in fact, the characteristic finite state machine), so our second stage (in Step 3), is to make a couple of convenient extensions to our algorithm. Our stack of symbols (recording what structures have already been seen) is initialized with a symbol representing an empty stack. The use of such sentinel values is probably already familiar to students. Also, we augment the symbols on the stack with information about which rule they will be contributing to, and so introduce and motivate the use of items (or, configurations) as the elements of the stack.

The third stage (in Step 3) involves building the CFSM for the grammar. Our sample grammar illustrates how Shift/Reduce and Reduce/Reduce conflicts can arise, and shows

how knowledge of subsequent input symbols may guide the choice. With this, we can then proceed to distinguish LR(1), LALR(1), SLR(1) and LR(0) parsing methods.

As time allows, we will present alternative algorithms for calculating First and Follows sets. Graph algorithms are available which are much easier for students to understand, use, and remember, than the standard algorithms described in textbooks. These algorithms have the additional advantage that they allow the easy creation of a variety of proper grammars which can be assigned to students for homework and exam questions.

SUMMARY

This tutorial shows how top-down and bottom-up parsing can be introduced to students without requiring much previous knowledge of computing material.

By the end, students will have seen grammars (CFGs), the parsing process using a stack (top down and bottom up), and will have seen the characteristic finite state machines for LR(0) and LR(1) grammars. The need for First and Follows sets will have been justified, together with the differences among LR(0), SLR(1), LALR(1) and LR(1) with respect to resolving conflicts.

After four class periods spent in this, they are capable and motivated to engage the material for themselves and build parsers using grammar generators.

- [1] Morell, L., Middleton, D., Using Information Flow to analyze Grammars, *Journal of Computing Sciences in Colleges*, 25, (5), 21-26, 2010.

THE FIRST WEBPAGE DESIGN*

NIFTY ASSIGNMENT

*Valerie Chu
LeMoyne-Owen College
807 Walker Avenue
Memphis, TN 38126
(901) 435-1378
valerie_chu@loc.edu*

The experience of accessing individual student personal web pages via the Internet is an important course objective of Introduction to Micro Computers. The purpose of the proposed assignment is to lead students through the experience of creating web pages which are publicly accessible from the Internet. From this assignment, students learn how to design a frame page with two columns, create an index page, produce images via peripheral devices, arrange materials by tables, and embed internal and external hyperlinks into web pages. Through the final step of uploading files to a server, students learn about client-server relationship. The assignment also demonstrates how directory tree structure files are transferred from a local PC to a server via the communication program SSH Secure Shell.

The course Introduction to Micro Computers emphasizes how software programs can be applied to real-life tasks. Instead of teaching the HTML language, the course relies on the software program Microsoft Expression Web to complete the assignment. The Expression Web program facilitates the procedure of arranging web page materials via a "what you see is what you get" (WYSIWYG) interface. The procedure utilizes an existing frame page template provided by Expression Web to organize a web-site.

One of the advantages of this assignment is that it reinforces the learning objectives of word processing which was covered immediately before the webpage design assignment. For example, students can create tables without borders in order to line up a list of items. They can apply word formatting skills to a web page environment. Moreover, students are expected to use a scanner or a camera to create images, which are ready to be inserted into a web page document.

This assignment is also very flexible to suit individual students' preferences. Many students spend extra efforts on their own Web sites to impress their peers. The suggested items on a Web site include, but are not limited to, an index page, a welcome page, a class schedule, a resume, a family page, and a friends page. Although their Web sites are static, this assignment fully utilizes the concepts of internal and external hyperlinks to link to the students' own pages and other people's sites.

* Copyright is held by the author/owner.

The assignment does not require the purchase of additional hardware devices. Any used desktop computer can be set up as a server to store student web pages. The server installed for my class is a used Dell computer (Pentium 4 CPU, 3.40GHz, 1GB memory, 160GB hard disk) using Ubuntu Linux 10.04 operating system.

Supporting files: <http://www.cscs-ms.org/nifty/11/chu/>

GENERATING PRONOUNCEABLE NONSENSE WORDS*

NIFTY ASSIGNMENT

Mark Goadrich

Mathematics and Computer Science Department

Centenary College of Louisiana

Shreveport, LA 71104

318-869-5194

mgoadric@centenary.edu

Audience: CS1/CS2

Concepts: Map Interface, File I/O, Probability Distributions, Markov Chains

Materials: English Word List, Latin Word List

This assignment explores the use of the Map interface to generate pronounceable nonsense words, similar to those found in "The Jabberwocky" by Lewis Carroll, using three probabilistic models of word creation that become progressively more accurate. Sample generated words can be seen in Table 1. Students were able to finish this assignment within a 3-hour lab session, and enjoyed thinking of definitions for their generated nonsense words.

As a first **Naïve** method, we start by selecting characters from the alphabet uniformly at random. To determine how many letters will be in the word, we write a function to calculate a histogram of word lengths based on a comprehensive list of words in the English language, using a Map with integer lengths as keys and the frequency counts as values. A second function then takes this histogram as a parameter and returns a word length proportional to the histogram counts, demonstrating how to transform a uniform into a proportional distribution.

The words generated naively above are still unpronounceable. Our second **Letter Freq** approach is to change the letter selection from uniform to proportional as well. A new function is used to calculate a histogram Map with letters as keys and letter frequency counts as values. The proportional probability function is then used twice, once for length, and then for letters, to generate the words.

But to make the words pronounceable, we need to care about the arrangement of the letters along with their frequency. To do this, we create a **Markov Chain** to generate the next letter in a sequence based on the previous letters. The **order n** of the Markov Chain is how many of these previous letters to use. A Map is created with the previous n letters

* Copyright is held by the author/owner.

of a word as the key, and a second Map as the value. These inner Maps are similar to the previous letter frequency maps and can be used in the same way to generate the next letter. If we include "\n" as a possible character, our words will naturally terminate. A Markov Chain of order 3 will produce the pronounceable nonsense we desire. As a final task, students can substitute the English word list with Latin to generate nonsense spells, such as "patuscus immoveo" or "stulor oportus."

Table 1: Generated Nonsense Words

Naïve	Letter Freq	Markov Order 1	Markov Order 2	Markov Order 3
bskodahi	toetnpasseil	bunces	roite	bristical
twnszwrde	pouinnepdfsgnw	gurionlystsmatlerc owaris	bacting	immony
hqikxlgiv	nstilnn	eoins	statophorial	glometributted

More information can be found at <http://www.ccsc-ms.org/nifty/11/goa/>

ELECTRIC BUGLE *

NIFTY ASSIGNMENT

Carl Burch

Hendrix College, 1600 Washington Ave, Conway AR 72032

(501) 450-1377

burch@hendrix.edu

When teaching about logic circuits in a sophomore-level course on computer organization, we assign students to use breadboards with TTL integrated-circuit chips. It is true that some students find the intricacies of wiring quite bothersome, and we could use logic circuit simulation software such as Logisim. But many other students find that working with physical circuits is a nice change of pace from doing assignments entirely on a computer. It also ensures that students realize that the logic-circuit concepts they are studying are real in the sense that one can actually build working circuits with them.

It is difficult, though, to find projects that are small enough to be built by students within a week but still do interesting things. We have settled on a project of developing an electric bugle, which features four buttons, corresponding to the four tones commonly played by bugles. These four tones (low G, C, E, and high G) can be used to play a wide variety of well-known bugle calls, including Taps, Reveille, First Call, and Mess Call.

The basic idea is simple: The digital trainer we use is capable of generating a 2 KHz square wave, and we show the students how to wire a basic buzzer. The circuit the students build divides the square wave by 12, 9, 7, or 6 depending on which button is pressed; if none are pressed, the square wave must not be sent to the buzzer at all. (Technically, the E should be generated by dividing the wave by 7.2 - but that's not feasible with an integer counter, and a more precise tuning would require more than four bits, which is the provided 74LS161 counter IC can handle.) The overall circuit requires the counter IC as well as several simple logic-gate ICs. The full assignment description can be found at:

<http://www.ccsc-ms.org/nifty/11/bur/>

One frustrating piece of the assignment is that the square wave goes very quickly, so debugging the circuit can be difficult. Consequently, the students often rely heavily on the instructor for debugging - though it usually doesn't take too much work to determine the problem. A fair fraction of students manage to complete it on their own, though.

* Copyright is held by the author/owner.

The assignment depends on purchasing a kit of electronics supplies for each student. The kit we use, detailed on the Web page, should cost less than \$200 per student. The most expensive piece by far is Kelvin's Laptop Digital Trainer.

In our course, this is the second and final assignment using the breadboard. The first is a more traditional, simpler assignment just to get acquainted with the trainer, ICs, and breadboards. After the electric bugle, the course moves to higher levels of the machine, leaving logic circuits. And this is just as well: Building more complex circuits on breadboards would introduce difficulties beyond the course's learning goals.

CREATING A PREDICTIVE MODEL FOR PARKING ON CAMPUS*

NIFTY ASSIGNMENT

*Janet Renwick,
University of Arkansas - Fort Smith*

Students often complete assignments from which faculty hope they learn specific techniques and concepts. However, it is more difficult to design an assignment that involves the entire problem solving process. In a data warehousing class, a group of juniors and seniors completed a comprehensive project that created a statistical regression model to predict the number of cars in university parking lots. This data, and the model developed, could then be used to identify potential solutions to parking congestion on our campus.

To begin, the class hypothesized what factors they thought might influence the number of cars. They came up with a list that included the weather (temperature and precipitation), the number of students in class at that time, the presence of special events on campus, the time of day, the day of the week, and the week of the semester. Students had to determine how to collect data for each of these independent variables and for the dependent variable, the number of cars.

In the class of ten, two students accepted leadership positions. The students decided to limit the scope of the project to a small group of parking lots and set four times during the day, five days per week, for which they would collect data. One committee developed a data collection sheet, including maps of the parking lots, and sign-ups ensued. It was more difficult than anticipated to get good data. Students discovered confusion over lot names and missed assigned times. The leadership team implemented an audit procedure to ensure data accuracy.

Another student designed an entry interface so that the data collected from the sheets could be entered into a data table. A student whose schedule made data collection difficult volunteered to take the lead on entering the raw data.

A group of two students tackled the "how many students are enrolled at that time" task. The data was taken from the on-line schedule of classes, but summarizing it proved to be much more involved than the students anticipated. Some classes only meet on campus for a few weeks and then move off campus (nursing). Class terms varied widely.

* Copyright is held by the author/owner.

The students developed a series of rules to deal with each of these situations - a data cleansing effort - and the data was processed.

After eight weeks of data collection, the class moved into an analysis phase. Regression techniques were applied, and a series of models developed. The final model found that enrollment, the day of the week, the time of the day, special events on campus, and the temperature were significant predictors of the number of cars in the selected lots. Surprisingly, the presence of rain did not prove to be significant, nor did the week of the semester. (Students felt strongly that the first week of the semester would have shown higher numbers of cars, but our data did not include that time period.) Similarly, we found that higher temperatures were associated with an increase in the number of cars. Students had hypothesized that warmer weather would lead to fewer cars.

This experience allowed students to apply statistical techniques in a real-life situation. They used a variety of tools and techniques to integrate data from a variety of sources. They learned research skills, applied statistics, and learned to work together as a team.

GAMERS CONVENTION MODEL*

NIFTY ASSIGNMENT

*Donna Wright
College of Science, Technology, Engineering & Math
University of Arkansas-Fort Smith
Fort Smith, Ar. 72914
(479) 788-7903
dwright@uafortsmith.edu*

INTRODUCTION:

This assignment is suitable for entry level data modeling students in their first data modeling class. It is assigned at the end of the term when students have a solid understanding of database design concepts. The assignment is devised to integrate entry level data modeling topics into an end-of-term project utilizing a gamers' convention theme to pique the student's interest. This assignment can be found at <http://www.ccsc-ms.org/nifty/11/wri/>

ASSIGNMENT OVERVIEW:

This assignment includes two parts - a classroom discussion followed by an individual homework project. The in-class activity involves students in a discussion about popular computer games and gamers' conventions with the goal of engaging them in an in-depth discussion of the process of staging a gamers' convention, proposed business rules that might define the activities and related data requirements.

The individual assignment requires each student to draw from the classroom discussion to create a case scenario assuming his/her own company will stage the convention. The student is challenged with writing business rules, creating normalized tables along with a conceptual, logical and physical database design to support their version of the company staging the gamers' convention.

LEARNING OUTCOME:

Using a game related scenario serves as a great "hook" for entry level IT students and keeps them interested as they work together to envision the activities required to bring such a convention to fruition. Creativity plays a role as each student decides on the details of his/her own convention and finalizing the project deliverables (normalized

* Copyright is held by the author/owner.

tables, conceptual, logical and physical database design) allows the student to apply the design concepts and skills that have been developed throughout the term.

SNOWFALL TO INTRODUCE SYSTEM SIMULATIONS *

NIFTY ASSIGNMENT

Matt Brown

Department of Computer and Information Science

Arkansas Tech University

Russellville, AR 72801

(479) 356-2161

hbrown11@atu.edu

A visual simulation of snowfall provides a nifty way to introduce students to discrete system simulations. In this assignment, students are given an intentionally unrealistic snowfall program and told to make it more lifelike by correctly choosing appropriate statistical distributions to insert into the simulation. The assignment was originally given in a senior level discrete system simulations course after introducing numerous statistical distributions to determine if students could correctly apply the distributions.

Several variables govern the appearance of a simulated snowfall. These factors can include the size and shape of the snowflake, the location the snowflake appears, the rotation of the snowflake, and horizontal motion as the snowflake falls. To create a snowfall that appears natural (i.e. no two snowflakes the same, random motion, etc.) statistical distributions must be correctly applied. Therefore, students would be expected to have a basic knowledge of various statistical distributions and their use before being given this assignment. At a minimum the students need to be familiar with the normal, Poisson, exponential, and uniform distributions.

The assignment is completed using spreadsheets and Visual Basic macros. Formulas for generating values from appropriate distributions are selected and placed by students into a range of cells within the spreadsheet. These simulated values are then used as inputs controlling the appearance of the snowfall. Using this platform allows students to quickly and interactively examine the impact of different distributions on the simulation.

Almost all of the students were able to create realistic snowfall. Student feedback, both during and after the assignment, was positive. The assignment provided a memorable challenge while demonstrating the importance of correctly applying simulation techniques. The full assignment description and supporting files can be downloaded at <http://www.ccsc-ms.org/nifty/11/bro/>.

* Copyright is held by the author/owner.

COLLABORATION PROBLEMS IN CONDUCTING A GROUP PROJECT IN A SOFTWARE ENGINEERING COURSE*

*Priyatham Anisetty, Paul Young
Computer Science Department
University of Central Arkansas
Conway, AR 72032
501-450-3462*

priyathamanisetty@gmail.com, pyoung@uca.edu

ABSTRACT

One of the largest problems in conducting a group project is establishing a collaborative environment among the members of the teams that are working on the project. A search was conducted to identify specific tools for the different phases of the software development process that might enable students to work better together collaboratively in hopes of enhancing their performance. In this paper the importance of having a group project in the software engineering course, problems related to working together collaboratively and proposed solutions to overcome those problems are discussed.

INTRODUCTION

The main aim behind software engineering courses in universities is to teach students how to learn and understand the importance of each phase of the software life cycle, project management, and also the software process so that students can design and build modern software systems in the future successfully. There are several different patterns for teaching software engineering. One among them is including a group project in the curriculum of the course. The main objective of having the group project in the course curriculum is to provide students with some experience in several different aspects of working in a team including conducting meetings, making group decisions, managing groups, writing reports, and making presentations. This requires proper communication

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

and collaboration among the teams and further among all the team members. This paper focuses on suggesting proper tools that would help teams to manage the tasks better.

This paper is structured in the following manner. First background is provided on the Problem-Based Learning (PBL) technique and the need for a group project. Next are presented problems associated with collaboration on a large scale group project along with potential means for addressing those problems. Finally, conclusions on the proposed collaboration solutions are drawn and recommendations made for future work.

BACKGROUND

By having a group project, students can have real time experience as a software engineer. Including a group project in the course curriculum is based on a learning technique called PBL (Problem-Based Learning).

PBL has its origin in the late 1950's. PBL is different from traditional learning techniques. In PBL, first the student is introduced to the problem, and then the student should work on the problem and come up with a solution. This is opposed to traditional learning where the student is provided with the material and then tested on that material. By following this traditional method students cannot gain practical knowledge, which is very much important in a course like software engineering. "A learned concept is much more valuable if we know how to apply it to a real problem". [1]

PBL is a student-centered instructional strategy in which students try to solve problems collaboratively. It was pioneered at McMaster University, Canada [9]. The main characteristics of PBL are:

- Learning is driven by challenging, open-ended, ill-defined and ill-structured problems.
- Students generally work in collaborative groups.
- Teachers take on the role as "facilitator" of learning.

Several medical schools have incorporated problem-based learning into their curricula, using real patient cases to teach students how to think like a clinician. More than eighty percent of medical schools in the United States now have some form of problem-based learning in their programs [9]. This problem-based technique can also be used in teaching a software engineering course where the teacher will serve as facilitator and students will be working in groups to develop a group project.

Small scale assignments will not provide the students with the complete experience of learning software design principles or how they should be applied in the real world [2]. This means in software engineering there is a gap between theoretical knowledge and experimental knowledge which needs to be filled. The gap between theoretical knowledge and practical knowledge needs to be filled and students provided with hands-on real-time experience, so that they can excel in their career as good software engineers. In order to bridge this gap an opportunity needs to be provided for students to work in teams that is apart from normal teaching of design principles. The main objective of having teams within the classroom is to provide the opportunity for students to learn good engineering practices [5].

When students were exposed to real-world pressures there were many situations where they did not know how to take advantage of the techniques which they learned in their courses and solve the problems in the real world [2]. Students found many of the principles which they have learned as obstacles rather than tools to solve the problems. This is resulting in students having a slow start in the industrial projects [2]. There should be something more in the course curriculum. This can be solved by having a project in the curriculum, which allows students to work in groups.

IMPORTANCE OF HAVING A GROUP PROJECT

In teaching software engineering there is vital need for doing a group project. The main motivation behind doing a group project in the software engineering course is to prepare the students well in advance in such a way that they meet industry requirements. Especially in courses like software engineering, it is really difficult to teach traditionally and have students understand about the rigors such as changes to the requirements and software design that software may go through, which are the heart of software engineering [6].

By having a group project in the course the students need to work in a team rather than individually. This will allow them to understand different program parts separately, how to split the whole project into sub-tasks, and how to work collaboratively. This will allow them not only to come across the difficulties associated with collaboration on a large scale group project but also to learn how to overcome these difficulties successfully and develop the final project.

First, by having a group project students will go through the entire software development by which they can gain real time experience in the class, which will provide them with the skills needed to perform well as a software engineer. By having a group project students will not only taste the software development process, but they will also learn how to respect others' view-points in decision making, manage their own time, and also understand software ethical issues [10]. This will also allow students to understand how they should behave in teams, how they can contribute to their teams, and how to work collaboratively.

Second, doing a group project will introduce students to different case tools that were used in developing software collaboratively; they will learn how to use them and gain familiarity with the tools. It will help them to get more from the tools.

A group project requires all the students to work in teams. Teams can be created in two fashions: one option would allow students to form their own teams; the other option is the professor should divide the teams. There are pros and cons with both options. In the first option there is a danger of having uneven splitting of stronger members that may result in two weaker members working together, which is not good as proven in psychological studies [5]. This problem can be overcome if the instructor divides the teams evenly so that they are balanced.

Once the teams are formed and the project is decided then it is the time to start working on the project. Until this time the facilitator (teacher) is involved in each and every step i.e. creating groups, deciding the project etc. But after deciding the project and once the teams are divided students will start working on the project, and then will come

across new problems in "Team collaboration" where they need to work as part of the group rather than individually.

COLLABORATION PROBLEMS AND PROPOSED SOLUTIONS

Software engineers need to work together in teams to achieve a specific goal. This requires students to learn to cooperate with each other while working on their part of the work [7]. In developing large software projects, engineers are typically not situated in one location. Instead, many projects are geographically distributed; they may be distributed within the country or internationally over multiple countries [7]. This means that engineers will be working collaboratively from different locations. When working on the group project in the software engineering course, all the students were present in the same school; still there is a problem of collaboration between them because students will not be working on the project in the classroom but they will be working on it from outside of the class. So they were working in a kind of distributed environment, although they were meeting in the classroom for meetings and presentations. Next the problem of providing team collaboration during an object-oriented development is discussed. Problems encountered and proposed solutions while developing the requirements, specification, design, and implementation will be offered.

Collaboration During the Requirements Phase

In the past, collaboration on a document would involve passing a document back and forth between authors. Each author would take a turn at improving the work, often correcting, modifying, or building on the work of the other authors [4]. This presents a problem as the authors are not allowed to work in parallel. One has to wait until the other author completes his total modifications before passing the document. So here some tools are needed which have the capability of allowing users to work in parallel in a collaborative fashion.

Gathering the requirements of the project encompasses determining the needs of the product. In the real world the development team will be in touch with the stakeholders and make sure that the requirements are satisfactory to their clients. In the software engineering class the group project may not always have real clients. In these cases the facilitator will act as client and students need to work with the facilitator to gather the requirements of the project. After gathering the requirements the students need to develop the requirements document. When preparing this document all of the students are taking part. It will be difficult for students to prepare the documentation for their specific part in the document and then combine all the pieces to create the main document.

Any tool that allows interaction on a shared resource has the potential to be a collaboration tool. There are some web-based tools by which documents can be shared, opened, and edited by multiple users at the same time. By using these tools students can work on their documents collaboratively from different locations. By using online collaboration tools, changes can be tracked and the document can be shared with a larger group of students. Some of the tools that provide these facilities are Google Docs, Peepel, Buzzword and Thinkfree [4].

Google Docs was selected from the many document-sharing services that are available online. There were many other online document sharing tools which require user fees but Google Docs are free. So, students can make use of them in creating documents for the project. Google Docs provides the user with word-processing, spreadsheets, presentations and drawing tools. Unlike normal office solution software, the work is conducted online and all users can work together at the same time on the document in real time. Google Docs also has the ability to track changes. In addition it comes with version-control features which will be helpful in case of loss of information.

For the requirements phase, a large portion of the required deliverables are text-based. Google Docs provide excellent collaboration facilities for handling these types of documents. For non-text based documents such as use case diagrams the next section will provide alternative collaboration tools. By using Google Docs, every student can know what other students are doing on their specific part of the document. Every student will be familiar with the document by having the capability to see the overview of the document that is being developed.

Collaboration During the OOAD Phase

Creating the text based portions of the specification document involves similar issues as seen in the creating the requirements document. Everyone needs to participate in developing the document. Here you can also make use of Google Docs to create the document by working collaboratively.

One of the requirements in the analysis phase is to develop class diagrams for the use cases completed in the requirements phase. There is a chance to create duplicate classes if there is no proper collaboration between the teams. There is also a need to make sure that all the classes are consistent. In order to overcome these problems one can make use of the revision control tool.

Revision control systems use a centralized model where all the revision control functions take place on a shared server. These systems have the capability to manage files and directories, and the changes made to them over time [3]. This allows you to recover older versions of your data or examine the history of how your data changed. There are many revision control tools that are available. Some of them are CVS, Subversion, and VP team server.

Apache Subversion is a software versioning and revision control system founded and sponsored in 2000 by CollabNet Inc. [7]. Developers use Subversion to maintain current and historical versions of files such as source code, web pages, and documentation. One can create a Subversion repository for their project; there are many paid sites as well as free sites providing this facility. This Subversion repository can be connected with a UML authoring tool. There are many UML tools available online, some of them are Visual Paradigm, Eclipse UML, and UML Studio. There are many other commercial and non-commercial UML tools available online.

In the analysis and design phases the object-oriented system design is completed, such as creating the Class diagrams, CRC cards, state chart diagrams, and interaction diagrams. Consistency in visual representation of these diagrams must be maintained. In the classroom visual paradigm has been chosen as the UML tool. It comes with specific

features like a synchronization engine for generating and updating the various diagrams. If any diagram is modified in the project then visual paradigm updates that particular class in the complete project. VP UML provides team collaboration capabilities such as VP team work server, Subversion, perforce, and CVS for designing projects collaboratively. As discussed earlier you can make use of Subversion's repository capability.

By using the UML tool one can connect to the Subversion repository. When a user or a developer connects to this Subversion repository he will be provided with all the data that is already present in the repository. He can look at what all others in the project are doing and therefore reduce the probability of creating duplicate diagrams. If someone in the project wants to add a particular functionality to the project, he can look at the code present in the version control tool and easily decide whether to create a new diagram or just update already existing diagrams by adding this functionality. After adding a new diagram or updating an already existing diagram the user should commit the work he has done and update the repository. Then the updated version is stored in the repository.

When the developer comes back to their computer to work on the project they will update the project that is already in their computer. After updating they will know that a particular diagram has more functionality or a new diagram has been added to the project. By using Subversion you can have collaboration and have consistent class diagrams. The last task in the design phase is to create the detailed design for the classes. Being text-based, these can be handled similarly to the text-based documents in the requirements phase.

Collaboration During the Implementation Phase

When the developers complete writing their Program Description Language (PDL) for the project, the focus shifts to implementation. "The implementation phase takes the requirements and design phase products and implements them using appropriate technologies [8]." Here the developers are expected to implement the code for the project. So the developers who are collaborating on a project require a tool and an infrastructure that can help them stay connected to each other and work together as a team [3]. There are many Integrated Development Environments (IDE's) that provide the capabilities for developers to work together. Some IDE's are Eclipse, Netbeans, Borland JBuilder, and Macromedia Dreamweaver.

By adding Subversion to Netbeans a repository can be provided for the project's code. Developers can connect to this repository also using Netbeans and download the code present in the repository. The repository should also contain test cases by which a user can test his code functionality. After testing his code with the test cases the developer will upload his code to the repository and commit the operation. Once every developer uploads their code then it is integrated into one working product.

One problem discovered with the integrated version contained in the NetBeans IDE was the lack of a mechanism to enforce the ordering by which new code was added to the repository. A user could randomly commit changes, sometimes breaking already working builds. Enforcement of the order in which changes were committed was a manual process, left totally to the team leaders and project manager.

CONCLUSION

In conclusion it was found that by using different tools like Google Docs, Subversion, Visual Paradigm, and Netbeans you can provide better collaboration among teams working on a group project. These tools were used for the first time in a senior-level software engineering course for developing the group project. Better results can be obtained if correct planning for the project is done and the tools chosen before starting work on the project.

If all students in the class are introduced to the tools before the start of the project work, they can get more benefit from the tools. Because the students will be familiar with the environment of the tool it will be easier for them to use the tool. It has been shown that using these tools improves better collaboration among members of a team on a group project.

REFERENCES

- [1] Alfonso, M. I., Mora, F., Learning Software Engineering with Group Work, *16th Conference on Software Engineering Education and Training (CSEE&T 2003)*, March 20-22, 2003.
- [2] Jarzabek, S., Eng, P., Teaching an Advanced Design, Team-Oriented Software Project Course". *18th Conference on Software Engineering Education and Training (CSEET '05)*, April 18-20, 2005.
- [3] Lomas, C., Burke, M., Page C. L., Collaboration Tools
<http://elf.westernsydneyinstitute.wikispaces.net/file/view/Educause+Learning+Initiative+08.pdf>, posted August, 2008.
- [4] Martin, J. A., Should You Move Your Small Business to the Cloud?,
http://www.pcworld.com/businesscenter/article/188173/should_you_move_your_small_business_to_the_cloud.html, posted January 29, 2010.
- [5] Robillard, P. N., Teaching software Engineering through a Project-Oriented Course, *9th Conference on Software Engineering Education (CSEE)*, April 21-24, 1996.
- [6] Somerville, I. *Software Engineering 8th Edition*, Addison Wesley, University of St. Andrews, United Kingdom, 2006.
- [7] van der Duim, L., Andersson, J., Sinnema, M., Good practices for Educational Software Engineering Projects, *Proceedings of the 29th International Conference on Software Engineering*, May 20-26, 2007.
- [8] Guide to use Netbeans IDE
<http://netbeans.org/kb/docs/ide/kenai-collaboration.html>, retrieved December 3, 2010.
- [9] Problem-based learning , http://en.wikipedia.org/wiki/Problem-based_learning, posted December 4, 2010.

- [10] Software Engineering Code of Ethics and Professional Practice,
<http://www.acm.org/serving/se/code.htm>, retrieved December 3, 2010.

FORMAL THEORY FOR SOFTWARE ENGINEERING

STUDENTS*

*Ken Riggs
Computer Information Systems
Florida A&M University
Tallahassee, FL
850-412-7351
krriggs@cis.famu.edu*

ABSTRACT

This paper describes the use of an automated proof assistant in an introductory, graduate level, Formal Methods of Software Engineering course. Proof is difficult and often seen as abstract but tools can be the basis for relating proofs to practice. The proof editor JAPE can animate formal proofs in various theories, providing students with a significantly self-driven exploration of theory. The existence of machine readable theory objects also presents the opportunity to automate relationships between theory and topics more familiar to the student - programming in this case. We have documented increased work and improved attitude among students toward formal methods and proof using this combined approach. Although the particular example is a graduate course, we believe the increased student involvement in a structured experience leads to better outcomes and can be well-employed in many courses, even as a stand off module.

INTRODUCTION

This paper concerns efforts to introduce proof in a meaningful and fruitful way to Software Engineering students. The paper discusses the use of a pedagogical tool for logics, its use and results and ways to increase its relevance to Software Engineering students. The specific experiences and results relate to a graduate course in Formal

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Methods of Software Specification. However we believe the lessons and tools can be extended in the computer science engineering and technology curricula.

This paper is devoted to two principles that we believe important: Environments for independent student exercise should be as rich and supportive as possible given the learning goals. Work in these environments should be strongly related to experience familiar to the learner. Both of these principles would seem obvious. The paper examines a tool that embodies the first principle and an exploit of the tool that we think demonstrates the promise of the second.

BACKGROUND

This subsection considers a course, a tool and a theory. The course is a course in formal specification for software. The tool is an automated proof environment pitched at the human conceptual level. The theory is the well-known, if typically difficult to learn, Hoare theory for program correctness.

Course

The course in Formal Methods of Software was somewhat traditional. It contained substantial work on program correctness and on formal specification. The first topic was the Hoare Logic for program correctness. This was originally pen and paper work. This paper is concerned with adding and exploiting automated pedagogy for this topic. We describe the Hoare Logic in more detail at the end of this subsection.

The second major topic was program specification. The course used the Z specification language for this. The Z language was taught using the Z/Eves tool which provides an IDE and proof automation for proofs of properties using. This tool has also been found particularly useful in classrooms. The problem was that the automated theorem prover (EVES) of Z/EVES produced proofs that were beyond opaque to students so that proof failures could not readily be remediated - or even explained. Our concern in this paper is developing a better appreciation of proof in students. Detailed consideration of Z and Z/EVES is beyond our scope here.

Hoare Logic

Hoare Logic is an axiomatic system for proving program correctness. It has few axioms but some are complex. Proofs quickly become long because of the proliferation of details. Automation is often a best practice for this kind of complexity. Here we can only indicate the form and complexity of Hoare Logic.

The central feature of Hoare Logic is the Hoare triple. A triple describes how the execution of a piece of code changes the state of the computation. A Hoare triple is of the form:

$$\{P\} \text{ code } \{Q\}$$

Where P is a logical formula for prior conditions, code is some element of structured code and Q is logical formula for post condition.

Some Hoare axioms are as simple as saying that the result of an assignment of a variable to an expression has property P just if property P was true of the value being assigned to the variable. A trivial example might be:

$$\{z=43\} \ x := z \ \{x = 43\}.$$

Here we employ the assignment axiom to establish that if $x=43$ was true of x before $x := x$ then $x=43$ after. Of course real proofs build up complex properties.

Other axioms are more complex. The most complex is the axiom for the while loop:

$$\{P\} \text{ while (condition) do body } \{ Q\}$$

Simply marshalling all the parts, called proof obligations, is difficult. The rule for proving that that 'Q follows from P after the loop' involves all the following:

1. Finding an invariant condition that is true during each iteration the loop and body after
2. Showing that the loop will terminate
3. Showing that the loop termination condition plus the invariant imply the desired predicate for after the loop.

A fuller introduction to the theory can be found easily on the web, e.g. beginning at the Wikipedia entry for Hoare Logic. [7] For now we simply note that a proof consists of chains and nests of statements derived from the theory's seven axiom schemata. So in the end we have a complicated organization of difficult and abstract pieces. The class results seem to indicate that a well-structured system such as JAPE can make it possible for 'average' students to successfully employ and evaluate so much complexity.

1: $\{i \geq 0 \wedge i \neq 0\} (i := i - 1) \{i \geq 0\}$	
2: $i \geq 0 \wedge i \neq 0 \rightarrow _M > 0$	
3: integer Km	assumption
4: $\{i \geq 0 \wedge i \neq 0 \wedge _M = Km\} (i := i - 1) \{_M < Km\}$	
5: $(i \geq 0) \text{while } i \neq 0 \text{ do } i := i - 1 \text{ od} \{i \geq 0 \wedge \neg(i \neq 0)\}$	while 1,2,3-4
6: $i \geq 0 \wedge \neg(i \neq 0) \rightarrow i = 0$	
7: $(i \geq 0) \text{while } i \neq 0 \text{ do } i := i - 1 \text{ od} \{i = 0\}$	consequence(R) 5,6

Figure Figure 1 - a Hoare Proof in progress with JAPE

JAPE Proof Editor

JAPE is a proof editor for multiple logics developed by Richard Bornat and Bernard Sufrin. [2] It is freely available and has been in use for more than a decade. A facile user interface and the ability to represent multiple theories were prime motivations in JAPE's development.

We give a brief sketch of using JAPE in a Hoare Logic proof. Figure 1 shows a JAPE proof in progress. Proofs are developed by both sequential and nested steps much like programs. The box in the figure represents nested obligations. The ellipses indicate missing sequences of statements needed to establish the following statement from the preceding.

JAPE acts somewhat like a syntax directed editor that knows what is required for a proof operation. Even long proofs may be constructed by apply axioms, filling in particulars and inventing necessary details. JAPE marshals the necessary bookkeeping but not the decisions about what part of the theory to use or what the particulars must be.

Professor Bornat, most notably with Aczel, has worked to verify and improve the pedagogical value of JAPE. [1] A number of papers on JAPE and its tests are collectively available from a single page and the interested reader may wish to continue there. [??] The result is a system that requires the student to enact principles of the theory through an interface. Missing details handled are displayed. The interface is visual and interactive. Figure 1 is a snapshot of a JAPE proof in Hoare Logic underway (minus distracting IDE details)...

GOALS

The goals of the revised course were to provide more effective and relevant exposure to formal proof. This course concerned only a few formal methods: formal first order predicate calculus, Hoare logic and Z specification. However, formal proof is fundamental to advanced methods for software assurance of all kinds. Yet proof is often poorly understood among mathematics majors as well as CS and STEM students in general. [4]

Our Formal Methods course evinced these problems in specific ways: 1) the complex axioms of Hoare Logic mean it is easy for students to unwittingly err in proofs – or worse, to miss their point. 2) The automatic verification of an industrial strength formal tool such as Z/EVES [6] used in the final part of the course was largely opaque to the student.

Surveying available proof tools, JAPE seemed uniquely appealing with its "quiet interface" allowing students to concentrate on concepts of the theory. [2][3] It was hoped that experience with (semi-) automated proof at the human conceptual level would enhance proof ability and provide at least existential corroboration that fully automated proofs were reliable. The JAPE IDE was expected to provide a rich exploratory environment to motivate the students.

METHODOLOGY

The course began with JAPE, training first in Natural Deduction, and then Hoare Logic. The amount of student independent work was significantly increased as we shall explain below. The increase in class time used up front was compensated to some degree by increased outside student work and some apparent gains in the assimilation of Z syntax.

Time was set aside for an introduction to logic using JAPE's built in theory for first order predicate calculus. This gave time to iron out the (few) difficulties the students had in using the interface. The set theory module was not attempted for reasons of time and some theoretical concerns with the theory as implemented there.

An immediate phenomenon of these additions was an increase in "good" questions - those related well to the problems at hand and their theoretical solution. To some degree this continued in the second part of the course but somewhat lessened. We report on a first attempt to validate the positive effects in the Results section.

Methodological Extension

Sometimes improvement goes both ways. Two of fourteen assigned HL conjectures were in fact false. This falsity cannot be directly proven in the logic. Student problems with attempts to prove false theorems seemed to indicate a need (or at least opportunity) to connect the experience of Hoare Logic more tightly with the student's experiences of debugging programs.

Proofs of false conjectures cannot be legitimately completed. Since these were proofs about programs we surmised that they might see the failure more readily if they were to debug the code itself. We built a small tool to translate HL proofs to code. Students could try to disprove the Hoare Logic conjecture by searching for bugs in the corresponding program. It also made the connection between code and Hoare logic, including predicates, more immediate.

The instructor built a tool (HL2Java) to transform JAPE Hoare Logic proofs into Java. This ad hoc tool could not solve all problems of Hoare proof translation but was adequate for the examples at hand. Presented with the program most students could readily find a set of inputs that caused the program to err. Some could also trace that back to their proof difficulties. In any case we believe that such a direct link, made possible by the existence of the JAPE IDE, was also instrumental in increasing student knowledge and improved student attitude toward the formal theories.

RESULTS

We were able to cover more problems and with apparently improved understanding using JAPE as an introduction to Z specification. Most students completed, or nearly completed, the 90+ Natural Deduction proofs and disproofs. Better students completed the 12 true Hoare Logic conjectures, some of which could easily reach 90+ lines. Still, the most interesting effect came in class: students' questions were more pointed (due to the examples at hand?) and more trenchant. (Due to improved understanding?)

Complete understanding of the automated proofs of Z/Eves remained beyond the students. This is because machine logic is often quite different than human logic. However students did seem more comfortable with them. Experience with complex logical formulas also seemed to ease students' apprehension of the rich mathematical vocabulary of Z. We also now use the JAPE HL examples, reformulated in Z, as our initial Z examples. These effects seem to be corroborated by a post class survey.

The in-class effect was striking enough that we surveyed students on their background with, perceived progress in and attitude toward formal methods. The main points are these: No students had any prior training in formal proof. Few had mathematics beyond Calculus I-II. They found the JAPE experience difficult but rewarding. The effect on their attitudes and self-evaluations were equally important. They accurately estimated their achievement. Their attitude to formal methods was generally positive.

CONCLUSIONS

We have employed JAPE in a Software Engineering curriculum where assurance is a goal, but formal logic and proof experience are weak. Jape is a proof editor for logical theories that has benefited from significant pedagogical design. Our goal was to find a tool to encourage students apply the necessary effort to understand formal proof. There is no royal road to proof, but well-built, perspicuous tools can induce much more effort.

Based on JAPE's proof representation we also created a simple prototype tool (HL2Java) to help students address propositions that were not provable. Translating the failed proof into a program related the proof problem to student experience and skill. It also seemed to increase appreciation of the relation between theory and practice. Such opportunity can arise only when the theories and activities are instantiated in some system as they are in JAPE. Once such a system is in place, other opportunities to connect to students may be constructed.

A small survey was given to investigate student knowledge and attitudes after these changes. Positive results seem evidenced by the amount of work students were encouraged to perform, the improved quality of in-class interactions, greater self-awareness and more positive attitudes. Student estimates of progress correlated almost exactly with the instructor's.

While this example was based on a rather esoteric class, logic is fundamental to any analytical endeavor and we believe the results are broadly relevant throughout STEM courses. JAPE was designed to allow development of new axiomatic theories as needed and can be extended. Well-designed automated tools provide the student a theory experience which is more compelling and immediate. Such tools also provide the structure for linking problems to more common student experience. In an age of distractions and online learning it seems we would do well to follow such methods.

REFERENCES

- [1] Aczel, J. C., Fung, P., Bornat, R., Oliver, M., O'Shea, T., & Sufrin, B. (2003) "Software that assists learning within a complex abstract domain: the use of

constraint and consequentiality as learning mechanisms", *British Journal of Educational Technology*, Vol. 34, No. 5, pp. 625-638, ISSN: 0007-1013

- [2] Bornat, R., Sufrin, J., Animating Formal Proof at the Surface: The Jape Proof Calculator, *the Computer Journal*, 42(3), 177-192, 1999.
- [3] Bornat, R., A Minimal Graphical User Interface for the Jape Proof Calculator, *Formal Aspects of Computing*, V 11, N 3, 1999, pp 244-271.
- [4] Pep, S., The Role of Logic in Teaching Proof, *The American Mathematical Monthly*, 110(10), 886-899, 2003
- [5] Riggs, K. R., Bringing Logic to Programmers, *Proceedings of the International Conference on Education and Information Systems: Technologies and Applications*, Orlando, FL, July11-15, 2007.
- [6] Saaltink, M., The Z/EVES System, ZUM '97: The Z Formal Specification Notation, *Lecture Notes in Computer Science*, V 1212/1997, Springer Verlag, pp. 72-85.
- [7] Wikipedia, Hoare Logic, http://en.wikipedia.org/wiki/Hoare_logic, visited 1/21/11.
- [8] Aczel, James, The Jape Visualisation Project, <http://www.aczel.co.uk/Jape/Papers/index.html>, visited 1/21/11

ANDROID APPLICATION PROGRAMMING*

CONFERENCE TUTORIAL

*Frank McCown
Computer Science Department
Harding University
Searcy, AR 71239
501 279-4826
fmccown@harding.edu*

As smartphones and mobile devices become ubiquitous, students are showing increasing interest in programming these devices, and many CS departments are adding mobile computing electives to their curriculum. Google's Android OS is a freely available and popular smartphone platform where applications are implemented in Java using the Android SDK.

In this tutorial, participants will be introduced to mobile application development and the Android SDK. We will write some simple Android apps with Eclipse and run them on an emulator. We will discuss some teaching strategies for those interested in teaching or using Android in an upper-level CS course.

Participants may simply observe, but to get the most out of the tutorial, participants should be capable of writing Java programs in Eclipse and should bring their own laptop preloaded with Eclipse and the Android SDK. Instructions on downloading and installing the Android SDK can be found here: <http://developer.android.com/sdk/>.

PRESENTER BACKGROUND

Dr. Frank McCown is an assistant professor of Computer Science at Harding University. Dr. McCown co-taught an upper-level Android and iPhone programming course at Harding University in spring 2010, one of the first Android and iPhone courses to be offered in the US, and has produced a number of teaching materials on Android programming. Dr. McCown developed the Memento Browser (<http://code.google.com/p/memento-browser/>), an Android app that allows users to transparently access archived versions of web pages.

* Copyright is held by the author/owner.

HTML 5 PROGRAMMING*

CONFERENCE TUTORIAL

*Dan Brandon
MIS, School of Business
Christian Brothers University
Memphis, TN 38104
901 321-3615
dbrandon@cbu.edu*

HTML5 will soon become the new international HTML standard replacing XHTML and earlier versions of HTML. It will be the main web programming tool not only for traditional devices such as PC's and laptops but also for emerging mobile devices such as iPhone and Android smartphones. It may also displace proprietary software such as Flash and Silverlight for rich web media presentations. Key web industry players are quickly adopting HTML5 and incorporating it into their browsers and applications. As a result, students are showing keen interest in this new HTML platform and CS/IT/IS instructors should soon be redesigning their web courses accordingly.

The HTML5 syntax is no longer based on SGML, and it is not even an XML language since it is served up as a MIME type of text/html. It comes with a new introductory line that still looks like an SGML document type declaration (<!DOCTYPE html>), which enables standards-compliant rendering in modern browsers. HTML5 has been designed to be backward compatible with the parsing of older versions of HTML, specifies currently undocumented but interoperable technologies such as AJAX, specifically defines error handling, and introduces many new features including: scalable vector graphics, integrated multimedia, geolocation, local storage, dragging and dropping, etc.

Participants may simply observe, or they may participate by bringing their own laptops or net appliances equipped with a text editor and modern browsers.

Presenter Background

Dr. Dan Brandon is a Professor of Management Information Systems at Christian Brothers University in Memphis where he teaches courses in MIS, programming, database, decision support, and project management. He has authored two books and numerous journal articles and conference proceedings including some for CCSC. He has designed and developed web based systems for a number of business organizations and was formerly the Director of Information Systems at the NASA Stennis Space Center.

* Copyright is held by the author/owner.

CS1 STUDENTS' UNDERSTANDING OF COMPUTATIONAL THINKING CONCEPTS*

Jake A. Qualls

University of Memphis

Department of Computer Science

Memphis, TN 38152

(901) 678-5465

jaqualls@memphis.edu

Michael M. Grant

University of Memphis

Instructional Design & Technology

Memphis, TN 38152

(901) 678-4918

mgrant2@memphis.edu

Linda B. Sherrell

University of Memphis

Department of Computer Science

Memphis, TN 38152

(901) 678-5465

Linda.Sherrell@memphis.edu

ABSTRACT

This paper presents a small, multi-method case study conducted throughout the spring 2010 semester of CS1: Introduction to Computer Science at the University of Memphis. This study explored students' comprehension of three Computational Thinking concepts: algorithms, abstraction and efficiency. Results indicate that students understand and value the concept of the algorithm but are only beginning to understand abstraction. In addition, student observations about efficiency represent the true focus of Computational Thinking, i.e., applying computer science concepts to other fields of study. Implications for teaching CS1 and for future research are also considered.

1. INTRODUCTION

Computational Thinking (CT) is an approach to problem solving that consolidates logic skills with core computer science (CS) concepts. Jeannette Wing identified in [13]

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

that "computational thinking includes a range of mental tools that reflect the breadth of the field of computer science." Computational Thinking encompasses many computer science concepts, including topics not regularly encountered in introductory courses. The most important and basic concepts need to be identified in order to transition into the world of CT.

For this study, the concepts of abstraction, efficiency and algorithms were identified as the "mental tools" vital to Computational Thinking. These skills, when thoroughly understood by students, provide a foundation for problem solving skills. Introductory courses often introduce the concept of the algorithm, discuss problems that rely on abstraction and, usually, end with a brief discussion of efficiency by utilizing basic searching and sorting examples. It is our goal to understand how students comprehend and apply these concepts within the CS1 courses offered at the University of Memphis. With the increased focus on CT, the researchers explored an initial, small-scale study focusing on CT concepts embedded within the current CS1 curriculum at the University of Memphis.

All of the aforementioned topics are important concepts for introductory computer science students. It is also imperative that the students have a solid understanding of these concepts in order to successfully progress through the program. Therefore, we aim to answer the following questions:

1. Do students thoroughly understand the concept of an algorithm?
2. Can students effectively apply abstraction to solve a problem?
3. What do students know about efficiency?

Sections 2, 3, and 4, cover difficulties in teaching Algorithms, Abstraction and Efficiency, respectively. Section 5 details the course format. Section 6 represents our approach to the study with results presented in Section 7 and a discussion in Section 8. Finally, Section 9 contains the conclusion.

2. ALGORITHMS

It is understood that the concept of the algorithm is one of the most basic and fundamental concepts of computer science and one that is often introduced at the beginning of introductory courses [9]. The concept is introduced to beginning students as a logical and systematic method for solving a problem. Solomon, in [12], argues that introductory computer science courses are losing focus by requiring language specific skills at the expense of basic algorithmic understanding. Mastery of algorithmic understanding is applicable to many fields of study. By focusing on this basic skill, students develop the ability to transition between intuitive understanding of the problem and formal description. In addition, students improve their ability to represent problem descriptions and solutions.

3. ABSTRACTION

Abstraction can be defined as the process of eliminating the minor details of a problem in order to focus on the major details. A fundamental computer science skill, abstraction is also an imperative cornerstone concept for computational thinking. Wing

noted that CT is "Conceptualizing, not programming" and often requires "thinking at multiple levels of abstraction." [11]. However, abstraction is often a difficult concept to convey in introductory CS courses. Students who are unable to think abstractly often have difficulty with computer science and other fields that rely on abstraction. Without this concept, students become overwhelmed with detail and often become frustrated with the programming process. Hazzan illustrates in [6] how students often reduce the level of abstraction so that new concepts can become "mentally accessible." By using examples from computer science and mathematics he argues, "students unconsciously employ cognitive mechanisms that enable them to make sense of abstract concepts." Categorizing abstraction as a "soft idea" [7], Hazzan notes that simply lecturing about the concept is not the solution in improving the ability of the students. In terms of abstraction, students must first see, feel and recognize the use of abstraction and experience a fair amount of uncertainty in the learning process to use the concept effectively.

From a developmental approach, Piaget [8] hypothesized that individuals progress through four stages of cognitive development and should be able to demonstrate more complex cognitive abilities with each new stage. The sensorimeter, pre-operational and concrete stages should be attained by the time an individual reaches early adolescence. It is within the fourth stage, the formal operational stage, that humans are thought to develop the ability to think abstractly. According to Piaget, the fourth stage of development occurs between adolescence and adulthood. As many students enrolled in introductory computer science courses are recent high-school graduates, it is probable that a few individuals have not progressed to the fourth stage of development and, therefore, are unable to sufficiently demonstrate the required thinking skills needed to apply abstraction. In fact, "only 35% of high school graduates in industrialized countries obtain formal operations."

If our students lack the appropriate cognitive skills to apply abstraction, then it may be appropriate to "train" our students for such tasks. Kramer makes this proposal noting that of the 60 courses offered at Imperial College, no one class exists where abstraction is the focus [10]. This statement is also valid for the University of Memphis. Many, if not all of the courses offered in the computer science curricula rely heavily on the use of abstraction but most often attempt to indirectly teach the concept.

4. EFFICIENCY

Efficiency is concerned with the minimization of resource consumption. A basic introduction to this concept is often discussed late in the CS1 curricula and usually in the context of basic searching and sorting techniques. Using illustrative non-"toy" examples is an excellent way to introduce efficiency and one that allows students to see how a problem solution can be improved upon [3]. It is imperative that students understand the importance of efficiency and why it is desirable to design and create efficient software.

Observations within a CS1 course at the Open University of Israel reported a dropout rate of 50% and a <60% pass rate for those students remaining in the course [3]. In an effort to improve the situation and stress the importance of efficiency, the authors introduced efficiency early in the CS1 course. They felt that the examples in most textbooks were too simplistic and found that it was difficult in convincing the students

that more efficient solutions existed. Part of the change consisted of presenting the students with problems that had unreasonable solutions. This was used as a motivating factor and a situation where it was easy to convince the students that more efficient solutions were possible.

Similar to the aforementioned approach, [5] integrates efficiency early in the course. Alternative design and program analysis is used to introduce efficiency in terms of running-time evaluation. The author notes that through proper planning, students realize that, with better insight, program results can vary.

5. COURSE FORMAT

The introductory computer science course at the University of Memphis is a three-hour lecture with weekly two-hour closed labs. The course covers problem-solving strategies with an emphasis in fundamental programming skills. Concepts include primitive data types, assignment statements, conditionals, loops and arrays. The required textbook is [1]. Typically, in addition to the ten lab assignments and two lab quizzes, the course is composed of four larger programming assignments with two exams and four quizzes. The course ends with a brief introduction to object-oriented programming.

6. METHOD

This small multi-method case study was designed to explore students' comprehension of CT concepts throughout the 2010 spring semester of CS1: Introduction to Computer Science.

6.1 Data Collection

An assessment (see Appendix A) was created to align to the three identified tenets of CT. Understanding the concept of the algorithm (Appendix A, Item 4) provides one with the skills needed to create a logical sequence of steps toward solving a problem. Item 4 of the assessment provides the students with an outline needed to implement the bubble sort algorithm. By requiring students to trace through the provided lists of numbers, one obtains an understanding on the students' ability to follow logical and systematic steps needed to solve the problem. In addition, students are faced with the decision to make proper algorithmic modifications if the end result is not an acceptable solution to the problem.

Abstraction (Appendix A, Items 2 and 3) is a concept imperative to both CS and CT because it allows one to structure a potential solution by eliminating details of the problem. Item 2 of the assessment presents students with a problem where they need to eliminate the complexities of binary representation in order to represent alphabetic characters. Item 3 requires the student to think abstractly about variable assignment in order to evaluate a sequence of instructions. Both items aim to capture whether or not students are able to apply the concept of abstraction.

Understanding efficiency enables one to be conscious of resource consumption. In addition, it is a tool that better equips the student to identify situations where efficient

solutions may or may not be possible. The assessment question focusing on efficiency (Appendix A, Item 1) provides a scenario where the student must guide a robot through a maze using three different operations. Upon successful navigation through the maze, students must decide if their solution is the only solution to the problem.

Face validity of the instrument was determined by two of the researchers and the course instructor for an appropriate measure of CT. The assessment was administered to 38 students during the first week of the study and prior to scheduled lectures. The same assessment was administered to 23 students during the last week of study. Five of the 23 post-test participants were removed from the study because they had not completed the pre-test exam. Reasons for this include late enrollment or non-attendance on exam day. The course instructor and the primary researcher independently assessed results for the items. A very high level of agreement was reached with scores between the two assessors.

In order to gather qualitative data from the students and triangulate the quantitative data [2], focus group interviews were conducted to gather thoughts and perceptions from students on concepts of CT and the CS1 course. An interview protocol was aligned to the research questions and CT components. Two focus groups occurred during the semester with a volunteer participation level of 6 students. One was held during midterms week; and one during the week of final exams. The first focus group was concerned with gathering intermediary data to ascertain student development up to midterms week. The second focus group was administered to corroborate pretest and posttest data and for comparison to midterm focus group responses. Interviews were audio recorded then transcribed. Qualitative analysis of the transcripts followed a constant comparative method [11] from codes to patterns. Quality and rigor of the qualitative data were maintained with an audit trail, peer debriefs and triangulation [11].

7. RESULTS

7.1 Quantitative Results

The pretest and posttest was composed of six questions. Four of the questions focused on the areas of algorithms, abstraction and efficiency. Table 1 provides the descriptive statistics for all items on the assessment.

Table 1. Descriptive Statistics for Quantitative Data

Concept	Efficiency	Abstraction		Algorithm
Item	1	2	3	4
Pretest Mean (n=18)	4.94	2.33	3.33	3.33
Pretest Std. Dev.	1.55	1.28	0.97	0.97
Posttest Mean (n=18)	5.05	2.92	3.83	4.28
Posttest Std. Dev.	1.43	0.35	0.51	1.17

Dependent t-tests were conducted for each of the four items, comparing means for pretest to posttest. For item 1 related to efficiency, a $t(17)= .199$, $p=.845$ illustrates that the course-related instruction was not statistically significant. Item 2 related to abstraction had a $t(17)=1.80$, $p=0.90$ and was not significant. Item 3 also related to abstraction had

a $t(17)=1.932$, $p=0.70$ and was also not significant. Item 4 related to algorithm had a $t(17)=2.72$, $p=0.015$. This result suggests the course instruction had statistically significant impact on student performance.

7.2 Qualitative Results

7.2.1 Understanding the Value of Algorithm

Participating students were confident about their understanding of the algorithmic concept and the value of the concept to the field of computer science. When asked about algorithms, all students provided a correct definition with the most accurate being "*the logical steps needed to solve a problem.*" Other variations include "*steps to solving a problem*" and "*a method to solving a problem and the steps you take.*" When asked to provide an example of a non-CS related algorithm, most, if not all, used the "Making a Sandwich" example of an algorithm. The most common CS related example mentioned was the linear search algorithm and students were able to give a detailed explanation for this specific, although basic, problem.

Students valued the concept of the algorithm and how it is a cornerstone for computer science. One student noted that the concept was important and stated, "*it's pretty much the basis of any program.*" This realization that algorithms are the basis for problem solving is an important realization. One student noted, "*You can't just jump into code without knowing the steps you have to take to solve the problem.*"

7.2.2 Defining Abstraction

When discussing abstraction, students demonstrated that they were in the initial stages of understanding the concept of abstraction. They were able to provide instances of where abstraction was used in CS1 but were unable to define the concept. Examples provided by the students included specific homework and quiz material, such as, "*Our last quiz with the GPA question. We had to calculate the GPA based on the numerical grade and how many credits they counted and then determine what it was worth.*" One student realized that problems might need to be abstracted out in multiple levels in order to solve the problem by noting, "*[the problem] would be broken down on more levels.*" A majority of the students acknowledged that all problems require some level of abstraction with one student remarking, "*you kind of do that with any problem you're trying to solve.*"

Uncertainty and hesitation were present when the students were trying to define the concept of abstraction. In fact, no students were able to correctly define the concept. The overwhelming response was they were not familiar with the terminology. One student noted, "*I do not recall that term being mentioned*" while another admitted, "*Honestly, I don't know what abstraction is.*" Reflecting on their experiences in the CS1 course, one student stated, "*It may have been described but never really termed that.*"

7.2.3 A CT Understanding of Efficiency

The students discussed efficiency in two specific categories: efficiency outside of computer science and efficiency in terms of productivity and timesaving measures. When asked about efficiency, many students mentioned situations outside of computer science. One student quickly stated, "*Most of the time [the programs] are related to medical solutions.*" Others noted, "*All of the programming that goes into automobile computers*" and "*the field of aviation.*" All are observations that represent the focus of computational thinking as applying computer science concepts to other fields of study.

In addition to the aforementioned observations, students also associated efficiency with productivity and time saving measures. They observed that efficiency is important and "*the more efficient something is, the more work you can get done.*" One student, when asked who is often concerned with efficiency, noted that, "*The end user. They want their things to work quickly. People are impatient. Obviously if one thing is faster than another then that person is going to choose the faster option.*"

8. DISCUSSION

The discussion of the findings integrates the quantitative and qualitative data. The discussion is organized by the research questions below.

8.1 Do Students Thoroughly Understand the Concept of an Algorithm?

Both the quantitative and qualitative findings suggest that students not only understand the concept of the algorithm but they recognize the value of the concept to the field of computer science. Assessment Item 4, which focuses on the concept, had a average gain of 0.95. This result suggests the course instruction had statistically significant impact on student performance. By combining the qualitative and quantitative data, it can be shown that students have a solid understanding of the concept and were capable of illustrating how algorithms are important to CS.

8.2 Can Students Effectively Apply Abstraction to Solve a Problem?

Assessment Items 2 and 3 represent questions focusing on abstraction. Mean difference of pretest to posttest gains for Item 2 was 0.59, while Item 3's gains were 0.50. There were only small gains. Neither of the items was statistically significant. Conducting focus group interviews revealed that students were unable able to accurately define abstraction, important information not captured via the assessment. However, students were capable of illustrating abstraction related examples encountered in the CS1 course. This demonstrates that they have adequate knowledge of abstraction but are not yet proficient with the concept.

8.3 What Do Students Know About Efficiency?

Assessment Item 1, related to efficiency, had the smallest mean gain of 0.11, and it was not statistically significant. The pretest score was the highest, which may suggest that

students had an understanding of efficiency upon entering CS1. The focus of the CS1 instruction may have had the least impact on this CT concept, though. Considering the qualitative data, students appear to have a general understanding of efficiency. The more important observation is that students were able to transition the idea of efficiency from a CS concept to one that is applicable to situations in other fields.

9. CONCLUSION

This paper offers an important first step in accurately identifying and defining the core CT concepts, as well as considering the alignment between CT concepts and CS curricula at the University of Memphis. The results presented in this multi-method case study are an initial attempt at understanding student comprehension of CT concepts. It should be noted that generalizing this study is limited. Additional limiting factors include the small sample size and the difficulty in keeping course instruction aligned with the targeted CT concepts. Future research should consider larger sample sizes with randomized or quasi-experimental designs, and more in-depth qualitative data collection may include multiple interviews with individual participants, as well as analyzing artifacts of student work. Longitudinal studies may examine CT concepts and student performance across CS courses. Additional data for the study are currently being collected from CS1 courses offered during the fall 2010 semester. Although not included in the current results, the newly collected data will shed further light on the overall learning experience for the CS1 students at the University of Memphis.

10. REFERENCES

- [1] Bravaco, R., Simonson, S., *Java Programming: From the Ground Up*. McGraw-Hill, 2009.
- [2] Cresswell, J.W., *Qualitative inquiry and research design*. Thousand Oaks, CA: Sage, 1998.
- [3] Gal-Ezer, J., Vilner, T. and Zur, E., Teaching Algorithm Efficiency at CS1 Level: A Different Approach, *Computer Science Education*, 14, (3), 235-248, 2004.
- [4] Gal-Ezer, J. and Zur, E., The Efficiency of Algorithms -Misconceptions, *Computers and Education*, 42, (3), 215-226, 2004.
- [5] Ginat, D., Efficiency of Algorithms for Programming Beginners, *27th SIGCSE Technical Symposium on Computer Science Education*, 28, (1), 256-260, 1996.
- [6] Hazzan, O., How Students Attempt to Reduce Abstraction in the Learning of Mathematics and in the Learning of Computer Science, *Computer Science Education*, 13, (2), 95-122, 2003.
- [7] Hazzan, O., Reflections on teaching Abstraction and other Soft Ideas, *Inroads*, 40, (2), 40-43, 2008.

- [8] Huitt, W. and Hummel, J. Piaget's theory of cognitive development. *Educational Psychology Interactive*, www.edpsycinteractive.org/topics/cogsys/piaget.htm, retrieved August 16, 2010.
- [9] IEEE ACM Computing Curricula 2001, www.acm.org/education/curric_vols/cc2001.pdf, retrieved September 5, 2010.
- [10] Kramer, J., Is Abstraction the key to computing?, *Communications of the ACM*, 50, (4), 37-41, 2007.
- [11] Glaser, B.G. and Strauss, A.L., *The discovery of grounded theory: Strategies for qualitative research*, New York: Aldine Publishing Company, 1967.
- [12] Solomon, J., Putting the Science in Computer Science: Treating Introductory Computer Science as the Study of Algorithms, *Inroads*, 39, (2), 46-50, 2007.
- [13] Wing, J., Computational Thinking., *Communications of the ACM*, 49, (3), 33-35, 2006.

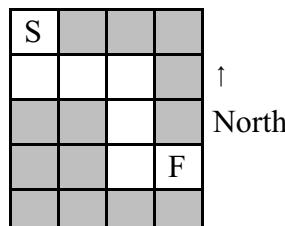
APPENDIX A

1.) (Efficiency): You are trying to guide a controllable robot through a maze, which consists of a grid of tiles. The robot can move over only the white tiles, not the shaded ones. Furthermore, the robot responds to only three commands:

forward moves it ahead by one tile, in the direction it's currently facing.

turn right turns it 90 degrees to its right. For example, if the robot were initially facing north, issuing a turn right command would make it face east.

turn left turns it 90 degrees to its left.



Assume that the robot is initially in the tile marked S and facing east. Write a sequence of commands that will guide the robot to the tile marked F. Is your sequence the only possible sequence that will guide the robot through the maze?

2.) (Abstraction): Internally, a computer represents all information (whether it's text, pictures, or sound) using ones and zeros. How do you think sequences of ones and zeroes might be used to represent the letters of the alphabet A-Z?

3.) (Abstraction): Given the following operations:

Operation	Definition
$x \leftarrow y$	Store the value y in the variable x.

$x \sim\leftarrow y$	Store the value y in the variable x , but only if $y > x$. If $y \leq x$, this operation does nothing.
$x \leftarrow\sim y$	Store the value y in the variable x , but only if $y < x$. If $y \geq x$, this operation does nothing.

What are the values stored in the variables a and b after the following list of operations has completed? Assume that the operations are performed sequentially (i.e., one after the other).

```

a ← 0
b ← 2
a ~← b
b ←~ a

```

4.) (Algorithm): Consider these instructions, which are meant for sorting a list of numbers (i.e., arranging the numbers from least to greatest):

- a.** Compare the first and second numbers in the list.
- b.** If the first number is greater than the second number, swap the positions of the first and second numbers. If the first number is not greater than the second number, do nothing.
- c.** Repeat steps (a)-(b) for the second and third numbers, third and fourth numbers, and so on, until you reach the end of the list.

Trace how these instructions would be carried out on a list containing the numbers 31, 4, 15, 9, 27, and 18 (in that order). Is the list completely sorted after carrying out the instructions? If not, how would you modify the instructions to ensure that they will sort any list.

TEACHING COMPUTER SCIENCE IN A BIG QUESTIONS FRAMEWORK*

*Steve Donaldson
Department of Mathematics and Computer Science
Samford University
Birmingham, AL 35229
(205) 726-2447
sfdonald@samford.edu*

ABSTRACT

It is something of a paradox that, despite the rapid change in technology, Computer Science pedagogy is in constant peril of falling into (and remaining trapped in) an abyss of staid strategies, monotonous methods, and tired techniques. Furthermore, common views outside the discipline place computer science and the prototypical computer scientist in an esoteric niche with little relevance beyond technological necessity. One potential means of escape from this quandary – a means to breathe new life into existing courses and to initiate the slow change of popular perception of the discipline – is to frame course content by big questions of meaning and value. The use of this approach in two quite different types of courses (discrete mathematics and artificial intelligence) is presented with the goal of illustrating its range of applicability, specific methodologies, and potential benefits.

INTRODUCTION

Having inherited some of the genetic composition of its engineering ancestry, it is probably not surprising that computer science inherited a collection of pocket-protector stereotypes as well, its practitioners frequently identified as "nerds" and the discipline itself often viewed as a sterile undertaking with little relation to the world of flesh and blood. The ubiquitous use of computers seems to have done little to slow this conception (and may even have exacerbated it). Furthermore, although there are plenty of counter examples, there must be enough computer scientists out there unconsciously supporting

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

the less-than-flattering impression of dullness, mechanism, and irrelevance to maintain current lay perceptions of the discipline.

Is it conceivable that this has anything to do with the way in which computer science students are educated? They do, after all, eventually end up being the ones about whom such judgments are made. Could the current difficulties in recruiting (and retaining) majors be associated with such perceptions? Is there any danger that computer science pedagogy could fall prey to its own special kind of loopiness (cf. [4]), trapped by repetitive and tiresome techniques that leave professors bored and students disengaged? Are the prevailing images accurate because computer science is really like that or is computer science like that precisely because that is what people believe?

One way to begin an escape from this chicken/egg dilemma is to acknowledge that computer science education should contribute to the production of more than just technically competent computer scientists. An hypothesis for how this might be accomplished is to frame computer science courses by big questions of meaning and value. As demonstrated below, this provides a scheme for (a) bringing new life to existing courses (increasing both student and professor interest), (b) helping students appreciate technical material for more than its immediate practical benefits, and (c) getting students to think in oblique ways about the material they are learning with the potential benefit of knowledge transfer across disciplines and into previously undreamed of domains. The ultimate goal is to cultivate more students who think (deeply) and fewer who are satisfied with learning a canned set of routines. Presumably, this is one of the primary reasons for teaching computer science instead of a soon likely-to-be-obsolete set of skills (e.g., such as those provided by myriad certification courses).

The following sections describe how computer science courses can be taught in a big questions framework. The practical objective is to present specific ideas about how to do so but the real target is to demonstrate how the discipline is relevant and meaningful beyond its obvious technological aspects.

METHODS

Before delving into details, it is important to identify just what is meant by the term "big question." Undoubtedly, any definition is necessarily subjective but there are several guidelines that can help distinguish between big questions of meaning and value and (hopefully important but more specific) scientific or mathematical research questions. In general, big questions of meaning and value (a) are of interest and importance for a large number of people (over long periods of time), (b) are of such a nature that answers have the potential for significant and lasting impact on human conceptions of meaning and value, and (c) subsume many smaller questions. These guidelines are not meant to suggest that some questions are not important (e.g., what to do with one's life, whom to marry, whether symbolic or sub-symbolic approaches are more fruitful for the construction of intelligent systems) but that their limited context does not represent the type of framework being presented here.

To illustrate the range of applicability of a big questions approach, two quite different types of computer science courses – discrete mathematics and artificial intelligence – will be considered. These courses differ in objectives (mathematical

foundations versus implementation mechanisms), audience (typically sophomores versus seniors), and frequently in numbers of students and types of assignments. Discrete math is a mainstay of almost all computer science programs whereas AI is sometimes excluded from the curriculum of smaller schools or made one of several options at larger ones. Interestingly, the overlap in content between these two courses (e.g., predicate logic for reasoning, graph theory for game playing) provides an apparent way to integrate the big questions theme on a larger scale.

A fruitful place to begin the process of integrating the material for either course into a big questions framework is to identify a collection of big questions that are appropriate for the course under consideration. A concept map [5] provides an ideal way to link course content to those questions as illustrated in figure 1 for a discrete mathematics course. Nodes shown in gray in the figure represent big questions and those shown in white represent standard course content. Arcs are labeled with a description indicating how content items are related to big questions. While some big questions will be primarily related to specific course content, others will be more sweeping in their scope and may apply to the course as a whole. A good example of the latter concerns how one looks at the world. Is it rough or smooth; discrete or continuous? Besides the obvious general relevance to discrete math, questions such as this particular one offer a chance to discuss novel hypotheses such as the discrete nature of space-time [5] – something most students (and professors) have probably never considered.

Production of such a map prior to the beginning of a course is an effective tool for helping instructors maintain sight of how and when to integrate big questions into class lectures and discussions and can be given to students as a constant reminder of the broader scope of class material. Because students entering a course such as discrete mathematics will have little to no concept of the relationships identified in such a map, it also reflects a portion of the learning goals for the course.

In order to ascertain the level of insight students possess at the beginning of a course, it is useful to ask them to produce a concept map prior to being given any information about course objectives or content. To do this without having to spend time defining and explaining concept maps, it suffices to ask the students to make a list of what they consider to be the big questions of life and, for each, to identify mathematical concepts that they consider relevant to that question. Asking them to give a brief statement indicating how the mathematical concepts are relevant to the corresponding big questions completes the assignment. A pre-printed form expedites this process. Responses can then be compared to the formal, instructor-prepared concept map and used to frame future discussion.

Many of the relationships between course content and big questions will seem natural, perhaps almost tautological, whereas others may be much harder to see from just a cursory consideration of the material. For example, truth tables, formal logic, and proof sequences lead naturally to questions such as, "What is truth (and how can one know it)?" However, it would not be a safe bet to assume that students make the conceptual leap from the limited applicability of binary logic as typically presented in textbook problems to wider epistemological issues. Yet, after they have been fed the standard logical fare – including how English statements can be mapped into predicate logic and how an entire logical edifice can be constructed which has the appearance of unassailable truth but

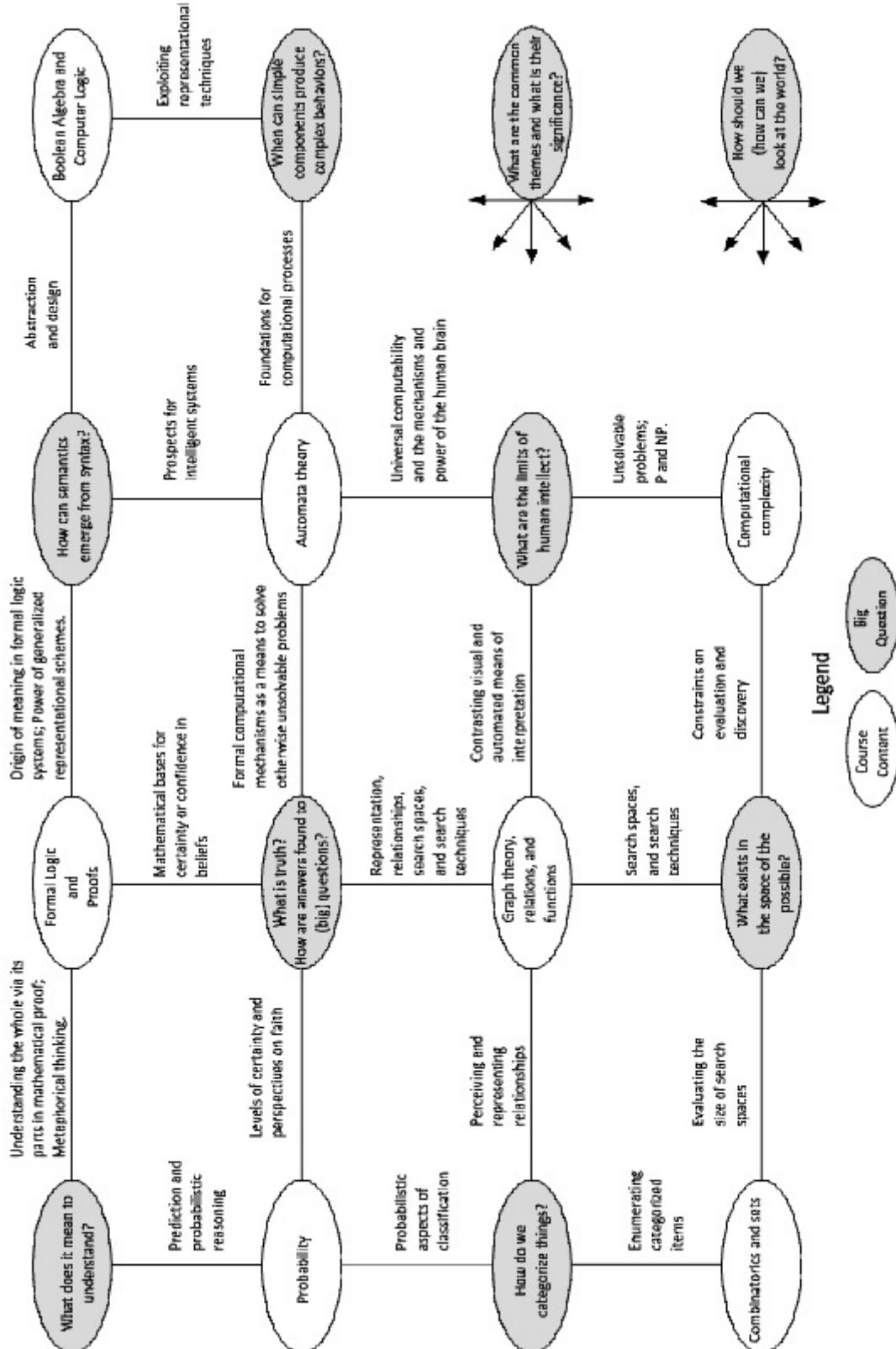


Figure 1
 Mathematical Perspectives on Big Questions of Meaning and Value
 (Course: Discrete Structures for Computer Science)

which will come crashing down if it is ever discovered that the assumptions upon which it is based are wrong – they are primed to see a much bigger picture. The fact that it took over fifteen hundred years for someone to rectify Aristotle's conclusions about the cosmos and the behavior of falling objects is a case in point, made possible as it was by Galileo's denial of the assumptions [1] [2]. Truth, it seems, is not always easy to come by, even for famous Greek philosophers. Psychologists are quick to point out more mundane but commonplace examples [3].

Some relationships between big questions and discrete mathematics concepts may not be so obvious, however, but may ultimately be more memorable. Consider, for example, the question, "What exists in the space of the possible?" in the context of a section on graph theory. Because graphs are ideal representational tools for showing the relationships between entities and permit location of a specific entity by some search process, it is quite easy to show students how a graph can be used to show the relationships between biological entities – e.g., based on their genetic code – where each node corresponds to a specific code and there are arcs between nodes (i.e., codes) that are similar (according to some criteria such as sequence similarity). Because some of the nodes would represent real species whereas others would not, somewhere in such a graph (if it included all possible codes), would be nodes representing the DNA for dinosaurs as well as creatures that have never existed but are biologically possible (e.g., a humanoid with four arms, entities with brains whose sophistication exceeds that of humans). This graph, in other words, could represent the search space of all possible biological entities. Or how about the search space of possible things to do next? One's future lies therein! Or the search space of all possible phrases? (Look out, Shakespeare.) And so forth... One can almost see the wheels turning as students contemplate the implications – and realize that they have a tool for thinking about them that their peers in other disciplines lack. Related computer science issues include how to actually search the graph, whether the search can be made tractable or is subject to combinatorial explosion, how one would recognize what a node actually means, or whether memory constraints would play a role in limiting the possibilities.

These types of relationships form the basis for discussion and can easily be turned into assignments, exam questions, or projects. It is also probably apparent that such questions are not limited to a course in discrete mathematics. Artificial intelligence provides a nice illustration. As with the discrete math approach, a good starting point to assess student thinking and establish a baseline from which progress can be measured is to ask students to prepare a list of ingredients necessary for constructing an intelligent humanoid system, to prepare a parallel list of big questions that might be raised when contemplating such a task, and to identify and describe relationships between the two lists.

A good supporting activity is to ask for students to provide true/false responses (on paper) to each of the following statements, shown to them in succession so that subsequent statements do not bias responses to earlier ones.

- a. Machine intelligence will never equal human intelligence.
- b. Humans are the most intelligent things that God could possibly create.
- c. Humans are really just a very sophisticated machine.
- d. Humans have never created anything that can outperform humans.

e. Some machines are already more intelligent than some humans.

This activity is geared to get students to think deeply about issues to which they have probably never given much prior thought. It is not unusual for a student to provide contradictory responses which (hopefully) helps them see that these are not trivial issues and that they are not yet thinking as deeply about the issues as they might have believed.

In general, the study of artificial intelligence raises a number of significant philosophical as well as scientific questions, many of which have had and continue to have enduring significance for humans. Perhaps one of the most significant is, "How does meaning arise from mindless mechanisms?" Students can be prompted to think about this via a series of leading questions:

- a. What is this question really asking?
- b. Why is it important?
- c. Why does it arise in the context of artificial intelligence?
- d. Where does it arise in the context of artificial intelligence?
- e. How do neural nets, formal logic, and programs illustrate issues associated with the question?
- f. What other big questions are related to this one and how are they related?
- g. How might the study of artificial intelligence help answer this question?

Following (or in conjunction with) presentation of technical material, class discussions aimed at helping students formulate answers to these questions would consider how brains are collections of electrical/chemical interactions between mindless components with no hint of overall purpose when viewed in isolation. This might be followed by reflecting on whether, if minds are the emergent property of a large number of specially wired mindless mechanisms, artificial devices might, one day, also have minds as well – and, when they do, if humans will still see themselves as special.

Clearly, this question is intimately connected with issues of syntax and semantics and suggests that, unlike the top-down imposition of meaning in a typical computer program, semantics in an intelligent system must be an integral part of the system (just as structure and semantics appear to be highly integrated in the human mind/brain). In fact, because no one has a clue how to build anything not involving mindless mechanisms, students contemplating this are faced squarely with the implications for biological entities, thereby leading them to see the significance of their studies to concepts of consciousness, free will, and the soul (cf. [6]).

RESULTS

Depending upon the sophistication and prior training of students, the results of the initial concept mapping exercises given to students (as described above) can vary widely and it is not unusual to discover that students in a discrete mathematics course do not have a ready response to either big questions or relevant mathematical concepts (and may substitute attempts at humor as a substitute for a serious effort). Similarly, AI students may initially show meager insight into the issues they are asked to consider. No matter – in each case they have been faced with the fact that there is a relationship between the two and that they, perhaps, should know about it. The door has been opened for subsequent discussions and demonstrations of those relationships.

Assessing the long-term validity of the hypothesis is problematic – one would like to be able to formally query students in five (etc.) years about this – but it is easy enough to ascertain via assignments and exam questions whether students are, at least, learning the proposed relationships. In a course such as AI, where a semester-ending project is frequently the norm, requiring students to consider the implications of their work with regard to how it might shed light on some particular big question offers an additional means of assessing the results of this approach.

Finally, although one is hesitant to make sweeping inductive generalizations, limited experience suggests that it may be much easier to implement this approach in some courses (e.g., AI) than in others (e.g., discrete math). Some just seem to be a natural "fit" but that may suggest that the greatest rewards are actually to be found in those where the relationships are less obvious. In any event, this approach does not require the creation of any new courses although it is likely that it may generate ideas for one or two. At least one thing is certain: the professor who commits to this method is forced to consider new approaches to presenting old material and is likely to begin to think about how everything can be viewed within a big questions framework. It is a bit like getting a new pair of glasses.

DISCUSSION

Framing course content by big questions presents a number of challenges. Instructors expect to cover a set amount of material in most courses and there is usually very little slack time. Frankly, if the plan is just to fill the gaps with periodic allusions to big questions, one should not expect much. Neither should this be attempted if it is felt that the result would be to "water-down" important material. Instead, a decision to attempt this can be viewed as an opportunity to reassess the relative merits of existing material and to weigh them against the possible advantages of changing the format. For instance, an instructor might decide that, important as it was to the historic context of AI efforts, some of the early AI work usually covered in textbooks really has little permanent relevance to the future of the discipline and will, in any event, not be retained by the student much beyond the exam. Replacing the time devoted to that material might be better spent discovering the implications of AI for humans both now and in the future and evaluating how such insights might actually contribute to design ideas.

Instructors adopting this approach can also expect to spend significant time in lesson, assignment, exam, and project preparation. Creation of a formal concept map, for instance, is a time consuming undertaking. There is no hiding the fact that bringing new life to a course is not without its birth pangs. The benefits, however, should be applicable to any course taught in a typical computer science curriculum (as suggested, for instance, by the use of this approach in courses as different as discrete math and AI). Almost certainly, those benefits will fluctuate dramatically based on such factors as the specific course, the time spent by the professor developing integration strategies, the time allotted for implementation of those strategies, class size, and student ability. Within this framework, however, there is considerable leeway for creativity on the part of the instructor and that is a significant part of the appeal. Ultimately, this hypothesis suggests that a big questions approach can help create students who are prepared to deal with new kinds of problems, are excited about learning, understand that their discipline is relevant

and meaningful in broad and significant contexts, and who become representatives of a new breed of computer scientists. It is an hypothesis awaiting further testing.

LIST OF REFERENCES

- [1] Galilei, G. *Dialogue Concerning the Two Chief World Systems*. New York: Modern Library, 2001.
- [2] Galilei, G. *Dialogues Concerning Two New Sciences*. New York: Macmillan, 1914.
- [3] Gilovich, T. *How We Know What Isn't So: The Fallibility of Human Reason in Everyday Life*. New York: Free Press, 1993.
- [4] Hofstadter, D. *Metamagical Themes: Questing for the Essence of Mind and Pattern*. New York: Basic, 1985.
- [5] Novak, J. *Learning, Creating, and Using Knowledge: Concept Maps as Facilitative Tools in Schools and Corporations*. New York: Routledge, 2009.
- [6] Pinker, S. *How the Mind Works*. New York: W.W. Norton, 1997.
- [7] Wolfram, S. *A New Kind of Science*. Canada: Wolfram Media, 2002.

This work has been supported in part by a grant from the Teagle Foundation.

USING EARLY INSTRUCTION SETS TO INTRODUCE COMPUTER ARCHITECTURE*

David L. Tarnoff

Department of Computer and Information Sciences - 70711

East Tennessee State University

Johnson City, TN 37614

423 439-6404

tarnoff@etsu.edu

ABSTRACT

Motivating computer science students to study computer architecture can be difficult, especially when complex, modern architectures such as the Intel® Core™ i7 are held up as examples. The solution presented here is to introduce computer architecture by first teaching students how to program simpler, historic machines such as Konrad Zuse's Z1, the Manchester Baby, and the Princeton IAS machine. The limitations of these machines can then be used to motivate students to learn the principles of addressing modes, instruction set architectures, and CPU register design. These early architectures can demonstrate how much, and how little, has changed in the area of computer architecture. This paper summarizes the results of research into the machine language instruction sets and architectures of the Zuse Z1, the Manchester Baby, and the Princeton IAS machine. It then presents examples of how these instruction sets can be used to motivate students in a senior-level computer architecture course.

INTRODUCTION

The first week of a course in computer architecture might acquaint students with a brief history of computing. The instruction might go something like, "The IBM Automatic Sequence Controlled Calculator for which the Harvard architecture is named contained 500 miles of wire 3,300 relays, and 1,464 ten-pole switches for the input of decimal constants. It weighed 10,000 pounds and required a five horsepower motor to

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

turn a 50' shaft that drove the calculator's mechanical components." [4] These facts, while accurate, do not encourage a student to go deeper into the topic, nor do they introduce the architecture used by these machines.

The goal of this work is to use very early instruction sets to motivate students to begin thinking about architectural issues. A study of these machines should also show students that concepts such as pipelining, microcode, and parallel processing have been around since Charles Babbage designed his Analytical Engine in 1838 [6].

The next few sections of this paper present a brief overview of the architectures of three early machines. Following these sections is a discussion of how these architectures can be used to support the study of modern computer architecture.

KONRAD ZUSE'S Z1

From 1936 to 1938, Konrad Zuse constructed his Z1, an automaton he had designed in order to perform the time-consuming calculations necessary in his occupation as a civil engineer [5]. It incorporated a number of important features still used in modern computing including a floating-point, binary numbering system, an addressable memory to store data, distinct decimal input and output units, an arithmetic unit with a carry look-ahead adder, and a control unit that used a two-stage execution pipeline. The machine itself was mechanical although Zuse later developed machines with electromechanical relays. His third machine, the Z3, was identical in architecture to the Z1 except that it replaced the unreliable mechanical elements with electromechanical relays and added a square root function [7].

The Z1 used a Harvard-style architecture executing programs directly from a punch tape reader (35mm standard movie film) while it stored data in a separate memory. In the case of the Z1, storing the programs in memory was avoided because of the expense of the memory [5].

Zuse's floating-point data format is remarkably similar to the IEEE Standard for Floating-Point Arithmetic (IEEE 754) in common use today. The most significant bit is a sign bit for the value. The next seven bits contain the base-two exponent coded in two's complement representation. The last fourteen bits are the significant without the "hidden bit", i.e., the most significant bit of the significand, which is always a one for non-zero values. Zero was represented with an exponent of -64 [7].

The Z1 instruction set consisted of eight 8-bit instructions including two for memory transfers and one each for input and output. The remaining instructions performed mathematical operations on the arithmetic unit's two floating-point registers, R1 and R2. Addition, subtraction, multiplication, and division combined R1 and R2 placing the result in R1 and clearing R2 [7].

The memory load instruction's operand field held the address from which data was to be loaded, but did not identify the destination register. This was because the first load would load R1 while subsequent loads loaded R2. The memory store instruction's operand field also did not identify a register. All stores would store R1 to memory and clear R1. A store would reset the loading process to load R1 next.

Table 1 presents the Z1 instruction set along with each instruction's opcode [7].

Table 1: Z1 Instruction Set [7]

Instruction/Description	Opcode	Cycles
Lu - read keyboard	01 110000	9 to 41
Ld - display result	01 111000	9 to 41
Pr z - load from address z	11 Z6Z5Z4Z3Z2Z1	1
Ps z - store to address z	10 Z6Z5Z4Z3Z2Z1	0 or 1
Lm - multiplication	01 001000	16
Li - division	01 010000	18
Ls1 - addition	01 100000	3
Ls2 - subtraction	01 101000	4 or 5

What the Z1 lacked was program flow control. There were no conditional branches. Loops, therefore, could only be implemented by connecting the paper tape's ends to form a physical loop in the code. There was also no way to implement decision structures [7].

MANCHESTER SMALL-SCALE EXPERIMENTAL MACHINE

In 1948, a proof of concept for an electronic memory called the Williams Tube resulted in the Manchester Small-Scale Experimental Machine (SSEM), the first stored-program computer. The SSEM, nicknamed "Baby," had a 32x32-bit memory, a 32-bit accumulator for intermediate results, and a register, CI, that points to the address of the current instruction [2].

Both instructions and data were stored using 32-bit words. The instruction contained a 13-bit address (bits 0 to 12) and a 3-bit function field (bits 13 to 15). Because the SSEM had a 32-word memory, only the first five bit positions were required for the address.

Figure 1 presents the SSEM instruction layout. The leftmost bit, bit 0, represented the least significant digit. It was common at the time of the SSEM development to view bits as if they were arranged along an X-axis with magnitude increasing from left to right. This was also true for numeric values, which were represented in 32-bit two's complement form [2].

Bit position:	0	1	2	3	4	5..12	13	14	15	16..31
Function:	Memory address		Unused		Function		Unused			

Figure 1: SSEM Instruction Layout [2]

The SSEM had a small instruction set, each function of which was identified using three bits. One function accessed the arithmetic unit, which was capable only of subtraction and negation. For flow control there were two unconditional jump functions, a conditional jump function, and a stop function. One function for loading and one function for storing allowed the SSEM to access data in memory. Table 2 presents the SSEM instruction set along with the binary function codes for programming it [2].

Program flow was controlled with the control register CI. Before an instruction was loaded, CI was automatically incremented by one to the next memory address. This meant

that if the SSEM initialized CI to 0 before running a program, the first instruction to be executed would be at location 1, not 0. This also meant that the destination address for jump instructions was given as one less than the desired address.

Table 2: SSEM Instruction Set [2]

Binary Code	Modern Mnemonic	Instruction/Description
000	JMP	Load CI w/address stored in location pointed to by instruction
100	JRP	Retrieve offset stored in location pointed to by instruction and add to CI
010	LDN	Copy the negated contents of the memory location to accumulator
110	STO	Store contents of accumulator to memory location
001	SUB	Subtract the contents of the memory location from the accumulator
101	-	Not defined (If used, SSEM performed a SUB)
011	CMP	Skip the next instruction if the content of the accumulator is negative
111	STOP	Halt the machine

The CMP function also affected CI. If a CMP instruction was encountered and the accumulator contained a negative value, CI was incremented a second time thereby skipping the subsequent instruction [2].

The SSEM's two jump commands are considered special cases in modern computing. The JMP function is now known as an indirect jump. This means that the memory address referenced by the instruction contained the address to be jumped to unconditionally. The JRP function is now known as an indirect relative jump. This means that the value stored at the address pointed to by the instruction represents an offset from the current instruction. Although JRP is not necessary for a machine of SSEM's simplicity, it was added because its cost was trivial, yet it allowed for relocatable code. For example, a loop with an instruction that says, "jump 5 memory locations back" instead of using a hard coded address can be placed anywhere in memory [2].

PRINCETON IAS MACHINE

After having worked on the Electronic Numerical Integrator and Calculator (ENIAC) with John W. Mauchly, a professor at the University of Pennsylvania's Moore School of Electrical Engineering, and J. Presper Eckert, Mauchly's graduate student, John von Neumann, a mathematician at Princeton's Institute for Advanced Study (IAS), developed a new design for a stored program architecture. This work was motivated by the difficulties with programming the ENIAC manually by setting switches and routing cables, an arrangement that was time consuming and did not allow for the storing of programs for later reload and execution. With Mauchly and Eckert's input, von Neumann wrote a report titled, "First Draft of a Report on the EDVAC," which proposed a machine that stored instructions in memory alongside the data [9]

John von Neumann then collaborated with Arthur Burks and Herman Goldstine from Princeton's Institute for Advanced Study (IAS) to develop a new machine based on the EDVAC design, a machine later known as the IAS machine. The IAS machine, described in their report titled, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," had the same architecture as that described in the EDVAC report although the later paper went into more detail regarding implementation including a description of the instruction set [1].

As implemented, the Princeton IAS machine had a 1,024 word binary memory, each location of which could store a 40-bit word. When used to store data, the 40-bit location used two's complement representation of a fraction, i.e., a fixed point value between -1 and 1. When used to store an instruction, each 40-bit location stored two 20-bit instructions identified as "left" and "right". The left instruction was executed first [3].

Each 20-bit instruction was partitioned into four parts: a 10-bit memory address, a step digit used to halt or continue operation, an 8-bit opcode, and a spare bit that identified special operations. The 10-bit address allowed the IAS machine to access any location in its 1,024 word memory. The step digit, if set to 0, halted the machine after the execution of an instruction without incrementing the program counter. If the step digit was set to 1, the program counter was incremented and the next instruction was executed. The 8-bit opcode defined the operation to execute. An instruction's last bit was 0 except in the case of a special order instruction [3]. Figure 2 presents the layout of the IAS instruction word.

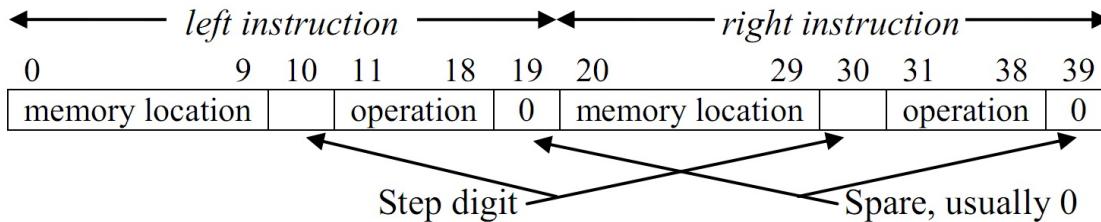


Figure 2: Princeton IAS Machine Instruction Layout

The IAS machine had seven registers for storing binary values. Two 40-bit registers, AC and MQ, in the arithmetic unit maintained the sources for and results of arithmetic operations. Five registers in the control unit maintained the step-by-step sequencing of instructions. These registers were the program counter (PC), the memory buffer register (MBR), the memory address register (MAR), the 40-bit instruction buffer register (IBR), and the register to hold the currently executed instruction (IR).

The final progress report on the Princeton IAS machine as submitted by Goldstine, Pomerene, and Smith also presented a detailed description of the instruction set including opcodes [3]. This description is summarized in Table 3.

In addition to the instructions presented in Table 3, there were six I/O instructions for transferring data between memory and the outside world. Over the course of its life, the IAS machine used I/O including teletypewriters, punch cards, and magnetic wire [10].

Table 3: Princeton IAS Machine Instruction Set [3]

Opcode (binary)	Operation description
11100101	Load AC with value from address in memory
11100100	Add value from address in memory to the AC
11101101	Load AC with negative of value from address in memory
11101100	Subtract value from address in memory from the AC
11110101	Load AC with absolute value of number from memory
11110100	Add absolute value of number from memory to the AC
11111101	Load AC with negative absolute value of number from memory
11111100	Subtract absolute value of number from memory from the AC
11100000	Multiply MQ with value from address in memory; result in AC:MQ
11100010	Multiply MQ with value from address in memory; upper half in AC
11111000	Divide AC by value from address in memory; AC=remainder, MQ=quotient
11100110	Load MQ with value from address in memory
11010100	Store AC to address in memory
11010101	Store AC to address in memory then clear AC
11010000	Unconditional jump to left instruction of destination address (step = 0)
11010000	Unconditional jump to right instruction of destination address (step = 1)
11011000	If AC ≥ 0 , jump to left instruction of destination address (step = 0)
11011000	If AC ≥ 0 , jump to right instruction of destination address (step = 1)
10100000	Shift AC right one position with LSB shifting into MQ
10100010	Shift AC right one position discarding LSB
10101000	Shift AC right one position discarding MSB
10100100	Transfer MQ to AC
N/A*	Replace address portion of left instruction at address w/leftmost 10 bits in AC
N/A*	Replace address portion of right instruction at address w/leftmost 10 bits in AC

CLASSROOM APPLICATION

For the past two years, the author has opened his senior-level computer architecture course by presenting the details of these machines along with Charles Babbage's 1838 Analytical Engine design and the architecture and instruction set of the IBM ASCC. The students are familiarized with the machines through assignments such as writing a simulator for the IAS machine or writing code for the Manchester Baby, which can be executed using on-line simulators [8].

The machines are then referred to throughout the semester as an introduction to different architectural topics. For example, addressing modes can be introduced through a classroom discussion on the difficulties in creating constants or pointers using the IAS machine. Each of these machines only supported one numeric representation allowing the students to see the benefit of numbering systems and their effect on a machine's

applications. The Baby's relative jump command can serve as a jumping off point for a discussion on relocatable code. Microcode can be introduced using the description of Charles Babbage's use of studs screwed into music box-style barrels to implement instructions. The binary instructions of the IAS machine were partitioned into fields aiding in instruction set implementation. All but the IAS machine were Harvard architecture due to the expense of memory, so a comparison can be made between the IAS machine and the others to show how Harvard differs from von Neumann. Almost every one of the machines described in this paper implemented a two-stage pre-fetch pipeline. Finally, all of these systems had to be halted in order to perform user I/O, which can be used as a starting point to discuss polled I/O and then interrupts.

Since implementing the changes outlined in this document along with the use of a single-board MIPS processor, the students' satisfaction with the course and their feelings for how worthwhile it was improved dramatically. As measured by the university's student assessment of instruction tool, their rating of how worthwhile the computer architecture course was increased on a scale of 1 to 4 from a 2007 mean of 2.86 to a 2010 mean of 3.63.

CONCLUSION

Many of the fundamental concepts of computer architecture have existed since the first computing automatons were created. The complex implementations of modern computers present a challenging learning curve for even the brightest students. By covering the fundamentals of computer architecture by holding up early architectures as an example, students appear to gain a better understanding of how architecture affects computing and why certain implementations are so important.

BIBLIOGRAPHY

- [1] Burks, A. W., Goldstine, H. H. and von Neumann, J., *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*, Princeton, NJ: Institute for Advanced Study, 1946.
- [2] Burton, C. P., The Manchester University Small-Scale Experimental Machine programmer's reference manual, *Computer Conservation Society*, 2, 1997.
- [3] Goldstine, H. H., Pomerene, J. H. and Smith, C. V., *Final Progress Report on the Physical Realization of an Electronic Computing Instrument*, Princeton, NJ: The Institute for Advanced Study, 1954.
- [4] IBM, IBM's ASCC (a.k.a The Harvard Mark I),
www-03.ibm.com/ibm/history/exhibits/markI/markI_intro.html, retrieved December 3, 2010.
- [5] Maxfield, C., The life and work of Konrad Zuse,
www.techbites.com/20090911498/myblog/articles/z000c-the-life-and-work-of-konrad-zuse-index.html, retrieved December 3, 2010.

- [6] Myhrvold, N. & Swade, D., An evening with Nathan Myhrvold and Doron Swade: Discussing Charles Babbage's Difference Engine, 2008, www.computerhistory.org/events/index.php?id=1206647564, retrieved December 3, 2010.
- [7] Rojas, R., Konrad Zuse's legacy: The architecture of the Z1 and Z3, *IEEE Annals of the History of Computing*, 19, (2), 1997.
- [8] Sharp, D., Manchester Baby simulator, www.davidsharp.com/baby/, retrieved December 3, 2010.
- [9] von Neumann, J., First draft of a report on the EDVAC (Original publication in 1945), *IEEE Annals of the History of Computing*, 15, (4), 1993.
- [10] Ware, W. H., *The History and Development of the Electronic Computer Project at the Institute for Advanced Study*, Santa Monica, CA: RAND Corporation, 1953.

STAGING A REALISTIC ENTITY RESOLUTION CHALLENGE

FOR STUDENTS*

*Yinle Zhou and John Talburt
Information Science Department
University of Arkansas at Little Rock
Little Rock, AR 72204
973 610-8735
yxzhou@ualr.edu
jrtalburt@ualr.edu*

ABSTRACT

This paper describes the experience of constructing and deploying a significant exercise in entity resolution as a way to more closely simulate the challenges often encountered in real-world data integration projects. Based on a consistent set of synthetically generated demographic data that have been separated and disrupted in a controlled manner, the datasets used in the exercise are large enough (several thousand records) to provide students with a significant challenge yet small enough to be managed within a semester course using tools that will run on a desktop platform. Because the starting state of the integrated data is known, student progress in re-integrating the data can be readily and objectively measured to give students feedback on their progress and also allowing them to assess the effectiveness of different strategies and approaches they might try. The details given here are based on the experience of conducting the ER challenge on three occasions in two different courses.

INTRODUCTION

Entity Resolution (ER) is the process of determining whether two references to real-world objects are referring to the same object or to different objects. The term entity describes the real-world object, a person, place, or thing [1]. References referring to the same object are said to be equivalent references.

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

ER is a topic of growing importance for business and government. Many researchers have been involved in this area since late 1950s. Newcombe et al [2] introduced the ideas of computerized record-linkage in 1959. Fellegi and Sunter [3] laid the foundation for a formal model of record-linkage by demonstrating a process for optimizing integration rules that satisfied preset bounds on the maximum rates of false positive and false negative decisions. Winkler [4] showed how to estimate the model parameters using the Expectation-Maximum (EM) algorithm. Researchers at the Stanford University InfoLab developed and described a more generic model called the Stanford Entity Resolution Framework (SERF) [5, 6] as well as a series of algorithms for carrying out entity resolution in specific contexts.

The creation of the ER Challenge was in response to suggestions from industry advisors that graduates from computing programs would be better prepared if they were given exercises that reflected more of the challenges that they are likely to encounter in a real-world projects. Some of these challenges include dealing with missing or incomplete metadata, assessing and improving data quality issues in data sources, designing an integration strategy, selecting and using appropriate tools, repeated trials and evaluations, and working in a team environment.

ENTITY RESOLUTION COURSE

Since 2009, the University of Arkansas at Little Rock has offered a graduate-level course titled Entity Resolution and Information Quality. The course systematically introduces Entity Resolution as an information science discipline with classes conducted into two parts: lectures on ER theory and complementary laboratory exercises. The lectures introduce ER foundations, principles, and methods. It also covers three major ER models: the Fellegi-Sunter Model of Record Linkage, the SERF Model of generic ER and consistent ER, and the Algebraic Model of ER. The course also includes studies of commercial ER offerings such as Acxiom® AbiliTec® [7] and published case studies of ER projects such as those found in Herzog [8]. In the laboratory portion of the course, students work with ER systems such as DataFlux® dfPowerStudio®, Infoglide® Identity Resolution Engine®, and OYSTER (Open sYSTem Entity Resolution Engine). The challenge project is assigned as a team project early in course to be submitted by the end of the semester. The purpose of the project is to give students hands-on experience in solving realistic entity resolution problem using the knowledge and skills gained in the course.

CREATING THE SOURCE DATA

Although ER can be applied to a number of entity types such as products and events, most ER is still concerned with personal identity in various roles such as customers of a business, patients of a hospital, or students in a school system. However growing concerns over individual privacy and identity theft have created an obstacle to working with real personally identifiable information (PII). In the current environment, organizations are very reluctant to allow any external access to this kind of information. The result is that it can be very difficult to obtain personal identity information to use for student exercises or to experiment with entity resolution methods and techniques. Even

in cases where this kind of data is available, it may not contain examples of a particular feature of interest, such as, records for the same person at different addresses or instances where a person has changed his or her name.

To address this problem, SOG (Synthetic Occupancy Generator) [9] was created as a tool to automatically generate a large number of realistic, but synthetic occupancy histories. SOG starts by scrambling publicly available name and address data then adding randomly generated dates-of-birth, social security numbers, and telephone numbers to produce realistic, but synthetic personal identities in a consistent set of occupancy (residence) histories. After the occupancy histories are produced, a second process separates and disrupts these histories into a collection of files that serve as the input sources for ER challenge problem. The first level of disruption is that each source file has a different format and subset of the identity attributes. For example some sources may have a date-of-birth attribute while other may not. Some sources may have the name in a single field and other may have it separated into first and last name fields. In addition, each history record may be duplicated in the same source or across several different sources.

A second level of disruption involves the introduction of data quality errors such as omitting attribute values, using different field formatting (e.g. in dates and phone numbers), transposing or deleting characters in string values, and replacing attribute values with common aliases, misspellings, and abbreviations. An important aspect of the disruption process is that during the disruption, it builds a cross-reference index between the identity represented in a history record and the record identifier of each source record that was created from that history record. The cross-reference index can then be used to evaluate the effectiveness of each team's integration effort by comparing the team's grouping of source records to the correct grouping according to the original identity used to create them.

AN EXAMPLE CHALLENGE

In a typical challenge problem, the students are given four source files to resolve. The following is a description of the most recent version. The description of the ER challenge and the four files described below are freely available for download from <http://ualr.edu/eriq>.

Three of the files are lists of entity references where the entities are individuals living in the United States. These files are named List A, B, and C and have the following characteristics.

- List A – a comma-separated file with 94,306 records. The fields include the record ID, name, street address, city-state-zip, PO Box, PO Box city-state-zip, social security number, and date-of-birth.
- List B – a pipe-delimited file with 100,777 records. The fields include record ID, first name, last name, street number, primary address, secondary address, city, state, zip code, and phone number
- List C – fixed-length field format with 76,059 records. The fields include first name, middle initial, last name, social security number, date-of-birth, and telephone number.

The fourth text file is a change of address (COA) file. The COA file allows students to connect and group records when an individual has move from one address to another.

- COA – a pipe-delimited file with 40,174 records. The fields include last name, the address elements from the former address (street number, street name, secondary, etc.) followed by the address elements of the new address.

The Data Challenge requires students to analyze the quality and characteristics of the files and partition the 271,142 references from Lists A, B, and C into groups or clusters of those that they deem to be equivalent references, i.e. represent a single individual identity. Each cluster is assigned a unique identifier called a Link ID.

The teams are asked to submit two results. One is a report which describes the strategy, tools used and matching rules. The second is a link index file that gives their resolution decisions. Each of the 271,142 records in the link index file has two values. The first value is the source record ID and the second value is the Link ID.

```
RecID,LinkID
B898130,0
A914385,0
A916572,0
C984402,0
B959775,1
C956489,1
C953606,1
```

Figure 1 Link Index Segment

An example segment of a link index file in comma-delimited format is shown in Figure 1. It shows the decision that the first four source records (one from List B, two from List A, and one from List C) are all references to the same individual. In addressing the challenge, students taking the course have access to several entity/identity resolution and information quality applications installed on a remote access server. A typical solution strategy has the following steps:

1. Assess the quality and characteristics of each list
2. Design and execute appropriate data cleansing and standardization operations
3. Establish an initial set of identity attributes and agreement (matching) rules
4. Resolve the references
5. Evaluate and analyze the results
6. Implement improvements and repeat previous steps

Assessing Quality

The source files are first loaded into MySQL or Microsoft® Access® tables. Teams using DataFlux can establish an ODBC connection to these tables and use a feature called DBViewer® to display the table contents and executing SQL queries against the data.

Viewing the data will immediately expose a number of data quality issues in the sources. For example in the name field there is no consistent use of upper and lower letters. Some name fields are all in upper case, some lower case, and many in mixed case. Some individual names are given only as initials, e.g. "L Brown" and there is

considerable variation in abbreviations such as "Avenue", "Ave" and "Av" in the street name suffix. Similarly there is an inconsistency in the representation of state names and abbreviations such as "Texas" and "TX".

Figure 2 shows the result of performing a pattern analysis on the telephone number field of List B using the DataFlux data profiling tool. It shows that even though this column is more than 97% complete, the values are formatted in six different ways.

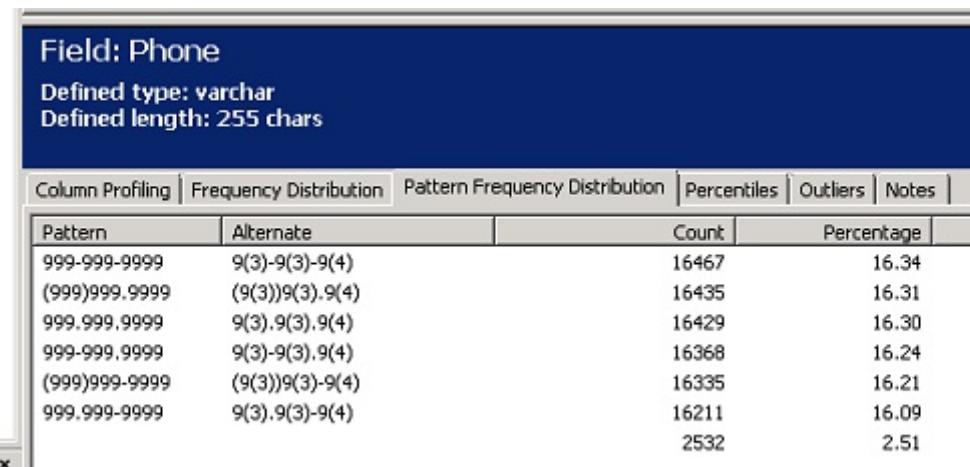


Figure 2: Pattern Frequency Distribution for List B Phone Number Field

These and other data quality assessments of Lists A, B, and C will reveal a number of problems that the students must address before attempting to resolve the references.

Data Cleansing

Most of the data quality issues discussed above can be addressed through data standardization. Student can write their own code to perform standardization functions, however most elect to use the standardization capability provided as part of the DataFlux application. To further facilitate the processing, standardization and other functions such as parsing can be automated through a graphical workflow interface called Architect®. Architect allows the students to drag and drop symbols that represent these functions onto a palate and connect the symbols with arrows to indicate the order of processing.

Establishing Agreement Rules

The agreement patterns for directly matching references are based on the identity attributes related to name, address, date-of-birth, social security number, and phone number. One issue that the students face is that the name and address attributes are sometimes given as composite fields. For example in List A the name is represented in a single field whereas in List B it is separated into two fields, one for first name and one for last name. Addresses have even more components including street number, pre-directional, street name, street suffix, post-directional, secondary identifier, secondary number, state name, city name, and zip code.

Rather than attempting to use all of these attributes, a typical strategy is to select only those key components that provide the best matching performance. This may require additional preparation steps to separate (parse) larger values into their constituent components such as extracting the street name from a larger address field.

Furthermore students must make decisions related to the level of match, i.e. allowing approximate match versus exact match. The level of approximate match will vary by the type of attribute. For example, the requirement for social security numbers may be an exact match, or at most the transposition of adjacent digits, whereas the first name could be an approximate or a nickname (alias) match, e.g. Bob allowed as a match to Robert. Typical agreement rules might look like the following

- Rule 1: Social security numbers are an exact match
- Rule 2: Last name is an exact match and social security number is an exact match or varies only by the transposition of two adjacent digits
- Rule 3: First name, last name, street name, and city name are an exact match
- Rule 4: First name is a nickname match, and last name, street name, and city name are exact
- Rule 5: First name is an approximate match and last name and date-of-birth are exact

Resolving the References

Once the agreement rules have been designed they must be implemented and executed. The implementation of the rules will depend upon the application used to perform the resolution. In the OYSTER resolution engine the rules are implemented as an XML script. For example Rules 1 and 2 shown above would be represented in OYSTER as

```
<IdentityRules>
  <Rule Ident="1"><Term Item="SSN" MatchResult="Exact"></Rule>
  <Rule Ident="2"><Term Item="LastName" MatchResult="Exact">
    <Term Item="SSN" MatchResult="Transpose"></Rule>
</IdentityRules>
```

Evaluating and Analyzing the Result

The evaluation of the result is easily done by the instructor. The evaluation of a team's ER result is the degree to which their groupings of the source records agree with the correct groupings from the original occupancy histories generated by SOG. There are a number of ways to compare the similarity of grouping (partitions). One measure that was specifically developed to compare ER results and relatively easy to compute is the Talbert-Wang Index (TWI) [11].

If A and B are two partitions (groupings) of a set S, and V is the set of overlaps between A and B (i.e. the set of non-empty intersections between groupings in A and B), then the TWI measure of similarity between A and B is defined as

$$\text{TWI} = \frac{\sqrt{|A| \cdot |B|}}{|V|}$$

The value of TWI will always be a value between 0 and 1, and its value will only be 1 when the two groupings A and B are identical. A smaller value of the TWI indicates less similarity between the groupings.

In the case, where A is the correct grouping and B is a team result, then the TWI value can be interpreted as entity resolution accuracy. The optimum result from entity resolution will be when the groupings are identical giving a TWI value of 1.

	True	Team 1	Team 2	Team 3	Team 4	Team 5
T-W Index	1.0000	0.6250	0.5227	0.4483	0.3970	0.2587
Group Count	20,067	41,418	65,185	17,541	127,108	1,489
Overlap Count	20,067	46,126	69,192	41,848	127,207	21,062
Avg. Group Size	13.5	6.5	4.2	15.4	2.1	180.4

Table 1 Team Evaluation Results

Table 1 shows typical results as submitted by 5 teams attempting the challenge. At this point, Team 1 has achieved the best results with an index of 0.625. Teams 4 and 5 are examples of where the processes exhibit pronounced under-grouping and over-grouping, respectively. On average, Team 4 is only finding single pairs of equivalent source records as shown by average grouping size of 2.1. On the other hand, Team 5 has consolidated all of the source records into only 1,489 groups compared to the correct result of 20,067 groups. Individual teams can ask for an evaluation of their intermediate results to check on their progress and as input to the analysis of their process.

CONCLUSION

The ER Challenge has proved to be very successful in engaging students in a realistic team project that exercises their knowledge, creativity, and problem solving skills. It is an interesting problem in that it has a known correct result, but for which there are many possible and equally viable solution strategies. To be successful, the teams must make a number of evaluations and decisions regarding data quality, data transformations, identity rules, and tools for implementing the integration process. Although the TWI provides a simple way to measure progress and establish team rankings, the final grade for the project gives more weight to the project design and team innovation than their TWI rank compared to other team.

ACKNOWLEDGEMENTS

The academic license for use of dfPowerStudio® was provided by SAS DataFlux®. The academic license for the use of the Identity Resolution Engine® was provided by Infoglide Software®. Partial support for the development of OYSTER was provided by a grant from the Arkansas Department of Education.

REFERENCES

- [1] Talburt, J., *Entity Resolution and Information Quality*, Morgan Kaufmann, 2010.
- [2] Newcombe, H., Kennedy, J., Axford, S., James, A., Automatic linkage of vital records. *Sci.* 130, 3381, 954-959, 1959
- [3] Fellegi, I., Sunter, A., A theory for record linkage, *J.Amer. Statist. Assoc.* 64, 328, 1183-1210
- [4] Winkler, W., Improved decision rules in the Fellegi-Sunter model of record linkage, Statistical Research Report Series RR93/12, U.S. Bureau of the Census, Washington, D.C. 1993
- [5] Benjelloun, O., Garcia-Molina, H., Kawai, H., Larson, T., Menestrina, D., Su, Q., Thavisomboon, S., Widom, J., Generic Entity Resolution in the SERF Project, *IEEE Data Engineering Bulletin*, 2006
- [6] Benjelloun, O., Garcia-Molina H., Menestrina, D., Su, Q., Whang, S., Widom, J., Swoosh: A Generic Approach to Entity Resolution. *The VLDB Journal*, 2009
- [7] AbiliTec customer data integration software, 2010, <http://products.enterpriseitplanet.com.dms/im/987435833.html>
- [8] Herzog, T., Scheuren F., Winkler W., *Data Quality and Record Linkage Techniques*, Springer, 2007
- [9] Talburt, J., Zhou, Y., Shivaiah, S., SOG: A Synthetic Occupancy Generator to Support Entity Resolution Instruction and Research, *Proceeding of the 14th International Conference on Information Quality*, 96-105, 2009
- [10] DataFlux, 2010, <http://www.dataflux.com>
- [11] Talburt, J. Wang, R., Hess, K., Kuo, E., An algebraic approach to data quality metrics for entity resolution over large datasets, *Information Quality Management: Theory and Applications*, L. Al-Hakim, Ed. Hershey, PA: Idea Group Publishing, 1-22, 2007

SENSITIVITY OF DIFFERENT MACHINE LEARNING ALGORITHMS TO NOISE*

Abhinav Atla
Computer Science Department
University of Central Arkansas
Conway, AR 72034
abhinav655@gmail.com

Victor Sheng
Computer Science Department
University of Central Arkansas
Conway, AR 72034
ssheng@uca.edu

Rahul Tada
Computer Science Department
University of Central Arkansas
Conway, AR 72034
t.rahlreddy@gmail.com

Naveen Singireddy
Computer Science Department
University of Central Arkansas
Conway, AR 72034
naveen.singireddy@gmail.com

ABSTRACT

Noise in data is an effective cause of concern for many machine learning techniques that are used in modeling data. Researchers have studied the impact of noise only on some particular learning algorithm, but only very few attempted to analyze the effect of noise on different ones. In this work, we study the noise sensitivity of four different learning algorithms under different intensities of noise. Particularly, we compare the noise sensitivity of decision tree, naïve bayes, support vector machine, and logistic regression. The algorithms are tested on different datasets that are artificially injected with different degrees of noise. The study helps us understand the impact of different levels of noise on the learning algorithms mentioned above. Furthermore, it also guides of choosing the learning algorithms. In general, naïve bayes is the most resistant to noise. However, it performs also the worst. The other algorithms perform much better than naïve bayes especially after the noisy level is lower than 40%. When we have approaches to improve the data quality (reduce the noise level), decision tree is the most preferred one, followed by support vector ma

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1. INTRODUCTION

Data extracted from real-world problems using supervised learning techniques frequently contains noise. Noise decreases the quality of the data and might affect the learning process leading to inaccurate models. However, the supervised learning algorithms are developed based on the assumption of clean data. This research is to discover the performance of learning algorithms with noisy training data. It will investigate the noise sensitivity of learning algorithms with various percentages of noise. The results of this research can assist choosing proper learning algorithms. It can recommend which learning algorithm is preferred under what level of noise. Furthermore, if we can reduce the noise level, which algorithm should be chosen? In this work we are concerned with inaccuracies that migrate from data contaminated with random errors. Random errors can be defined as the uncertainty detected by executing learning algorithms with different conditions of noise. In this work we utilize classifiers such as decision tree, naïve bayes, support vector machine, and logistic regression to perform classification efficiently on different datasets. We study the performance of these algorithms under different noise levels from 0% (no noise) to 50% (very noisy).

The paper is organized as follows. A brief summary of the related literature on handling noise is provided in section 2. Section 3 reviews the different learning algorithms used to analyze their sensitivity under different noise levels. Sections 4 and 5 presents an explanation of the mechanism used for generating noise and the analysis of the experimental results for the four machine learning algorithms. Finally Section 6 provides conclusions about the performance of the algorithms on different datasets and various levels of noise.

2. RELATED WORK

Real-world data always contain noise. In order to apply learning algorithms, data preprocessing is the first step. There are many tasks during data preprocessing. Among them, data cleaning is one of the important tasks. There are many works on detecting noise and reducing noise (Xiong 2006; Zhu et al., 2003, 2006; Kubica 2003; Brodley 96, 99). There are two types of noise: attribute noise and class noise (Zhu et al., 2003, 2006). Brodley (Brodley 96, 99) detected and removed the class noise. Most of the work has been done on noise detection focused on supervised classification problems. However, Xiong et al. (Xiong et al., 2006) studied the approach of remove noise for unsupervised learning. Kubica & Moore (Kubica 2003) studied how to identify and remove the attribute noise, such that the remaining information in the training examples can still be used in modeling. Different from previous work, this paper focuses on the impact of class noise on supervised learning algorithms and investigates the noise sensitivity of different learning principles through analyzing the performance of learning algorithms under different level of noise. Thus, we can choose the proper learning algorithm. This is also needed even if the detecting and reducing noise approaches are applied to improve the data quality.

3. LEARNING ALGORITHMS

There exist more than 20 different learning algorithms, including the basic ones and improved variations. According to the categorization of WEKA (Witten and Frank), we choose the fundamental one from each category. That is, we are going to investigate the noise sensitivity of four basic learning algorithms. They are decision tree, naïve bayes, support vector machine, and logistic regression.

A decision tree algorithm (DT in short) partitions the input space into small sets, and labels them with one of the various output categories. That is, it iteratively selects a specific attribute to extend the learning model. According to the values of the specific attribute, a large group of cases are categorized into sub-groups. The essence of the algorithm is the criteria of selecting a specific attribute from the set of attributes available. There exist several criteria, such as accuracy improvement, entropy reduction, information gain, and gain ratio (details of these criteria can be found in (Mitchell machine learning book)). The noisy label information will directly affect the estimation of all the criteria. Thus, the model built on decision tree algorithm would be affected by the noisy label.

Naïve bayes (NB in short) (John and Langley 1995) is based on bayes theorem. Specifically, it is based on the properties of estimating the frequency of each value of every attribute under a given class from the obtained dataset. We can image that there is no difference of these estimation with/without noisy labels if both the class distribution and the distribution of the noisy label follow the complete random distribution. That is, naïve bayes is noisy-insensitive, particularly in estimating the conditional probabilities of each attribute value under given classes. We will see that there is little difference on the performance of naïve bayes under different noisy levels. Of course, it is not always true that the conditions are satisfied on the real world datasets. Noise still impacts the performance of the model built on naïve bayes.

Support Vector Machine (SVM in short) is one of the kernel approaches for classification. It constructs a hyperplane in high dimension space to classify cases into different classes. Its intuition is to find the hyperplane that has the largest distance to the nearest training cases. These nearest training cases are commonly referred as support vectors. According to the vectors on each side, a sub-hyperplane can be built for each side. The maximum margin between the two sub-hyperplanes is achieved to reduce the general classification errors. When the data have noise, it is possible that these support vectors could have noise too. Thus, the hyperplane and the two sub-hyperplanes (found based on support vectors) can vary. That is why support vector machine is very sensitive to noise. We further discuss this after describing experimental results.

Logistic regression (logistic in short), like naïve bayes discussed above, is part of a category of statistical models called generalized linear models. However, unlike the independent assumption among the variables in naïve bayes, logistic regression has no such assumption. In addition, it also makes no assumption about the distribution of the independent variables. Although both logistic regression and naïve bayes are statistical models, the basic ideas of them are different. Logistic regression estimates the probability of being positive case (for binary classification). It can be inferred that the probability migrates toward the uniform distribution when more noise labels are involved. Thus, it is more noise sensitive. The experimental results also show this (refer to Section 5).

4. EXPERIMENTAL SETUP

To study the noise sensitivity of the four different learning algorithms (decision tree, naïve bayes, support vector machine, and logistic regression), we run experiments on all the classification datasets downloaded from WEKA website. In the experiments, we assume that these datasets are clean (no noise, especially no noise class labels). In order to have the datasets to study the noise sensitivity, we inject noise into these datasets. Since we focus on investigation of the class noise, we only introduce noise into the class labels.

Here is the simulation procedure:

1. Repeat 10 times
 - a. Divide a dataset into training part (70%) and test part (30%)
 - b. Keep the original class labels in array for all training examples in the training part
 - c. For each training example (simulate the noise labels with the control of the noise level)
 - i. If its label is wrong, randomly generate a label from the labels other than the original label to replace the original label
 - d. Build classification model using a specific learning algorithm
 - e. Make prediction for the test examples in test part, and output the classification accuracy
2. Output the average classification accuracy over 10 repeats

We studied the noise sensitivity on the four learning algorithms step by step. We first focus on the impact of noise labels on binary classification. The binary datasets from WEKA website are *bmw*, *expedia*, *kr-vs-kp*, *mushroom*, *qvc*, *sick*, *spambase*, *tic-tac-toe*, and *travelocity*. We further study the performance of each learning algorithm for multiclass classification to see whether the noise impact on the performance on binary classification stays. The multiclass datasets from WEKA website are *anneal*, *audiology*, *autos*, *balance*, *glass*, *heart-c*, *heart-h*, *iris*, *letter*, *lymph*, *primary-tumor*, *segment*, *splice*, *thyroid*, *vehicle*, *vowel*, *waveform*, and *zoo*. Because of the page limitation, we could not show the properties of these datasets here.

From the procedure above, we can see that we have a parameter (noise level) to control the amount of noise introduced. It should be noted that the noise introduced here are random. In details, in the step 1.c.i of the simulation procedure above, for each training example, we randomly generate the noise labels to replace the original labels in the training data. Specifically, for binary classification, if the label of a training example is wrong, its opposite label should be used to replace the original label. For multiclass classification, any one from the rest labels (different from the original label) has the same probability to be used to replace the original one.

5. EXPERIMENTAL RESULTS

In this section, we compare the performance of different machine learning algorithms on the datasets with different percentages of label noise injected. Noise degree is controlled from 50% to 10%. In addition, to see the noise label impact, we also have

the experimental results for each dataset under the four learning algorithms without noise. Evaluation of the machine learning algorithms performance is done by comparing the average test accuracy of the models constructed on the training dataset and tested on the test dataset. The results on the binary datasets are shown in Table 1. The average results overall all nine binary dataset are shown in Figure 1. And the results of the multiclass datasets are shown in Table 2.

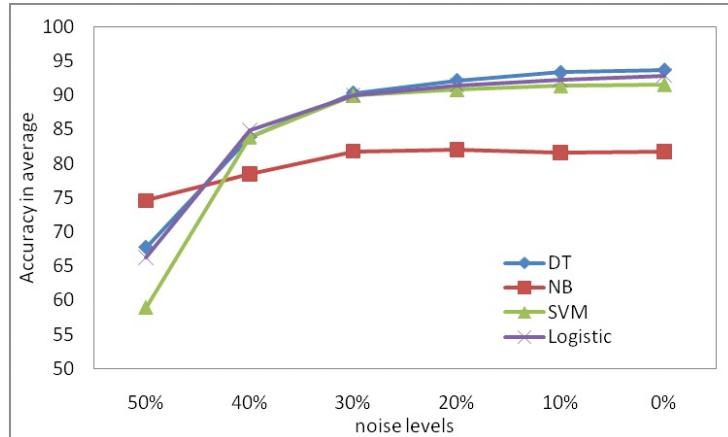


Figure 1: The average accuracy over the nine binary datasets for the four learning algorithms.

From Figure 1 and Table 1, we can conclude that naïve bayes (NB) is the most noise tolerant among the four learning algorithms. Its performance does not change much when the noise level decreases (that is, the label quality increases). Its curve over the noise level is almost flat. It is obvious that naïve bayes always performs the best in average when the noise level is the highest (50%), followed by decision tree (DT) and logistic regression (logistic), followed by support vector machine (SVM). However, when the noise level reduces, other three algorithms perform better. Especially, when the noise level reduces from 50% to 40%, the performance of all the three algorithms improves quickly. Although their performance keeps improving when the noise level continues to reduce, the acceleration of the improvement slows down. Among the three noise sensitive learning algorithms, decision tree performs the best, followed by logistic regression,

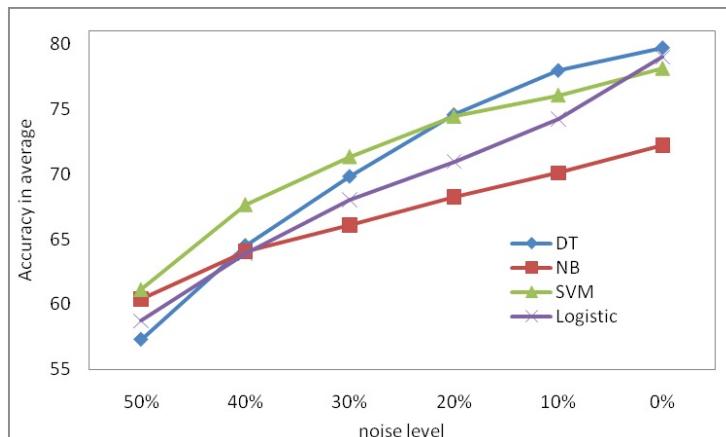


Figure 2. The average accuracy over the 18 multiclass datasets for the four learning algorithms.

followed by support vector machine. In other words, if we have approaches which can improve the quality of the labels, decision tree is the first preference, followed by logistic regression and support vector machine.

Dataset	%Noise	50%	40%	30%	20%	10%	0%
bmrg	DT	80.23	81.70	82.94	85.20	86.28	86.07
	NB	69.92	67.60	68.92	66.74	66.41	65.54
	SVM	76.87	76.87	76.87	76.87	76.87	76.94
	Logistic	76.87	76.81	77.59	78.25	79.08	78.95
expedia	DT	87.39	87.42	90.61	91.97	92.89	93.34
	NB	85.88	85.68	85.93	85.40	84.60	83.54
	SVM	87.45	87.77	88.59	90.85	91.42	91.88
	Logistic	89.20	90.04	91.11	91.32	91.55	91.55
Kr-vs-kp	DT	72.55	83.86	95.70	98.22	99.12	99.24
	NB	66.48	72.93	79.58	83.88	86.10	87.30
	SVM	57.55	83.26	90.65	90.86	94.16	95.38
	Logistic	60.99	73.36	85.71	92.11	95.22	97.44
mushroom	DT	74.69	90.23	99.61	99.87	100.00	100.00
	NB	96.14	96.71	97.32	97.56	97.21	95.87
	SVM	74.09	97.89	99.86	99.98	99.98	100.00
	Logistic	74.63	96.07	99.74	99.93	99.98	100.00
qvc	DT	83.97	85.47	86.95	87.81	88.88	88.95
	NB	69.24	69.77	68.03	67.02	65.86	65.92
	SVM	82.40	82.40	82.54	82.84	83.27	83.26
	Logistic	82.79	82.98	83.50	83.64	83.26	83.12
sick	DT	47.52	84.08	95.03	97.24	98.62	98.75
	NB	40.37	64.17	83.16	82.93	88.74	92.14
	SVM	24.79	80.52	93.58	94.08	94.08	94.08
	Logistic	35.59	84.04	94.18	95.20	95.28	96.61
spambase	DT	47.57	86.65	88.09	89.41	91.17	92.38
	NB	85.22	86.02	86.36	85.35	77.22	79.45
	SVM	41.68	59.72	87.49	90.43	90.58	90.68
	Logistic	55.77	76.96	86.40	90.46	91.72	92.88
tic-tac-toe	DT	65.51	71.74	77.77	80.24	83.07	84.18
	NB	67.32	69.16	71.71	73.94	73.07	70.94
	SVM	65.75	95.61	96.48	97.70	98.22	98.22
	Logistic	80.59	91.15	95.26	96.17	97.91	97.91
travelocity	DT	50.09	83.14	95.87	99.08	99.63	99.71
	NB	90.54	94.32	94.90	94.87	94.89	95.01
	SVM	19.64	90.47	93.02	93.09	93.10	93.18
	Logistic	40.02	91.89	96.45	95.81	95.76	97.20
average	DT	67.72	83.81	90.29	92.12	93.30	93.62
	NB	74.57	78.48	81.77	81.97	81.57	81.75
	SVM	58.91	83.83	89.90	90.74	91.30	91.51
	Logistic	66.27	84.81	89.99	91.43	92.20	92.85

Table 1. The accuracy of the four learning algorithm on binary classification.

We further study the noise sensitivity of the four learning algorithms with multiclass classification. We have done the experiments on 18 datasets. Because of the space limitation, we could not show their results here. We just show the overall average in term

of accuracy of the 18 datasets in Figure 2. Figure 2 verifies the conclusions we made from the experimental results for binary datasets: naïve bayes is the most noise tolerant learning algorithm and it performs well when the noise level is the highest (50%). Although it performs better when the noise level reduces, the performance of the other three learning algorithms improves significantly. When the noise level is higher than 20%, support vector machine performs the best in average, followed by decision tree, followed by logistic regression. However, when the noise level is further reduced, decision tree performs the best, followed by support vector machine, followed by logistic regression.

6. CONCLUSION

In this research, we studied on how the quality of models is affected by different amounts of noise for different machine learning algorithms. The study was performed on four different classifiers called decision tree, naïve bayes, support vector machine, and logistic regression. A detailed experimentation proves that the behavior of each algorithm depends on the percentage of noise injected and the characteristics of different datasets.

This kind of a study is very useful in situations of real world data processing that may contain implicit and explicit errors. The results show that naïve bayes is the most noise tolerant algorithm. However, decision tree performs the best in average under different noise level for most datasets (binary and multiclass), followed by logistic regression and support vector machine. When we have approaches to improve the data quality (reduce the noise level), they are more preferred than naïve bayes.

7. REFERENCES

- [1] Brodley, C.E. and Friedl, M.A. Identifying and eliminating mislabeled training instances, *Proc. of 13th National Conf. on Artificial Intelligence*, 1996, pp.799-805.
- [2] Brodley, C.E. and Friedl, M.A. Identifying mislabeled training data, *Journal of Artificial Intelligence Research*, 1999, 11, 131-167.
- [3] John, G. H., and Langley, P. Estimating Continuous Distributions in Bayesian Classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, San Mateo, 338-345, 1995.
- [4] Kubica, J., and Moore, A. Probabilistic Noise Identification and Data Cleaning, pp.131-138, *Proceedings of the third IEEE International Conference on Data Mining (ICDM'03)*, 2003
- [5] Witten, I.H., and Frank, E. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. Morgan Kaufmann Publishing, 2005.
- [6] Xiong, H., Pandey, G., Steinbach, M. and Kumar, V. Enhancing data analysis with noise removal. *IEEE Transactions on Knowledge and Data Engineering*, 2006, 18, 304-319.

- [7] Zhu, X., Wu, X. and Chen, Q. Eliminating Class Noise in Large Datasets. *Proceedings of the 20th ICML International Conference on Machine Learning (ICML 2003)*. Washington D.C., 2003, pp. 920-927.
- [8] Zhu, X., Wu, X. and Chen, Q. Bridging Local and Global Data Cleansing: Identifying Class Noise in Large, Distributed Data Datasets. *Data Mining and Knowledge Discovery*, 2006, 12, 275-308.

IMPROVING NON-SMALL CELL LUNG CANCER

CLASSIFICATION IN DATA MINING COURSES*

Quoc-Nam Tran, Lamar University

ABSTRACT

Data mining courses have become popular electives in many graduate and undergraduate computer science curriculums. In these courses we usually give a comprehensive coverage of data mining concepts, algorithms, methodologies, management issues, and tools, which are illustrated through simple examples for an easy understanding and mastering of these materials. However, these courses often fail to provide students with opportunities to develop and evaluate applications in real-world environments. We fill these gaps by creating new materials in cancer research which would open opportunities to develop and evaluate applications in real-world environments for the senior-level and graduate students in computer science.

In this paper, we show how new data mining techniques can be developed and used to improve the classification of non-small cell lung carcinoma (NSCLC) cancer. The outcomes of this research are used as new materials for students in our data mining courses.

1 INTRODUCTION

Advanced technologies allow us to capture and store vast amount of data in many fields such as finance market and medical research. One of the grand challenges of the information age is to turn raw data into information and to turn information into knowledge by finding patterns, trends, and anomalies in the vast amount of data.

Data mining courses have been offered in many graduate and undergraduate computer science curriculums. However, in many undergraduate curriculums, especially at small schools, these courses usually present a comprehensive coverage of data mining concepts, algorithms, methodologies, management issues, and tools, which are illustrated through simple examples for an easy understanding and mastering of these materials, but

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

often fail to provide students with opportunities to develop and evaluate applications in real-world environments [7 ,12 ,13]. As a result, students are not as prepared as they should be for graduate study or careers in industry. We propose to fill the aforementioned gaps by developing new data mining algorithms for cancer research which open the opportunities to develop and evaluate applications in real-world environments for the senior-level and graduate students in computer science. Specifically, we show how to develop new data mining algorithms for finding genetic markers for non-small cell lung cancer and use these new genetic markers to improve the classification of the disease.

Non-small cell lung carcinoma (NSCLC) is the most common cause of worldwide cancer premature death with a very low survival rate of 8%-15%. Early stage patients can have up to four times the survival rate of 40%-55%. Hence, discovering biological markers that can be used to improve the diagnosis and prognosis of the disease is an important clinical challenge [8 ,9 ,10].

NSCLC is sub-categorized as adenocarcinomas, squamous cell carcinomas, and large-cell carcinomas, of which adenocarcinomas are the most common [11]. The histopathological sub-classification of lung adenocarcinoma is challenging. For example, in one study independent lung pathologists agreed on lung adenocarcinoma sub-classification in only 41% of cases [5]. In another study, proportional hazard models identified an optimal set of 50 prognostic mRNA transcripts using a 5-fold cross-validation procedure. This signature was tested in an independent set of 36 squamous cell lung carcinomas (SCC) samples and achieved 84% specificity and 41% sensitivity with an overall predictive accuracy of 68% [6]. Combining the SCC classifier with their adenocarcinoma prognostic signature gave a predictive accuracy of 71% in 72 NSCLC samples.

In this paper, we present a new data mining algorithm that finds genetic markers and uses the markers to predict with 99.5% accuracy whether a patient has NSCLC and the sub-type of cancer in case the patient has NSCLC. The algorithm discovers novel genetic changes that occur in lung tumors using gene-expression profiles. We discovered that a very small set of nine genes (JAG1, MET, CDH5, ABCC3, DSP, ABCD3, PECAM1, MAPRE2 and PDF5) from the dataset of 12,600 gene-expression profiles of NSCLC acts like an inference basis for NSCLC lung carcinoma and hence can be used as genetic markers. This very small and previously unknown set of biological markers gives an almost perfect predictive accuracy for the diagnosis of the disease. Clearly, finding the gene-expression profiles of these 9 genes for a patient will cost much less than finding the full-scale gene-expression profiles of 12,600 genes for the patient. Furthermore, it will help to speed-up the diagnosis of the disease, which is crucial for the survival of the patient.

Our algorithm overcomes many challenges which specifically arose from datasets of gene-expression profiles. The algorithm gives a significant improvement in accuracy of diagnostic prediction in comparison with a predictive accuracy of 68%-71% in other molecular classification methods for lung carcinomas.

2 A NEW DATA MINING ALGORITHM FOR CLASSIFICATION

Before presenting our new algorithm for finding genetic markers and predicting NSCLC lung cancer, we will address the challenges one has to overcome while working with gene-expression profile datasets. Basic information about Gini indexes and classification algorithms can be found in many data mining books [7 ,12 ,13].

Range/Class	C_1	C_2	C_3
R_1	4	6	30
R_2	6	30	4
R_3	0	4	16

Table 1: Bias due to the order of classes

The first challenge that arose from the gene-expression datasets is the bias due to the order of cancer types or classes in data mining terminology. We consider a simple example of a gene expression profile in Table 1 where the gene dataset D has d elements and three classes. The gene expression values were discretized into three ranges. Clearly, the cancer types or classes can be rearranged in any order. When this gene is evaluated by the current microarray analysis methodologies, for example by calculating the Gini

$$\text{index } gini_A(D) = \sum_{i=1}^m \frac{|R_i|}{d} gini(R_i) \quad , \text{ the first two rows contribute}$$

$$\text{Gini index because } gini(R_i) = 1 - \sum_{j=1}^n p_{i,j}^2 \quad p_{i,j} = \frac{|C_{i,j}|}{|R_i|} \quad \text{where}$$

frequency of class C_j in R_i . We have the same problem when entropy is calculated instead of the Gini index. That said, when one just considers the probability distribution without taking into account the order of the classes, the first two partitions will be considered the same. Clearly, the two partitions should not be considered the same because Partition R_1 says that 75% of patients with gene expression values within this range are classified into Class C_3 while Partition R_2 says that 75% of patients with gene expression values within this range are classified into Class C_2 . Hence, in order to have a robust gene selection method, one has to differentiate the partitions with different class orders because they have different amount of information.

To solve this problem, we generalized the well known Lorenz curves, a common measure in economics to gauge the inequalities in income and wealth. In Figure 1, we illustrate how modified Lorenz curves and modified Gini coefficients are calculated. The Equality Line (Eq) is defined based on the percentages of elements in $|C_1|$,

$$|C_{1,2}| = |C_1| + |C_2| \dots |C_{1,n}| = \sum_{i=1}^n |C_i| \quad , \quad \dots \quad \text{at } x \text{ coordinates } 0$$

n is the number of classes and $|C_1| \leq |C_2| \leq \dots, |C_n|$. The Lorenz curve of a partition, say R_j , is defined based on the percentage of elements in $|C_1^j|, |C_1^j| + |C_2^j|, \dots, \sum_{i=1}^n |C_i^j|$ at x coordinates $0, 1/n, 2/n, \dots, 1$. The Gini coefficient of a partition, say R_j , is defined as $(\int_0^1 L(R_j) dx - \int_0^1 Eq. dx) / \int_0^1 Eq. dx$. One can easily see that the partitions with different class orders are now differentiated.

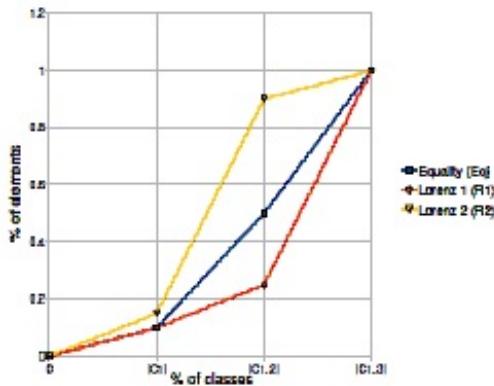


Figure 1: Lorenz curves

Another challenge for microarray analysis methodologies comes from the order of discretized gene expression values. We consider another simple example of two gene expression profiles in Table 2 with three classes. The gene expression values were discretized into four ranges. In contrast to the previous challenge, the ranges of gene-expression values do follow some order. When this genes are evaluated by the current microarray analysis methodologies, for example by calculating the Gini index of gene A using database

$$t^D gini_A(D) = \sum_{i=1}^m \frac{|R_i|}{d} \cdot gini(R_i)$$

where $d = |D|$, the two genes would have the same rank. Clearly, the gene expression profile on the right hand side of Table 2 must be ranked higher in comparison with the one on the left because the gene values maintain the order even after being discretized. That said, one has to take into account the gene expression profiles with different value orders because they have different amount of information.

Class/Range	C_1	C_2	C_3	Class/Range	C_1	C_2	C_3
R_1	3	0	0	R_1	3	0	0
R_2	0	100	0	R_2	4	0	0
R_3	4	0	0	R_3	0	100	0
R_4	0	0	5	R_4	0	0	5

Table 2: Bias due to the order of gene expression values

To solve this problem, we generalized the Gini coefficients by taking into account the splitting status and the Gini ratio. The splitting status of D with respect to the attribute

$$\text{is calculated as } \text{split}_A(D) = 1 - \sum_{i=1}^m \left(\frac{|R_i|}{d} \right)^2.$$

The Gini ratio of D with respect to the attribute A is defined as

$$\text{LorenzGini}(A) = \Delta gini(A) / \text{split}_A(D)$$

$$\Delta gini(A) = gini(D) - gini_A(D).$$

Furthermore, to take into account the gene expression profiles with different value orders, the Gini coefficient is calculated as $\text{gini}_A(D) = \sum_{i=1}^m \frac{|R_i|}{d} \cdot \delta(i) \cdot gini(R_i)$

where $\delta(i)$ is the sum of the normalized distances between the i th row and rows $i-1, i+1$.

The coefficient $\delta(i)$ is used as a weight to emphasize a row when it is close to its neighbors. Our new algorithm is as follows:

Input: A gene-expression dataset D with up to 34,000 dimensions.

Output: A small subset of genes as genetic markers and a prediction model for NSCLC lung cancer

Step1: Discretize the values.

Step2: Select genetic markers by using the genes with highest ranking LorenzGini.

Step3: Build the prediction model to classify patients using the genetic markers.

We have implemented this algorithm in Maple [4] and Weka [1], a popular system for data mining. A threshold can be used for controlling the number of significant genes for genetic markers. The splitting status of dataset D with respect to a gene A can be calculated as a by-product when the reduction in impurity of D with respect to the attribute A is calculated. Therefore, the time complexity and space complexity of the algorithm are the same as the complexities of Gini index algorithm.

Notice that our new method works for any dataset with 2 classes. For any number of classes, even when the number of classes is equal to 2, the new method is completely different with other microarray analysis methodologies.

3 EXPERIMENTATION

To test and validate our algorithm, we extract the gene-expression profiles of NSCLC from the mRNA expression profiles in [3] in that a total of 203 snap-frozen lung tumors ($n = 186$) and normal lung ($n = 17$) specimens were used to create the datasets. Of these, 125 adenocarcinoma samples were associated with clinical data and with histological slides from adjacent sections. The 203 specimens include histologically defined lung adenocarcinomas ($n = 139$), squamous cell lung carcinomas ($n = 21$), pulmonary carcinoids ($n = 20$), and normal lung ($n = 17$) specimens. Total RNA extracted from samples was used to generate cRNA target, subsequently hybridized to human U95A oligonucleotide probe arrays according to standard protocols.

As the result, we obtained a dataset of 12,600 gene-expression profiles for 197 patients. We discretized the continuous values of gene expressions using the method described in the previous section. It is worth to mention that getting the 12,600 gene-expression profiles for each patient is very costly and time consuming.

To test the accuracy of classification models, we use k -fold cross validation, which is a common method for estimating the error of a model, on some benchmark medical data sets [2]. The reason for using this testing approach is that when a model is built from training data, the error on the training data is a rather optimistic estimate of the error rates the model will achieve on unseen data. The aim of building a model is usually to apply the model to new, unseen data – we expect the model to generalize to data other than the training data on which it was built. Another reason for using this testing approach is that the available medical data sets are small and no test data set is available. It is well-known that k -fold cross-validation is very useful for this type of data sets.

For a reliable evaluation of the accuracy, we test the classification algorithm for many values of k . For each value of k , the data set D is randomly divided into k subsets D_1, D_2, \dots, D_k . We leave out one of the subsets $D_i, i=1..k$ each time for being used as a test data set for cross validation. The remaining subset $\bigcup_{j \neq i} D_j$ is used to build the model. The cross validation accuracy computed for each of the k test samples are then averaged to give the k -fold estimate of the cross validation accuracy. To ease the effects of the random partitions on the data set, this whole process is repeated 10 times and the results are then averaged again to give the estimated accuracy of the comparing algorithms.

3.1 Finding genetic markers for classification

We use our new algorithm described above, which was implemented in Maple in Weka to select 250 genes with the highest LorenzGini indexes. To further reduce the size of the gene subsets and to improve the prediction accuracy, we evaluate different combinations of genes to identify an optimal subset in terms of accuracy for the Bayesian Net classification. The gene subsets to be evaluated are generated using different subset search techniques. We use Best First and Greedy search methods in the forward and backward directions. Greedy search considers changes local to the current subset through the addition or removal of genes. For a given parent set, a greedy search examines all possible child subsets through either the addition or removal of genes. The child subset that shows the highest goodness measure then replaces the parent subset, and the process is repeated. The process terminates when no more improvement can be made. Best First search is similar to greedy search in that it creates new subsets based on the addition or removal of genes to the current subset with the ability to backtrack along the subset selection path to explore different possibilities when the current path no longer shows improvement. To prevent the search from backtracking through all possibilities in the gene space, a limit is placed on the number of non-improving subsets that are considered. In our evaluation we chose a limit of five.

The algorithm returns a set of nine genes (JAG1, MET, CDH5, ABCC3, DSP, ABCD3, PECAM1, MAPRE2 and PDF5) from the dataset of 12,600 gene-expression

profiles of NSCLC. We exploit this small set of genes to differentiate all sub-types of NSCLC lung cancer. Next, our algorithm uses the information from these 9 gene-expression profiles to predict with 99.5% accuracy whether a patient has NSCLC and the sub-type of cancer in case the patient has NSCLC. During the testing and validation process, all patients with lung adenocarcinomas were correctly predicted, all patients except one with squamous cell lung carcinomas were correctly predicted, all patients with pulmonary carcinoids were correctly predicted, and all patients with normal lung specimens were correctly predicted. The only false prediction was a patient with squamous cell lung carcinomas but incorrectly predicted as adenocarcinomas. As we can see, this very small set of genes gives an almost perfect predictive accuracy for the diagnosis of the disease. When the number of genes is further reduced or increased, the accuracy starts to decline. That said, this set of nine genes acts like an inference basis for NSCLC lung carcinoma and hence can be used as genetic markers.

3.2 Comparing with other gene selection methods

We now investigate the classifying accuracy of the significant genes generated by LorenzGini with respect to the size of the reduced microarray datasets. We compare with SAM for our experiment work. SAM combines t-test and permutations to calculate a False Discovery Rate to provide a subset of genes that are considered significant. We select four sets of 50, 100, 150, 200 and 250 most significant genes by using the parameter values of 0.556, 0.458, 0.4188, 0.383 and 0.3568, respectively.

We then use the Bayesian Net classification in Weka to check the accuracy of the most significant gene sets generated by LorenzGini and SAM [1]. Besides our fresh implementation of LorenzGini algorithms, simple converters were written to connect SAM and Weka. For a reliable evaluation of the accuracy, we test the classification algorithm for many values of k as specified in our validation plan.

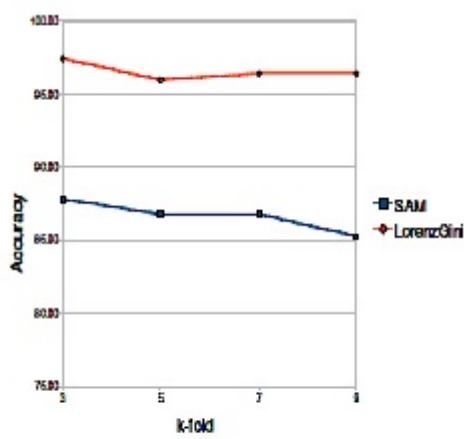


Figure 2: SAM's & LorenzGini's accuracy

accuracy. This observation is also true for other classification methods.

Figure 2 shows the accuracy of the gene expression profile classification using Bayesian Net algorithm on SAM's gene sets and on LorenzGini's gene sets with 50 genes. As we can see, the classifying accuracy has been improved with the LorenzGini's gene selections. We also observed that the accuracy of the gene expression profile classification using Bayesian Net algorithm on SAM's gene sets declined when the number of genes is reduced to 50 and below. In contrast, the accuracy of the gene expression profile classification using LorenzGini's gene sets is stable even when the number of genes is reduced to 9, which has the highest

4 CONCLUSION

We presented new methods and data mining algorithms that can find very small sets of 9 biological markers and give an almost perfect predictive accuracy for the diagnosis of NSCLC lung cancers. Finding the gene-expression profiles of these 9 genes for a patient will cost much less in comparison with finding the full-scale gene-expression profiles of 12,600 genes for the patient. Furthermore, this small set of biological markers will help to speed-up the diagnosis of the disease, which is crucial for the survival of the patient. These new methods and algorithms provide opportunities for students in data mining courses to work on other real-world problems by applying the same ideas on other type of diseases.

REFERENCES

- [1] <http://www.cs.waikato.ac.nz/ml/weka>, 2009.
- [2] ASUNCION, A., AND NEWMAN, D. UCI machine learning repository, 2007.
- [3] BHATTACHARJEE, A., RICHARDS, W. G., STAUNTON, J., LI, C., MONTI, S., GOLUB, T. R., SUGARBAKER, D. J., AND MEYERSON, M. Classification of human lung carcinomas by mRNA expression profiling reveals distinct adenocarcinoma subclasses. *Proc. Natl. Acad. Sci. USA* 98, 24 (2001), 13790-13795.
- [4] CHAR, B. W., GEDDES, K. O., GONNET, G. H., LEONG, B. L., MONAGAN, M. B., AND WATT, S. M. *Maple V Language Reference Manual*. Springer Verlag, 1991.
- [5] COLLINS, F. S., MORGAN, M., AND PATRINOS, A. The human genome project: lessons from large-scale biology. *Science* 300 (2003), 286-290.
- [6] COX, B., KISLINGER, T., AND A., E. Integrating gene and protein expression data: pattern analysis and profile mining. *Methods* 35, 3 (2005), 303-314.
- [7] HAN, J., AND MICHELIN, K. *Data Mining: Concepts and Techniques*, 2 ed. Morgan Kaufmann, 2006.
- [8] JEMAL, A., SIEGEL, R., WARD, E., HAO, Y., XU, J., AND THUN, M. J. Cancer statistics, 2009. *CA Cancer J Clin* 59 (2009), 225-249.
- [9] JEMAL, A., SIEGEL, R., WARD, E., MURRAY, T., XU, J., AND THUN, M. J. Cancer statistics, 2007. *CA Cancer J Clin* 57 (2007), 43-66.
- [10] SINGHAL, S., MILLER, D., RAMALINGAM, S., AND SUN, S.-Y. Gene expression profiling of non-small cell lung cancer. *Lung cancer* 60, 3 (2008), 313-324.
- [11] WATSON, J. D. The human genome project: past, present, and future. *Science* 248 (1990), 44-49.
- [12] WITTEN, I. H., AND FRANK, E. *Data Mining: Practical machine learning tools and techniques*, 2nd ed. Morgan Kaufmann, 2008.

- [13] YE, N., Ed. *The Handbook of Data Mining*. Lawrence Erlbaum Associates, 2003.

PROGRAMMING CONTEST HOSTING*

CONFERENCE TUTORIAL

Steve Baber

Harding University, Searcy, AR

David Hoelzeman

Arkansas Tech University, Russellville, AR

Becky Cunningham

Arkansas Tech University, Russellville, AR

Rick Massengale

University of Arkansas Fort Smith, AR

This tutorial is based on the experiences of the presenters' years of experience in hosting and/or participating in the ACM ICPC and CCSC-Midsouth programming contests. The presentation covers all aspects of contest hosting including hospitality, problem development, hardware and network, programming related software, and the Programming Contest Control system (PC2) developed by California State University, Sacramento (<http://pc2.ecs.csus.edu/>).

Hospitality

For this portion, the presenter will discuss issues related to organizing food and drink, decorations, and general environment in order to create a comfortable contest environment. Issues with prizes and plaques will be discussed.

Problem Set Development

For this portion, the presenter will discuss issues related to collecting programming problems in order to create a contest problem set. Discussion will also concentrate on what makes a good problem for a programming contest.

Hardware

The presenter will discuss what hardware is needed to successfully host a contest and the hardware needs of the team computers and judging computers. Discussion will center on the ideal where all contest related hardware is equal and when it would be appropriate to have differing hardware.

Network

The presenter will discuss the network needs of a contest that utilizes an electronically based submission utility (i.e. PC2). Hard-wired and wireless networks will be discussed.

* Copyright is held by the author/owner.

Software

The presenter will discuss software that can be installed to support the contest operations. This includes text editors, IDEs, compilers, etc.

Network Submission Software

The intent of this section is to provide a hands-on experience in configuring (PC2). At the end of the tutorial the attendee should have implemented a simulated contest site on a single computer. Time permitting, a discussion or implementation of a multi-site setup will be made (depending on available resources).

Presenter Backgrounds

Steve Baber, Harding University, Searcy, AR, baber@harding.edu

Steve Baber has worked with programming contests since the Fall 1983 ACM South-Central Regional contest. He has coached at least one team every year since. Dr. Baber has authored many contest problems, has served as a judge for several ACM regional contests, and has filled the role as one of the contest directors for the CCSC Midsouth contest since 2005.

Becky Cunningham, Arkansas Tech University, Russellville, AR, rcunningham@atu.edu

Becky Cunningham has been an assistant site director for five ACM regional contests since 2002. Her responsibilities have included the coordination of meals, snacks, and drinks; budget tracking; decorations; name tag and certificate printing; and other miscellaneous items that foster a comfortable contest environment.

David Hoelzeman, Arkansas Tech University, Russellville, AR, dhoelzeman@atu.edu

David Hoelzeman first became involved in programming contests as a team coach to teams participating in the South Central regional competition of the 1999 ACM ICPC. He has continued to coach teams for the Mid Central regional competition of the ACM ICPC as well as for competitions held by Acxiom Corporation. He has been site director for five competitions of the Mid Central ACM ICPC since 2002 and has assisted with, or been one of the contest directors for, the CCSC Midsouth contest since 2008. The PC2 software has been utilized in all but one of these competitions.

Rick Massengale, University of Arkansas Fort Smith, AR, rmasseng@uafsmith.edu

Rick Massengale first became involved in programming contest as a participant, on the Arkansas Tech University Center team in 1992-93, where he was a member of a team that made it to the 1993 ACM Scholastic Programming Contest Finals. His next involvement came in 2005 and 2006 where he served as the Technology Director for the ACM ICPC programming contest. He facilitated networking the contest floor area and

judging center, allowing the use of the PC2 software in these competitions. He has since started the Programming Team program at UA Fort Smith where he has served as a sponsor and coach since 2007 for teams competing in the ACM ICPC, CCSC Midsouth conference and Acxiom Corporation contest.

CHALLENGES AND PROFESSIONAL TOOLS USED WHEN TEACHING WEB PROGRAMMING*

*Yi Liu and Gita Phelps
Information Technology Department
Georgia College & State University
Milledgeville, GA, 31061
478 445-3344
yi.liu@gcsu.edu, gita.phelps@gcsu.edu*

ABSTRACT

Web programming can be taught in many different ways with different topics and tools covered. This paper summarizes the challenges, describes our course arrangement, and discusses the professional tools used in a web programming class. Our web programming course is offered as an elective to junior and senior CS students.

INTRODUCTION

With the fast development of Internet, most software applications have been oriented toward web-based applications. Web programming is one of the fastest growing fields in the computing job market. Many schools are aware of the importance of web software development, and provide web programming courses for students. However, there is no consensus about how to incorporate the course into the computer science curriculum [1]. The content and the level that the course is offered widely varies between universities and instructors. This paper describes a web programming course offered to junior and senior CS students in our school. The contribution of this paper is twofold: 1) briefly summarize the challenges in our web programming course design, so other educators may be aware of, or handle these difficulties in their early course design phase; 2) introduce and share the experiences on two professional tools, Eclipse and Navicat Lite. Eclipse has been widely introduced and used in Java desktop applications, but it has not been introduced and discussed in web programming course to our knowledge. The main reasons include: the complexity of integrating Eclipse with web server and database

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

server; unexpected long learning-time required for the students; and the rapidly updated status of the software itself.

CHALLENGES

Teaching web programming has many challenges. Selecting the topics and providing the sufficient depth is the first challenge an instructor faces. A web application is a complex, multi-faceted execution model, so focusing a single technology and performing a depth oriented approach does not work for web programming [2]. A course may include topics in web creating tools, client side programming and server side programming, each containing many options. For example, there are a variety of web creation tools that allow users to draw web pages and minimize coding during a web application development process. An instructor may introduce tools such as Adobe Flash, Microsoft Silverlight, Adobe Flex, Dreamweaver and Java Server Faces. Client side programming adds dynamic behavior to the client by executing a program in the browser and may incorporate XHTML, CSS, JavaScript, DOM, XML, and Ajax. Server side programming involves writing scripts that reside on the host computer; examples of server-side topics include, C#, PHP, JSP/Java Servlet, ASP.NET, Ruby, Perl and Python. Other topics typical in a web programming course include online database management, usability, web services, privacy, and security issues.

The second challenge is that web technologies change rapidly as new technologies and standards emerge. An instructor usually debates whether he/she should adopt the new technologies and standards. Adopting the latest technologies and following the new standards may require the instructor to address some difficult issues in class, such as backward compatibility, learning new software, and deploying projects in browsers which may not evolve with the development of the new technologies and standards.

The third challenge is inadequate integration among current web technologies and inconsistent implementation of standards [3]. Inconsistency between web browsers adds to the complexity of development, and these difficulties are reflected in the curriculum of web programming. Stepp, Miller, and Kirst [1] constrained students to use Firefox and ignore problems arising on Internet Explorer and other browsers. We found that some tools did not adequately explain errors. For example, web browsers produced no output for most errors. Most of the textbooks recommended Microsoft Script Debugger and Firebug Plugin; however, Microsoft Script Debugger reports the line number the error occurs in the browser's own internal rendering of the document, which may not match the source file and the Firebug Plugin does not report the element that causes the error. As a result, students were often frustrated and complained about spending an unusually long time on coding and debugging.

After reviewing the above challenges and difficulties, we decided to focus on the three objectives: 1) understand the foundations of web application designs; 2) be able to integrate the technologies in developing multitier web applications; 3) give practical experiences in creating web applications. Our course focused on widely supported, frequently used web technologies, not the latest technologies. In order to facilitate the learning outcomes of the second and third objectives, we introduced two free professional coding tools: Eclipse for Java EE [11] and Navicat Lite [10].

RELATED WORK

Stepp, Miller, and Kirst [1] implemented a lower-level web programming course for major and non-major students. They claimed a course without a lot of prerequisites, offered at a lower level would attract a large number of students and could potentially increase enrollment in subsequent computer science courses. Robert E. Noonan [4] developed a senior-level web programming course which emphasized server-side programming, database interaction, and security. Xusheng Wang [5] also discussed a web programming course focusing on server-side techniques with PHP and MySQL. M. J. Lantis [6] introduced a web editor as a development platform for teaching HTML and client-side programming in the internet 101 course. He claimed that using a web editor was efficient for mundane tasks.

Our course is different in that we offer the course at junior-level. Dealing with only CS majors gives us the opportunity to cover technologies at a deeper level. We introduce the technologies on the three tiers (presentation tier, logic tier, and storage tier) of web application designs, and focus on the integrations among the three tiers. Finally, we require students to complete the assignments and the group project without using any web creation tools.

COURSE ARRANGEMENT

The first objective of the course is to understand the foundations of web application designs. A web application usually includes three tiers - presentation, logic and storage. The *presentation* tier focuses on outputting the results in a web browser. The *logic* tier processes data, and performs logical decisions by using web content technology. The *storage* tier performs database functionalities. The course is organized based on these three tiers. The textbook written by Robert w. Sebesta [7] is a good fit for our course; however, we supplement it with an e-book[8] that provides a more detailed explanation of technologies on the logic tier. Furthermore, we believe that students who write code for web pages understand better about the foundation of web design than those who use web creation tools that draw the web pages. Therefore, none of the web page "drawing" tools are taught or allowed on course assignments.

Table 1 shows the contents covered in each tier. In order to help junior students understand the presentation tier with enough in-depth knowledge of each topic, only three topics, XHTML, CSS, and JavaScript, and the integration among these components are taught.

JSP and Java Servlet are taught on the logic tier. Java is still the most popular programming language [9]. All the Java-related tools are free to students, and it is the primary programming language used by our CS students. Using Java on this tier allows our students to concentrate on interaction and integration with the other two tiers as they develop complicated applications.

The storage tier maintains a database management system. MySQL, a fast, reliable and easy to use database, is introduced during the discussion of the storage tier. We also surveyed two free and popular server-side scripting languages, PHP and Ruby. We

provide many examples to demonstrate how these two languages integrate with other technologies.

Tier	Topics covered	Period
Presentation tier	XHTML, CSS, JavaScript, and integration among them,	4 weeks
Logic tier	JSP, Servlet, HTTP, cookies, and Model View Controller	4 weeks
Storage tier	MySQL, Integration with Servlet, PHP, and Ruby	6 weeks

Table 1: Topics covered in each tier

TOOLS

The second and third objectives of the course are to integrate the web technologies and give students practical experiences on creating web applications. In-class labs, web page assignments, and a real-world web project incorporating all tiers of web development were assigned during a semester. Students learned many web technologies quickly and sometimes had trouble when they integrated the technologies for their assignments. Some errors they got were painfully difficult to locate and fix. In our class, we did not allow the students to use web creation tools; however, the students could use the professional IDE tool, Eclipse for Java EE.

On the presentation tier, using HTML, web page, and CSS editors provided by Eclipse can significantly reduce the number of syntax errors. The JavaScript editor can indicate incorrect syntax in the time of writing code. In general, each editor not only supports the basic functionalities for a specific type of a file, such as wizard for creating the file, web page preview, and validation, but it also has real-time syntax error checking, tag palette, and content assist.

On the logic tier, Eclipse can greatly simplify the deployment of a dynamic web project by automatically configuring the running parameters for a project, and moving output class files between the project output folders and the deployment folder on Tomcat. Eclipse also provides a runtime environment in which a student can debug projects.

On the storage tier, after comparing with many Mysql admin tools, students are advised to use the free Navicat Lite version [10]. Our school's policy requires a server to use SSH and SSL for off-campus connections. The simple and intuitive GUIs of building SSH and SSL tunnels, and managing databases are the main reasons to choose Navicat Lite.

EXPERIENCES WITH TOOLS

We introduced Eclipse at the beginning of the semester. The features available in Eclipse for the presentation tier were easy to learn and use. The students liked the

integrated development environment, and claimed that it was much more convenient and intuitive than using other individual tools, such as text editor, browser, and debugging tools. However, the students experienced difficulties when their web projects moved to the logic and storage tiers. First, the students had to integrate Tomcat and MySql servers within Eclipse. The default run configuration provided by Eclipse was for desktop applications. Some major steps are listed below:

1. When adding a Tomcat server within Eclipse, the location and the deployment of Tomcat server have to be changed manually through the tool "properties" of the server.
2. Manually change the output path of a project to "\WEB_INF\classes" folder in order to run the project on a Tomcat server within Eclipse.
3. Copy JDBC.jar and Servlet.jar files in the library folder.
4. Manually edit web.xml file to ensure that all servlets have the corresponding servlet mapping in the file for all servlet in the project.

Second, some functionalities of Eclipse do not work as we expected. For example, Eclipse is supposed to automatically build a project whenever the source code changes. However, we found that it did not do so for a dynamic web project sometimes, and the related reason was unknown. The students had to force Eclipse to build the project by cleaning and building it manually.

The students experienced unexpected difficulties and long learning time when using Eclipse on the logic tier. Most students stopped using Eclipse when the course was taught for the first time. When the course was taught for the second time, four classes, including one lecture, one lab demo, and two lab exercises, were provided for guiding the students on using Eclipse with the Tomcat server. Detailed step by step instructions were given to students, and many examples of deploying dynamic web projects within Eclipse were shown. Although the students still initially reported that Eclipse was complex after the classes were done, they later claimed that using Eclipse significantly reduced the time spent on coding and deploying a project, and made web programming relatively fun and easy. The average debugging and deploying time was shortened about 30% after comparing the feedbacks received during two semesters, one without Eclipse and the other with Eclipse. At the end of the semester without Eclipse, one out of four groups could not finish the project and another group had many minor errors in their project. During the semester with required Eclipse, all four groups successfully completed their projects. We conclude that it is beneficial to use Eclipse in the web programming course.

The students reported no difficulty using Navicat Lite. In fact, all of its interfaces are easy to learn, and well designed. However, they mentioned that the interface obtained from SSH tab should tell the user explicitly that the address, username, and password were expected from the Tomcat server rather than from the MySql server.

CONCLUSION

We briefly summarized the challenges in teaching web programming courses. These challenges include selecting which topics and how to cover topics in a suitable depth level, how to handle the rapidly evolving technologies, and what unique difficulties are faced when programming web applications. We presented the outline of our course and

highlighted two professional tools, Eclipse and Navicate Lite, used in the course. Although Eclipse is widely used in Java desktop applications, it is not introduced and discussed in web programming courses to our knowledge. We shared our experiences on using Eclipse and Navicat Lite. After comparing the feedbacks received from students and the qualities of group projects from two semesters, we believe that using these professional tools, especially Eclipse, can ease some of the pains on learning web programming, and improve the success rate of the course projects.

REFERENCES

- [1] Stepp, M., Miller, J., and Kirst, V. *A "CS 1.5" introduction to web programming*, Proceedings of the 40th ACM technical symposium on Computer science education. pp. 121-125, 2009
- [2] Verbyla, J., Roberts, G., Web technology as curriculum, *Proceedings of the 3rd Australasian conference on Computer science education*. 1998.
- [3] Rode, J., Nonprogrammer web application development. *Conference on Human Factors in Computing Systems*. pp. 1055 - 1056, 2005.
- [4] Noonan, R. E., A course in web programming, *Consortium for Computing Sciences in Colleges, Journal of Computing Sciences in Colleges*, Vol. 22, pp. 23-28, 2007.
- [5] Wang, X., A practical way to teach web programming in computer science, *Consortium for Computing Sciences in Colleges, Journal of Computing Sciences in Colleges*, Vol. 22, pp. 211 - 220, 2006.
- [6] Lantis, M. J., Using a web editor as a development platform for teaching HTML and client-side programming in the internet 101 course: nifty tools and assignments, *Journal of Computing Sciences in Colleges*, p. 97, 2008.
- [7] Sebesta, R. W., *Programming the World Wide Web*, Addison Wesley, 2009.
- [8] Brown, M., Hall, L., *Core Servlets and JavaServer Pages*. [Online] 2010. <http://pdf.coreservlets.com/>.
- [9] Tiobe. TIOBE Programming Community Index for March 2010. <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. [Online] 2009.
- [10] Navicat. Database Tool: Navicat Lite. <http://www.navicat.com/en/download/download.html>. [Online] 2010
- [11] Eclipse. <http://www.eclipse.org>, [Online] 2010.

ROOTKIT ATTACKS AND PROTECTION

- A CASE STUDY OF TEACHING NETWORK SECURITY*

*Thomas Arnold and T. Andrew Yang
Computer Science Department
University of Houston - Clear Lake
Houston, TX 77058
yang@uhcl.edu*

ABSTRACT

A rootkit is a type of malware designed to gain privileged access to a computer while hiding itself from the user and the operating system by, for example, compromising the communication channels within the OS. A rootkit can hide files, data, processes, and network ports, and can typically survive a system restart. This stealthy design enables the rootkit to escape detection by most anti-malware tools, which may be effective in detecting/removing malware like viruses but are generally ineffective against rootkits. To answer this challenge, special anti-rootkit software applications have been developed. In this paper, we first review the status quo of Windows-based rootkits and anti-rootkit software. We then discuss how rootkits may be incorporated into classes to teach computer and network security concepts.

1. INTRODUCTION

The Windows operating system architecture is designed as a series of components and layers that allow it to execute on a wide variety of hardware platforms. These layers and components utilize different types of communication channels in order to function, including system call tables and I/O Request Packets (IRPs), among many others [1]. This architecture provides a high level of portability and extensibility, but at the same time creates opportunities for attackers to compromise the system. If one of the communication paths between the layers or components is intercepted by a malicious process, the attacker can perform stealthy activities like keystroke logging, spam email distribution, or Denial

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

of Service (DoS) attacks. Rootkits focus on these communication paths and interfaces to perform these types of malicious activities and conceal their presence on the system at the same time.

Additionally, the Windows OS uses a two-ring memory protection architecture to distinguish between kernel-mode and user-mode operations [1]. One of the disadvantages to such an approach is that the device drivers operate in the same "ring" as the OS kernel, even if they are implemented by a third party. This provides an opportunity for rootkit authors to implement their malicious software as a device driver and obtain kernel-mode privileges, essentially giving them complete control over the system.

While general-purpose anti-malware software may be effective in protecting the system against many forms of malware, they are generally ineffective in detecting and removing rootkits. Special purpose rootkit detection and removal software applications have been developed to answer this unique challenge. To provide comprehensive protection against such threats, computer security professionals should be aware of the potential impact rootkits may generate upon a computer system or a network. When teaching computer and network security concepts, the instructor should signify the stealthy design of rootkits and the limitations of general-purpose anti-malware software such as virus scanners. It is therefore desirable to construct a case study of rootkits, which may be used to deliver those important concepts in computer science and related classes.

In the rest of the paper, we first discuss the various designs of Windows-based rootkits, which is followed by a review of rootkit detection techniques. Section 4 discusses a sample project of using rootkits in teaching network security concepts.

2. CURRENT WINDOWS-BASED ROOTKIT TECHNIQUES

This section provides an overview of several rootkit design techniques that are currently in use against Windows-based systems. The design and detection of rootkits can best be described as an "arms race", with the rootkit authors and the security community engaging in a constant process of one-upmanship. The initial rootkits focused on UNIX-based systems, and used primitive designs that replaced system files with malicious versions, which were easily detected by file system scanners. Over time, the rootkit techniques have evolved into modifying undocumented OS data structures or becoming extremely hardware-dependent and relying on virtualization techniques.

- **System Hooking:** System hooking is a technique used by rootkits to redirect system call pointers to malicious programs that reside in memory [2]. The result is that the rootkit software can intercept the flow of execution and filter the results that are returned to the calling programs, hiding itself in the process. Typical locations for hooking in the Windows OS include Import Address Tables (IATs), System Service Dispatch Tables (SSDTs), or the Interrupt Descriptor Table (IDT). Detection of hooking is difficult for signature-based anti-malware applications or file integrity scanners because the system files on disk are not altered. In order to counter this technique, memory scanners such as *Rootkit Unhooker* [3] and *GMER* [4] were developed.

- **Direct Kernel Object Manipulation (DKOM):** Another commonly used technique that rootkits implement is known as Direct Kernel Object Manipulation (DKOM), where kernel data structures are manipulated to hide processes, change privileges, etc [2]. One of the earliest known rootkits to perform DKOM was the FU rootkit, which modified the EPROCESS doubly linked list in Windows to "hide" the rootkit processes. This technique took advantage of the fact that there are two separate lists for processes and threads. The ETHREAD list is used by the CPU scheduler while the EPROCESS list is used primarily for bookkeeping purposes [5]. By modifying the links in the EPROCESS list (and leaving the ETHREAD list alone), the rootkit was able to hide itself and still allow the malicious threads to continue being executed by the CPU. DKOM requires a lot of reverse engineering and a detailed knowledge of Windows internals, and can be very challenging to detect due to many undocumented features and the proprietary nature of the Windows OS.
- **System Routine Patching:** In this technique, the rootkit author modifies the source code of a system routine to cause the execution path to jump to malicious code which is resident either in memory or on disk. Some of the early UNIX-based rootkits completely replaced the system file with a modified version using the same name [2]. Modern Windows-based rootkits may embed a JMP instruction within the system binary to redirect the execution path [5]. This can be performed against the system binaries stored in the OS file system, or even in real-time against code loaded in memory. If the modification was performed on the file system, this can be easily detected by file integrity monitoring systems. Run-time modification can be detected by applications such as Kernel Path Protection, which is provided by the 64-bit versions of Windows.
- **Filter Drivers:** The Windows driver architecture was designed in a layered manner, so that third party hardware manufacturers can insert their drivers within already existing layers and utilize existing functionality provided by the Windows OS [2]. This feature also creates yet another opportunity for rootkit authors to inject their malicious code to interrupt the flow of I/O Request Packets and perform activities such as keystroke logging or filtering the results that are returned to anti-malware applications.
- **Hardware and Virtualization-based designs:** There are some recent proof of concept rootkits that operate independently of the OS, using hardware virtualization and chipset exploits to operate in the BIOS or PCI expansion cards [2]. At this time, the hardware-specific rootkits are not very prevalent in the wild, and techniques to detect these types of rootkits are likewise very sparse [2].

3. ROOTKIT DETECTION TECHNIQUES

There exist a number of different methods to detect rootkits, including signature-based detection, file integrity monitoring, cross-view analysis, hooking detection, and network-based detection. Most of the rootkit detectors employ several of these techniques, in order to provide the widest range of capability and increase their chances of success. In this section, we briefly describe each of these techniques, and include some examples of current software that uses them. One caveat for all of the

techniques described in this section is that a kernel mode rootkit can always alter the results reported to the anti-rootkit (ARK) software, and the lack of a reported detection is not always indicative of a clean system. However, in practice, many rootkit authors do not always include anti-detection or anti-forensics code in the malware, due to time and effort required to thoroughly address all the potential detection methods [6].

1) Signature-based Detection: The most common method for attempting detection of rootkits (and malware in general) is the signature-based technique [7]. Once a sample of malware has been obtained, the byte pattern of that software is heavily analyzed to identify a unique fingerprint that will distinguish this specific malware from legitimate software, as well as other types of malicious software. The fingerprint "signature" will then be integrated into a database that can be used by detection software when performing system scanning. If a scanned piece of software has a pattern that matches an entry in the malware database, it is extremely likely that it is malicious and should be flagged to the system and the user. While this technique has been successfully used for over twenty years, the main weakness is that it cannot detect new types of malware, until a sample can be analyzed to extract a signature. Popular programs that include signature-based detection include the *Microsoft Malicious Software Removal Tool* [8], *Kaspersky Internet Security* [9], and many others.

2) File Integrity Monitoring: File integrity monitoring is a detection technique first employed on UNIX systems by Tripwire in the early 1990s [7]. The method calculates cryptographic hashes for critical, unchanging operating system files and compares them to known values that are stored in a database. Typically this database is generated against a clean version of the operating system, so when a mismatch is detected, a file has been altered (likely by malicious software). This technique worked well against rootkits which patched system binaries on disk, but rootkit authors quickly adapted to use hooking or DKOM techniques instead. As a result, file integrity monitoring is not widely employed as a method of detection for modern anti-rootkit detectors.

3) Rootkit Hooking Detection: Detection of rootkit hooking is a straightforward process. The SSDT, IAT, and IDT each has a set of function pointers for each service or interrupt, which are all within a specific range in memory. When the rootkit modifies a hook to point to a malicious service or interrupt routine, the memory location almost invariably is located outside this specific range of the "clean" system, and is easily detected by anti-rootkit software [5]. Inline function and I/O Request Packet hooking is detected in essentially the same manner. While hooking is easily detected, it should be noted that a kernel-mode rootkit can alter the results of the detection software and make it appear that everything is nominal. Hooking detection is thoroughly provided in tools such as *Rootkit Unhooker* [3] and *GMER* [4], as well as several others.

4) Cross-View Analysis: Cross-view analysis involves looking at the system from the "user" or "API" view, and comparing it to the low-level hardware view [5]. The idea is that a rootkit will not be able to hide itself when the raw hardware is scanned. If a particular file or registry key is absent from the API view but is present in the hardware view, it is likely that a rootkit is attempting to hide itself from the system. This technique was first used in SysInternals' *RootkitRevealer* [10] software, and is now used in many other detectors. Rootkit techniques such as DKOM and system routine patching can be detected using the cross-view method.

5) Network-based Detection: In theory, rootkits may attempt to hide their network communication in an effort to evade detection from the OS or user. However, there exist a number of tools and methods to identify the existence of hidden network traffic by analyzing the communication and open ports from an external perspective and comparing it against the internal system view [11]. However, if the rootkit utilizes covert channels to hide the network communication, these techniques and tools will not be as effective of a detection method.

4. TEACHING NETWORK SECURITY USING ROOTKITS

The most effective way of raising someone's awareness of a certain threat is to give that person first-hand experience of that threat. A person who has experienced the damaging power of a hurricane, for example, will not likely to forget its devastating effects. Furthermore, hands-on labs and projects are important to achieve effective *experiential learning* [12]. Our approach is therefore to develop hands-on projects for students to gain first-hand experience of a rootkit attack, in an insulated environment. The procedure outlined below provides a proposal of how the project could be designed.

- **Project objectives:** The student will
 - (a) gain hands-on experience of setting up a computer system, introducing rootkit attacks against that system, and running available anti-rootkit software to clean up the system;
 - (b) become familiar with rootkit related concepts and tool sets;
 - (c) develop awareness of system vulnerabilities that may be exploited by malware.
- **Tools utilized:** portable disk drives, Windows OS, disk formatting tools, rootkit software (e.g., the *Rustock* rootkit), debugging tools (e.g., *TCPView* [13] or Windows Kernel Debugger [14]), anti-rootkit tools (e.g., Rootkit Unhooker [3], or GMER [4]).
- **Procedure:**
 - 1) To carry out the projects, each student is given a portable disk drive, and asked to have the Windows OS installed on it. To verify that a clean system is used, a disk formatter such as *Derek's Boot and Nuke (DBAN)* [15] can be used to zero-out all sectors on the hard drive before installing the OS.
 - 2) The rootkit is installed into the disk drive. In order to verify that the rootkit is installed, a kernel mode debugging session is performed to verify the existence of system hooks or other symptoms of rootkit infection. Tools such as *TCPView* [13] may be used to uncover processes that are invoked by rootkits. In Figure 1, for example, the highlighted *svchost.exe* process opens a TCP connection to a host in Russia known to be associated with the *Rustock* rootkit.
 - 3) Once the portable drive is verified to be infected with the rootkit, the anti-rootkit software is installed and scans are performed on the infected drive. If the rootkit is detected, and the anti-rootkit software provides an option to remove the rootkit, removal is attempted and confirmed by subsequent scans of corroborating tools, and performing another kernel mode debugging session to verify the absence of the symptoms observed after infection. Figure 2 is a report

generated by the *Rookit Unhooker* [3] software, detecting hooks installed by a *Rustock* rootkit in a system.

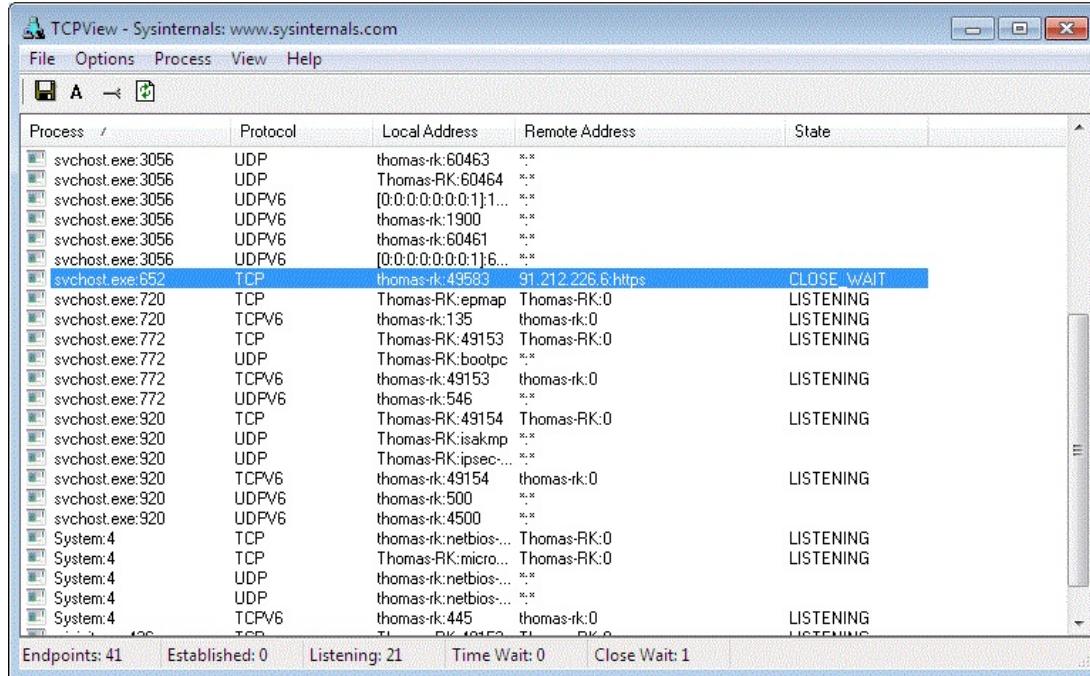


Figure 1. Sample TCPView snapshot showing rootkit infection

5. SUMMARY

In the paper, we have reviewed the status quo of rootkits, their various design techniques, the available anti-rootkit software, and countermeasures implemented in those software. We have also outlined the general procedure of introducing rootkits and anti-rootkit software into course projects. Via the hands-on projects, students are able to develop awareness of system vulnerabilities in a networked environment, and gain first-hand experience of the attack-and-protection race involving rootkits.

REFERENCES

- [1] M. Russinovich and D. Solomon, *Windows Internals, 5th Ed.*, Microsoft Press, 2009.
- [2] S. Sparks, S. Embleton, and C. Zou. "Windows Rootkits - a Game of Hide and Seek", in Y. Xiao, F.H. Li, and H. Chen (Eds): *Handbook of Security and Networks, Chapter 19*, World Scientific Press, 2010.

- [3] Rootkit Unhooker:
<http://www.rootkit.com/vault/DiabloNova/RkU3.8.382.584.rar>
- [4] GMER Rootkit Detection Software: <http://www.gmer.net/>
- [5] B. Blunden. *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Plano, TX: Wordware Publishing, 2009.
- [6] M. Davis, S. Bodner, and A. LeMasters, *Hacking Exposed: Malware and Rootkits*, McGraw-Hill, September 23, 2009.
- [7] C. Ries, "Inside Windows Rootkits", Vigilantminds, Inc.,
<http://madchat.fr/vxdevl/library/Inside%20Windows%20Rootkits.pdf>, May 22, 2006.
- [8] Microsoft Malicious Software Removal Tool:
<http://www.microsoft.com/security/malwareremove/default.aspx>
- [9] Kaspersky Internet Security: <http://usa.kaspersky.com/>
- [10] Rootkit Revealer: <http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx>
- [11] U.S. Patent Application No. 11/271327 (applicant: Peter Szor), 02/16/2010.
- [12] Denning, P., "Great Principles in Computing Curricula," *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, 2004, pp. 336-341.
- [13] TCP View: <http://technet.microsoft.com/en-us/sysinternals/bb897437.aspx>
- [14] WinDebug: <http://www.microsoft.com/whd/c/devtools/debugging/default.mspx>
- [15] Derek's Boot and Nuke: <http://www.dban.org/>

Rootkit Unhooker LE v3.8.388.590 Service Release 2		
File	Action	Setup
S SSDT	Shadow SSDT	Processes
Drivers	Stealth Code	Files
Code Hooks	Report	
Hooked Object	Hook Address and Location	Type of Hook
ntoskrnl.exe+0x0005C9B4	0x828AC9B4-->828AC976 - [ntoskrnl.exe]	Inline - RelativeJump
ntoskrnl.exe+0x0005CAE0	0x828ACAE0-->828ACAAA - [ntoskrnl.exe]	Inline - RelativeJump
ntoskrnl.exe+0x0005CC38	0x828ACC38-->C0B87EF - [unknown_code_p...]	Inline - PushRet
[936]svchost.exe->ntdll.dll-->NtProtectVirtualMemory	0x770C5380-->0015000A - [unknwon_code_p...]	Inline - RelativeJump
[936]svchost.exe->ntdll.dll-->NtWriteVirtualMemory	0x770C5F00-->0022000A - [unknwon_code_p...]	Inline - RelativeJump
[936]svchost.exe->ntdll.dll-->KiUserExceptionDispatcher	0x770C6448-->0014000A - [unknwon_code_p...]	Inline - RelativeJump
[936]svchost.exe->user32.dll-->GetCursorPos	0x75FEC198-->00C4000A - [unknwon_code_p...]	Inline - RelativeJump
[936]svchost.exe->mswsock.dll+0x00002BBC	0x74C12BBC-->0051000A - [unknwon_code_p...]	Inline - RelativeJump
[936]svchost.exe->mswsock.dll+0x00004B1	0x74C14B1-->004F000C - [unknwon_code_p...]	Inline - RelativeJump
[936]svchost.exe->mswsock.dll+0x000046B7	0x74C146B7-->0050000C - [unknwon_code_p...]	Inline - RelativeJump
[1480]explorer.exe->ntdll.dll-->NtProtectVirtualMemory	0x770C5380-->004C000A - [unknwon_code_p...]	Inline - RelativeJump
[1480]explorer.exe->ntdll.dll-->NtWriteVirtualMemory	0x770C5F00-->004D000A - [unknwon_code_p...]	Inline - RelativeJump
[1480]explorer.exe->ntdll.dll-->KiUserExceptionDispatcher	0x770C6448-->001D000A - [unknwon_code_p...]	Inline - RelativeJump
[1480]explorer.exe->mswsock.dll+0x00002BBC	0x74C12BBC-->0065000A - [unknwon_code_p...]	Inline - RelativeJump
[1480]explorer.exe->mswsock.dll+0x00004B1	0x74C14B1-->0063000C - [unknwon_code_p...]	Inline - RelativeJump
[1480]explorer.exe->mswsock.dll+0x000046B7	0x74C146B7-->0064000C - [unknwon_code_p...]	Inline - RelativeJump
[3580]iexplore.exe->ntdll.dll-->NtProtectVirtualMemory	0x770C5380-->0035000A - [unknwon_code_p...]	Inline - RelativeJump
[3580]iexplore.exe->ntdll.dll-->NtWriteVirtualMemory	0x770C5F00-->0036000A - [unknwon_code_p...]	Inline - RelativeJump
[3580]iexplore.exe->ntdll.dll-->KiUserExceptionDispatcher	0x770C6448-->001D000A - [unknwon_code_p...]	Inline - RelativeJump
[3580]iexplore.exe->user32.dll-->CreateWindowExW	0x76000E51-->6EAC8157 - [C:\Windows\Syst...	Inline - RelativeJump
[3580]iexplore.exe->user32.dll-->DialogBoxIndirectParamW	0x76024AA7-->6EBEF5E8 - [C:\Windows\Syst...	Inline - RelativeJump
[3580]iexplore.exe->user32.dll-->DialogBoxParamW	0x7602564A-->6E9E4BA7 - [C:\Windows\Syst...	Inline - RelativeJump
[3580]iexplore.exe->user32.dll-->DialogBoxParamA	0x7603CF6A-->6EBEF585 - [C:\Windows\Syst...	Inline - RelativeJump
[3580]iexplore.exe->user32.dll-->DialogBoxIndirectParamA	0x7603D29C-->6EBEF64B - [C:\Windows\Syst...	Inline - RelativeJump
[3580]iexplore.exe->user32.dll-->MessageBoxIndirectA	0x7604E8C9-->6EBEF51A - [C:\Windows\Syst...	Inline - RelativeJump
[3580]iexplore.exe->user32.dll-->MessageBoxIndirectW	0x7604E9C3-->6EBEF4AF - [C:\Windows\Syst...	Inline - RelativeJump

Figure 2. Hooks Detected by Rookit Unhooker

DESIGN OF AN EXTENSIBLE INTERPRETER USING INFORMATION HIDING*

Larry Morell
Arkansas Tech University, Russellville, Arkansas 72801
479-968-0355
lmorell@atu.edu

ABSTRACT

Information hiding is applied to the design of an extensible interpreter, i.e., one which enables new grammar rules to be added at any time, even while program constructs are being interpreted. Each construct of the language is represented by one or more grammar rules; each grammar rule is implemented by a single class which contains all the information necessary for scanning, parsing, and interpreting the corresponding construct.

BACKGROUND

Complex systems, according to Parnas [1], are those implemented by a team of programmers which will likely experience significant change over their life cycle. Developing a large project in reasonable time requires the work be subdivided; Parnas calls such a programmer's work assignment a module of the system. Deciding what modules are needed for a system, defining the behavior of these modules and restricting their interrelationships are the central activities of design [1,2,3].

Information hiding [2] is a design principle used to decompose a system into modules that emphasizes reducing the impact of anticipated changes. Any design process consists of a set of design decisions. When a design decision changes, its related code may change. Minimizing the impact of design changes requires minimizing the extent to which one design decision affects other design decisions. Inter-dependent design decisions (design decisions which change in tandem) should be in the same module; design decisions which may change independently should be located in different modules.

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Modules must hide their design decisions from other modules, and reveal through its interface only information that is not expected to change.

Design by information hiding then involves three steps. First, determine the design decisions that might change and the extent to which each might change. These changeable design decisions may include how information is represented, when it is accessed and written, algorithms employed to process information, how quickly answers are computed, and how much space is required, what internal functions are called, etc. Second, name the modules that hide related information and define their interfaces, which tell other module authors what they may assume when about the module. Third, define which modules use other modules, minimizing mutual use [3].

Note that information hiding is not defined in terms of implementations or programming language constructs. The principle of information hiding applies to the design, and not necessarily to the implementation. A design based on information hiding may not be recognizable at run-time and there is no particular reason it should be. For example some operations of a module may be implemented by a macro; in such a case hidden secrets of one module are buried within another modules code, because it invoked the macro-based operation. Neither should information hiding be equated with object-oriented design. A good object-oriented design is likely to follow information hiding, but not all object-oriented designs preserve information hiding. Overuse of inheritance increased linkages among classes and reduces flexibility; design patterns based on information hiding enhance flexibility and reduce linkages among classes [4]. Designs based on information hiding can be implemented in any language.

In [2], Parnas contrasted information hiding with the traditional approach where "each major step of the processing is a module". Compilers and interpreters have mostly been implemented this way [3]. Interpreters must scan their input, create tokens, confirm that the sequence of tokens match a particular grammar, create some intermediate form, and then interpret the intermediate form. These processing-oriented modules typically share data structures such as token ids, list of first symbols, a symbol table, and the format of the intermediate representation.

What is wrong with such a design? Put simply, it may be appropriate for a language with stable syntax and semantics, but for a experimental language whose constructs regularly change, it is a nightmare. The author of this paper has built several interpreters by this approach, mostly throw-away class examples or simple interpreters which rarely need updating. However his interpreter for Genesis [6] has met with moderate success and has undergone hundreds of changes in the past 7 years, and he has paid the price for not following information hiding.

The interpreter's original design is traditional. There are five principal components: a scanner, a parser, an interpreter, and I/O module and a runtime module. Implemented in Java it uses classes which hide the representation of underlying data structures and was implemented in about 200 hours. The real test of a good design, however, is not how fast it can be implemented, but how readily it can be changed. Some changes were trivial, requiring only isolated modifications to the scanner, parser, or interpreter. Some new constructs, however, required changes in nearly every module of the system. Additionally, deleting a construct meant reversing these changes, often leaving code that confused additional modifications.

The author felt it necessary to redesign the system before proceeding to the next revision of the language for several reasons. First, it was hoped that students could be taught the structure of the system, and it would become a bed for many interesting student projects. However, as constructs were added, the complexity increased, and much more background was needed to enable students to enhance the language. Second, documentation for the system was difficult to maintain, since changes often affected multiple modules. Third, there was a strong desire to make the language extensible. Briefly, this means a program in the language will be able to define new programming constructs and use them as if they were native operations in the language. Extensibility would allow other instructors to tailor Genesis to their style of presenting algorithms.

THE NEW DESIGN

Three stages of information hiding were mentioned earlier. First, the aspects of the system that are likely to change and the degree to which they are likely to change are identified. The principal change for this project is the ability to add or delete constructs to/from the language. The second stage requires hiding design decisions that change in a single module. Since adding and removing construct are anticipated changes, each construct should be isolated in a separate module from other constructs, and should have all the information needed to process all aspects of that construct. This avoids having to modify more than one module when a construct is added, deleted, or modified. Interfaces must reveal only what is needed by calling modules.

The third stage requires defining the relationships between the modules so as to avoid mutual use. It was decided to use call-back routines via a registration system to limit the degree of connectedness among modules.

Each syntactic construct in the base language is represented by a distinct class. Each such class contains the following:

- A definition of the grammar rule being represented.
- A registration routine that registers the rule along with its parser and an associated translator to be applied when instance of the rule is parsed.
- Declaration of the state needed to record the results of the translation.
- A routine for evaluating (interpreting) the construct.

Note this places all the information necessary to scan, parse, and interpret a construct in a single Instruction module.

Other extensions such as operations for pretty printing and for translating to another language are possible.

Following is an instruction module for arithmetic expressions that satisfies the grammar rule

```
<ArithmeticExpression> ::= <Term> <AddOp> <ArithmeticExpression>.
```

```
public class ArithmeticExpression_AddOp extends GenesisInstruction {
    private static String rule =
        "<ArithmeticExpression> ::= <Term> <AddOp>
<ArithmeticExpression> ";
    private static SyntaxRule syntaxRule = new SyntaxRule(GRAMMAR, rule);
    private InstructionList il;
```

```

public static void initialize() { // static initialization
    GRAMMAR.addRule(syntaxRule,
        new ArithmeticExpression_AddOp(),
        new GenericParser(syntaxRule.lhs())));
}

public Instruction createInstruction(InstructionList il) {
    ArithmeticExpression_AddOp exp = new ArithmeticExpression_AddOp();
    Instruction first = il.getFirst(); // first operand
    Instruction second = il.get(1); // binary operator
    Instruction third = il.get(2); // second operand

    // See if the third is a Term
    if (third instanceof ArithmeticExpression_AddOp) {
        ArithmeticExpression_AddOp term = (ArithmeticExpression_AddOp)
third;
        term.il.addFirst(second);
        term.il.addFirst(first);
        exp.il = term.il;
    }
    else { exp.il = il; }
    this.il = exp.il; // save it with the local instruction
    return exp;
}

public StickyNote eval() {
    StickyNote accumulator = il.getFirst().eval();
    int n = 1;
    while (n < il.size()) {
        // Process a pair (operator, operand), applying the operator
        // to the accumulator and the operand
        BinaryEval operator = (BinaryEval) il.get(n);
        StickyNote operand = il.get(n + 1).eval();
        accumulator = operator.eval(accumulator, operand);
        n = n + 2;
    }
    return accumulator;
}
}

```

The class name captures the whole rule, not just the non-terminal on the left hand side. A grammar rule is defined as a string, encoded as a `SyntaxRule`. The `initialize()` routine (invoked from the main program), registers the syntax rule with the grammar, and associates with it two objects: an instance of this class that defines the grammar rule, and the parser to be used to parse any arithmetic expression. After the main program registers all rules of the base language with the grammar, it requests the grammar to begin parsing, which in turn invokes the parser associated with the start symbol of the grammar, which, in turn invokes appropriate parsers registered with right-hand sides as they are discovered. As a parser recognizes a maximal-length rule, it builds a list of `Instruction`'s created by calls to lower-level parsers. This list is passed to the `createInstruction()` of the registered `Instruction`. The `createInstruction()` given here for arithmetic expressions builds a list terms separated by addition operators (+ or -). Thus, input of $3 + 5 - 4$ will in a list of five instructions: [3] [+][5][-][4].

The result of parsing is a graph of `Instruction`'s. The main program invokes the root of the graph, by calling its `eval()` routine. This cascades into calls on other `eval()` routines with appropriate updates to the run-time environment. Instructions that can produce both l-values and r-values have two `eval()` operations. These are invoked to interpret the program. Evaluating [3] [+] [5] [-] [4] requires traversing the list from left to right, applying each operator encountered to the accumulated result and the next term.

The system uses a generalized parsing routine that runs off an extensible grammar. The system begins with an empty grammar. Calling the registration routine mentioned above adds a grammar rule to the system and informs it where to deliver the results of its parsing. The parser is oblivious to the number of grammar rules; this allows rules to be added to the system as it is parsing.

The Grammar module only needs to invoke a parser, but does not need to know how the parser works. This allows different parsers to be invoked if needed and eliminates a circular usage among parsers and the grammar. Similarly parsers invoke other registered parsers without knowing how they work. All parsers return a single result (called an `Instruction`). A parser for a rule with a right-hand side of N symbols will likely invoke a separate parser for each member of the right-hand side. When done, it will have collected a list of N instructions; this list is passed to an object of the class associated with this rule (via its `createInstruction()`), which stores the relevant parsed data, and returns a single instruction. This instruction is then used by higher-level parsers, stored in higher-level lists, and so on.

TRADEOFFS

Redesigning the interpreter has required several tradeoffs between implementation time and flexibility. The initial plan was to snip out all code related to a given construct (scanning, parsing, and evaluating) from the original implementation and move that code to a separate class. It became obvious that while parsers may be easily separable, scanners were not. A scanner may need to read beyond the token it is recognizing to recognize its end. When there is a choice of next token, it may be necessary to invoke each scanner until a desirable match occurs. In either of these cases backtracking may be needed. The solution? Information hiding, of course! An `AbstractBuffer` was defined to serve as a standard interface for buffered input; this interface was implemented via a `Buffer` class. Expecting that someday the parsing may involve backtracking, a `BufferWrapper` class was developed to wrap all the I/O operations of `AbstractBuffer`, applied to an arbitrary `Buffer`. To provide routines for backtracking, `BufferWrapper` was extended to `StackedBuffer`, to implement I/O routines to allow for backtracking.

The decision to replace individual recursive-descent parsing routines for each grammar rule with a centralized parse table was costly. Deciding how to build a generic parser to run off an extensible grammar took as much time as the original development. However, the reduction in overall complexity is enormous. Several thousand lines of parsing code is replaced by a few hundred lines of code distributed over all the `Instruction`'s. Effectively any intellectual activity involved in writing the old-style parser has been replaced with the effort needed to define a grammar rule, and the few lines of code needed to store parsed results in an instance of a class that defines the rule.

Since the new design requires a separate class for every grammar rule (*not* just for every non-terminal), there is an explosion of classes. However, the classes are nearly identical and several NetBeans code templates have been created to remove the tedium of defining most of their content. Approximately four grammar rules can be added to the system per hour. So ignoring the one-time overhead in developing the generic parsing system, the development speed using this approach for building a working system is much faster than building the original system.

CURRENT STATUS

Approximately 50 grammar rules have been added to the system. No timing tests have yet been conducted, but the parsing appears slower than the hand-written recursive-descent parser. Instruction classes average about 30 lines each, most of which are identical across all classes; approximated 10 lines differ per class. It is anticipated that there will be 200 grammar rules, which results in approximately 2000 lines of implementation code versus over 5000 lines in the current implementation. When complete the source code will be available on the web at <http://cs.atu.edu/~morell>.

SUMMARY

Information hiding, as defined by Parnas, is a design principle for splitting a system into modules. This research has applied information hiding to the design of an interpreter for an extensible language. The design needed to accommodate two types of change: adding and removing language constructs, and enabling the language to be extended at runtime. To add (or remove) a construct merely requires adding (or removing) classes that correspond to the construct, and modifying the main program to invoke (or not invoke) the procedure for registering the construct with the grammar/parsing system. It is likely that the registration in the main program will use reflection in the future by the system discovering and registering grammar rules present in the compilation directory. To enable the language constructs to be extended at run time requires a mechanism for enhancing the grammar/parsing system at run time. This mechanism is already at the core of the current system, so run-time enhancement of the grammar is possible within the current system.

Traditional parser-generator systems allow one to define the syntax of a construct along with the construction of its intermediate form in one place; the interpreter for the intermediate form must be supplied separately. In this approach all aspects of defining a construct (scanning, parsing, and interpretation) are defined in a single class. This, along with run-time construct extension should enable others to extend and contract the language as desired.

REFERENCES

- [1] Parnas, D. Some Software Engineering Principles. *Software fundamentals: collected papers by David L. Parnas*, Daniel M. Hoffman, and David M. Weiss, eds., Addison-Wesley Longman Publishing Co., 2001. pp. 257 - 266.

- [2] Parnas, D. On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, 15 (12), 1972.
- [3] Parnas, D. Designing software for ease of extension and contraction, *IEEE Transactions on Software Engineering*, SE-5 (2), 128-138, 1979.
- [4] Freeman, E. and Freeman E., *Head First Design Patterns*, Sebastopol, CA: O'Reilly Media, 2004.
- [5] Andrew W. Appel, *Modern Compiler Implementation in Java*, 2nd Edition. Cambridge University Press, 2002.
- [6] Morell, L., Algorithms in Genesis, MSCCC '04: *Proceedings of the 2nd Annual Conference on Mid-south College Computing*, Little Rock, AR, April 2004, pp 5-16.

MANAGING VENDOR ALLIANCES*

PANEL DISCUSSION

*Janet S. Renwick (contact person)
University of Arkansas - Fort Smith*

*Stephen Baber
Harding University*

*Brian Henehan
University of Arkansas - Fort Smith*

*Bruce Thigpen
University of Arkansas - Fort Smith*

At the core of computing technology is innovation and change. Keeping current is an on-going challenge, in terms of both faculty expertise and budgetary demands. Mutually beneficial vendor alliances help answer this challenge.

Computing departments need current technology in order to support learning. Vendors benefit by gaining access to a guaranteed set of users - users who will be required to use their products and thus likely to recommend its usage to others.

Vendor alliances, however, require administration by faculty or IT staff. While the cost of participating in the alliance may be low, administering it to hundreds of students may require significant effort. Some alliances are easier to administer than others due to built-in limitations.

This panel will discuss the benefits and challenges of participating in vendor alliances. Attendees will benefit from the exchange of ideas and strategies.

Moderator:

Janet S. Renwick, PhD, Associate Professor of Information Technology, University of Arkansas - Fort Smith.

Panelists:

Stephen Baber, PhD, Professor of Computer Science, Harding University.

Dr. Baber serves as the Computer Science Department's contact with regard to academic contracts with Microsoft, Intel, Apple and VMWare.

* Copyright is held by the author/owner.

Brian Henehan, Assistant Professor of Information Technology, University of Arkansas - Fort Smith.

Mr. Henehan manages vendor alliances with Microsoft and VMWare for over 800 enrolled students.

Bruce Thigpen, Assistant Professor of Information Technology, University of Arkansas - Fort Smith.

Mr. Thigpen is the Cisco Networking Academy Legal Main Contact and Curriculum Lead at UAFS.

AN INTRODUCTION TO PROGRAMMING IN HASKELL*

CONFERENCE WORKSHOP

*Gabriel J. Ferrer
Associate Professor
Hendrix College*

*Department of Mathematics and Computer Science
1600 Washington Avenue
Conway, AR 72034
(501) 450-3879
ferrer@hendrix.edu*

In this hands-on tutorial, the functional programming language Haskell will be introduced. No prior experience with functional programming in general or Haskell in particular will be assumed. Haskell is of particular interest in that as a purely functional language, it contains much potential for innovation in software verification. Due to its lazy evaluation strategy, Haskell programmers are able to experiment easily with different types of algorithms and data structures in comparison to those available in mainstream programming languages. The features of Haskell have influenced contemporary languages such as Python and Perl 6.

The tutorial will begin with an overview of the use of higher-order functions and list comprehensions as Haskell's primary iteration mechanism. Particular attention will be paid to exploring the implications of lazy evaluation in regards to performance. This will be followed by an exploration of the major data structures used in the purely functional Haskell context.

Following this will be an exploration of Haskell's type system, including the module system, algebraic data types and pattern-matching. Particular attention will be paid to type classes, the Haskell approach to disciplined function and operator overloading.

Property testing using the randomized tester Quickcheck and the model finder Smallcheck will next be described. This unit of the tutorial will demonstrate the advantages of pure functions for the purpose of software verification.

Purity and referential transparency introduce difficulties for I/O. The use of monads in Haskell to preserve purity while still allowing for all of the I/O operations present in conventional programming languages will be discussed in detail. Particular attention will be devoted to exploring how monads enable the use of Haskell's type system to make principled use of side effects.

* Copyright is held by the author/owner.

The tutorial will conclude with an overview of some of Haskell's popular libraries, as well as a discussion of future trends in the language.

Each unit will involve hands-on exercises to enable participants to learn actively as the tutorial progresses. Interested participants will be guided in installing Haskell on their own computers.

SHORT MOBILE GAME DEVELOPMENT PROJECTS FOR

INTRODUCTORY CS COURSES*

CONFERENCE WORKSHOP

Delvin Defoe (contact person)
Department of Computer Science & Software Engineering
Rose-Hulman Institute of Technology
5500 Wabash Avenue, Terre Haute, IN 47803, USA
Tel: 1-812-877-8815
Fax: 1-812-872-6060
defoe@rose-hulman.edu

Stan Kurkovsky
Department of Computer Science
Central Connecticut State University
1615 Stanley Street, New Britain, CT
06050, USA
Tel: 1-860-832-2720
kurkovskysta@ccsu.edu

Emily Graetz
Rose-Hulman Institute of Technology
5500 Wabash Avenue, Terre Haute, IN
47803, USA
Tel: 1-812-872-6165
graetzer@rose-hulman.edu

Game development and mobile computing have been successfully used to increase student motivation. However, instructors with no background in mobile computing, computer graphics, and/or game development may find it difficult to develop or adopt course materials on these topics. This workshop is designed to address these concerns. Using Java Micro Edition, we have developed several project-based course modules focused on mobile game development and designed to study fundamental programming principles (e.g. loops) while also exposing students to more advanced concepts (e.g. databases). Using a mobile phone emulator, participants will test-drive one of our modules and develop a simple game, which can then be transferred to and played on a mobile device. A Windows or Mac laptop is recommended.

Intended audience: This workshop is intended for faculty members teaching college-level and high school Java-based courses in CS who are looking to improve student motivation. All other faculty members interested in integrating game development as a motivational tool in their courses will also benefit from this workshop.

Presenter's Biography: Delvin Defoe is an Assistant Professor of Computer Science and Software Engineering at Rose-Hulman Institute of Technology. His research interests

* Copyright is held by the author/owner.

include dynamic memory management, multicore computing, mobile game development, and computer science education. He has published several articles on memory management, a Nifty Assignment on modeling software engineering techniques to CS 1 students, and looks forward to making substantial contribution in the other areas. Dr. Defoe is currently involved in a program to motivate high school students to pursue undergraduate studies in STEM disciplines. Dr. Defoe teaches undergraduate courses in Software Development, Operating Systems, Computer Architecture, and the Theory and Practice of Garbage Collection. He recently participated in a Summer School on Multicore Programming in an effort to introduce parallel computing in the CSSE curriculum and served on an NSF TUES 1 proposal review panel.

Stan Kurkovsky is a Professor of Computer Science at Central Connecticut State University (CCSU). His research interests are in the areas of mobile and pervasive computing, software engineering, and computer science education. He has written over 60 peer-reviewed conference papers and journal articles, several of which discuss using mobile game development in CS classrooms (including a SIGCSE-2009 paper). Dr. Kurkovsky is currently serving as a PI on an NSF STEM scholarship grant providing financial support for academically talented students majoring in Computer Science, Mathematics, and Physics. Dr. Kurkovsky teaches undergraduate and graduate courses in Software Engineering, Computer Networks, and Mobile Computing. Dr. Kurkovsky has successfully taught the course in the focus of this workshop; students in this course reported overwhelmingly positive experiences. Most recently, he participated in an ITiCSE working group, which conducted a large-scale multinational study that addressed many issues of attracting, engaging, and retaining CS students.

Emily Graetz is a Student at Rose-Hulman Institute of Technology majoring in Computer Science and Mathematics. She is particularly interested in low-level programming, artificial intelligence, and Einstein's theories of relativity. A two-time participant in the USAMO, Emily has scored highest of all Rose-Hulman students in nearly all mathematics competitions she has entered, including the Putnam and Virginia Tech competitions. During the past year, she has constructed a working processor on an FPGA efficient enough to run all instructions in one minimum length cycle and written a Scheme interpreter capable of loading itself three times recursively. Emily is working on a tic-tac-toe program designed to learn autonomously all of its techniques during game play. She has been involved with the mobile game development project for the past year, creating sample student solutions and a central server for project testing.

Drs. Defoe and Kurkovsky are currently serving as the PIs on a collaborative NSF CCLI grant "Using Mobile Game Development to Improve Student Learning and Satisfaction in Introductory Computer Science Courses." This workshop presents some of the work stemming from this grant project, which has also been applied in CCSU and Rose-Hulman courses.

Materials provided: A number of free software packages will be used during the workshop. These include Java Platform Micro Edition Software Development Kit 3.0, J2SE SDK version 1.6 or later, and Mappy Editor for tiled maps. The workshop participants will be provided with CDs containing all software, as well as with handouts describing how this software can be downloaded, installed, and used. Provided CDs will also contain the source code, data, and relevant course materials for the currently existing

mobile game development modules. Workshop attendees will also be invited to register on the project website that hosts all current versions of the modules, along with a discussion board for instructors interested in or those who have already adopted our course materials.

Length of time required:

1. Mobile games as a motivational and engagement tool (20 min)
2. Overview of the course modules (30 min)
3. Test-driving a module (80 min)
4. Wrap-up (20 min)

Audio/Visual and Computer Requirements: Ideally, this will be a hands-on workshop in a computer laboratory setting where every participant could follow and experiment with the presented software. The workshop can also be presented as long as the computer running Windows with a projector and an Internet connection is available to the presenter. Wireless access will be a plus since participants will be encouraged to bring their laptops. **Laptops are recommended.**

Space and Enrollment Restrictions: There are no special requirements for the configuration of the room.

Other critical information: Today, most college students have mobile phones and spend a considerable amount of time using them for browsing the web, texting, or playing games. Introducing students to mobile application development allows students to better relate to the course material and make a stronger connection to real-world applications they see every day. Mobile game development is by far less complex than traditional game development due to its smaller scale and simpler graphics. Students are always interested in computer game development; the course model presented in this workshop brings this topic to students very early in the curriculum and serves as a good tool to increase student retention.

AMICABLE PAIRS IN PARALLEL*

NIFTY ASSIGNMENT

Gabriel Foust

Harding University, 915 E Market Ave, Searcy AR 72149

(501) 279-4434

gfoust@harding.edu

This assignment is meant to expose students learning C++ in CS1 to some basic ideas in parallel programming. Not only is this an interesting glimpse into an advanced field of computer science, but it also encourages students to begin thinking about what types of problems can be solved in parallel. To complete the assignment requires knowledge of basic branching and looping constructs—in particular, the for loop—as well as knowledge of basic array manipulation. No knowledge of parallel programming concepts is required; details are hidden by using OpenMP. By giving the assignment immediately after introducing arrays, it may help reinforce basic array concepts in addition to providing exposure to concepts of parallel programming.

Note that this assignment requires the use of some sort of multi-processor (e.g. dual-core) machine in order to obtain the described performance gains. It also requires the use of a C++ compiler which supports OpenMP. My students used Visual Studio 2008, Professional Edition; however, current versions of GCC also support OpenMP. The full assignment description may be found at the following URL:
<http://ozark.hendrix.edu/~burch/ccsc-nifty/10/fou/>

The assignment is to calculate all amicable pairs less than 100,000 using an array. An amicable pair is a pair of numbers for which the sum of the proper factors of the first number gives you the second number, and vice versa. For example, 220 and 284 make an amicable pair because the sum of the proper factors of 220 is

$$1 + 2 + 4 + 5 + 10 + 11 + 22 + 44 + 55 + 110 = 284$$

and the sum of the proper factors of 284 is

$$1 + 2 + 5 + 71 + 142 = 220$$

The students are provided with an algorithm to solve this problem which first builds an array containing the sum of the proper factors of all numbers less than 100,000, then searches it for matches. Although this not the most efficient algorithm, it was chosen because the work of building the array can be easily done in parallel.

* Copyright is held by the author/owner.

Once the basic algorithm is working, the students insert instructions to time it. (On our 3GHz, dual-core machines the program takes about a minute to run.) Next the students use OpenMP to execute it in parallel. OpenMP is an API for writing parallel programs in C/C++ and Fortran. Using OpenMP greatly simplifies this assignment as a single directive can be used to execute a for loop in parallel. Thus, the work of creating threads and dividing the work between them is hidden from the students. The students use the OpenMP directive in three different ways:

1. A directive to execute the loop which calculates the sums in parallel. This yields only modest gains in efficiency because the loads are not shared in a balanced fashion.
2. A directive to execute the same loop in parallel, but this time using scheduling to share the loads in a more balanced fashion. This yields considerable gains in efficiency.
3. A directive to execute the loop which reports the results in parallel. This yields garbled output, and illustrates that not every loop is suitable for parallel execution.

This assignment received favorable feedback from students. It is a simple enough program that students can easily understand what is going on, and it produces very dramatic improvements from using parallel programming.

COMMUNICATION AND ORIGAMI *

NIFTY ASSIGNMENT

Mark Goadrich

Mathematics and Computer Science Department

Centenary College of Louisiana

Shreveport, LA 71104

318-869-5194

mgoadric@centenary.edu

This assignment is designed to be the first lab of an introductory course on computer science. I assume no prerequisites of prior programming experience, and this assignment is independent of the language chosen. Students often stumble when thrown into writing code because of two competing issues: they must learn the semantics of how to think like a computer scientist, and at the same time they must learn the syntax of how to speak like a computer scientist. Here I focus only on the importance of clear and concise syntax in the context of describing instructions for origami.

Communication, be it between people, computers, or extraterrestrial intelligences, is a central concept in computer science. In all communication, one entity has information and wants to convey it to another. Arguably, pictures are one of the most basic forms of communication, found in ancient Petroglyphs and Cave Paintings. They are still used today in the form of Symbol Signs for universal communication in airports, train stations, etc. If you encounter someone who does not speak your language, drawing pictures and pantomiming are great ways for both of you to communicate. There will be many times during the semester when students wish they could show the computer a picture of what they would like it to do instead of writing computer code. Unfortunately when conversing with computers or ETs, we will not have this luxury.

This assignment uses a language students are familiar with (English), but in a way that simulates the experience of talking with a computer. Students will precisely describe a set of instructions (called an algorithm) for folding an Origami dove, using no pictures or talking, and hope that someone else is able to understand their algorithm. An origami pattern is usually communicated with both pictures and words, allowing it to mostly transcend languages, however here the two will be separated. In computer science terms, the student will assume the role of Programmer, their lab subject will be the Compiler, and together they will be producing an origami dove for the end User.

In particular, the importance of this assignment is the evaluation by the student of their own ability to communicate clear and concise instructions. Students are asked to reflect on comments from the lab subject, in response to the questions "What was the

* Copyright is held by the author/owner.

most difficult step to follow? Were any steps missing?" They also reflect on their own experience, answering questions such as "What was the most difficult concept to convey with words? What elements of contextual information did the subject use? Were there any steps that you felt were easier to describe with English than with pictures?"

Files can be found at <http://www.ccsc-ms.org/nifty/10/goa>

TopSpin*

NIFTY ASSIGNMENT

Mark Goadrich

Mathematics and Computer Science Department

Centenary College of Louisiana

Shreveport, LA 71104

318-869-5194

mgoadric@centenary.edu

This assignment is designed for a CS2 Data Structures and Algorithms course, the assignment is based on Java but can be easily ported to other languages. TopSpin is a mechanical puzzle composed of twenty small numbered plastic pieces connected in a loop, where part of the loop overlaps a rotating disc large enough for four pieces. These pieces can be rotated as a group so that different pieces can be in the disc to be rotated. The goal of the puzzle is to place the pieces in ascending order. This assignment asks the students to create new data structures for the TopSpin puzzle using two different underlying implementations, one with arrays and another with circular doubly-linked list nodes.

The TopSpin interface consists of four methods. `shiftLeft()` and `shiftRight()` are public void methods that rotate the pieces one space to the left or to the right. `spin()` is also public and void, and reverses the pieces currently inside the spin mechanism. Finally, `isSolved()` is a public boolean method that returns true if the puzzle pieces are in numerical order.

?The assignment consists of students writing two implementations. The first implementation of the TopSpin interface should use an array as the basis of the TopSpin game. Students will devise an efficient implementation based on what we have studied about queues, stacks and lists, each of which generally included additional integer components to track the state of the collection.

The second implementation of the TopSpin interface should use the ideas from linked Lists as the basis of the TopSpin game to create a doubly-linked list capable of rotating some internal pieces. Devise an efficient implementation based on what we have studied about queues, stack and lists and the pointer manipulation involved. Students must use the `DoublyLinkedListNode` in this implementation.

Many courses teach the use of data structures and supply the code for the basic structures of stacks, queues and trees along with the text for the course. In this assignment, students will apply their knowledge of the standard List implementations and

* Copyright is held by the author/owner.

perform algorithmic analysis in a new domain, as well as hone their object-oriented programming skills.

Files can be found at <http://www.ccsc-ms.org/nifty/10/gob>

OBJECT-ORIENTED SPACE PHYSICS *

NIFTY ASSIGNMENT

Carl Burch

Hendrix College, 1600 Washington Ave, Conway AR 72032

(501) 450-1377

burch@hendrix.edu

To teach object-oriented concepts in Java during CS1, I use a matched pair of short, guided assignments emphasizing the application of computers to the n-body problem and particularly to space navigation. This has the advantage of not only teaching object-oriented concepts, but also seeing an application of computers to scientific simulation and reviewing some basic physics. Both assignments display the simulation graphically. Though the assignments were designed for a curriculum using the ACM Java Task Force's API, the assignments themselves have little direct dependence on this API; it was little trouble to port the assignments so that they use Swing alone. The full assignment description and supporting code are at <http://www.ccsc-ms.org/nifty/10/bur/>.

The assignments are cumulative. In course evaluations, students often report that they appreciate such cumulative assignments. But this also introduces the problem of students who don't complete the first part and so are at a disadvantage for the second. This assignment pair attempts to address this issue by keeping the first part fairly short, though it does often require a bit of instructor help. Another way to address the problem of students failing to complete the first part is to schedule the assignments so that students start the second a full week after the first is due; that way, students who submit the first portion late will still be able to start the second on time.

The first assignment is a simple n-body simulation for six bodies, including the sun, the inner four planets, and a comet (Encke). The student's job is to write a class Body that manages a body's position within the solar system, which includes the physical equations for computing force. Many students are not quite able to get these physical equations exactly right and will require instructor help to debug them.

This assignment is intended as the first assignment where students develop their own classes. The assignment provides a skeleton class to the students, and they are told to add several specified methods into the class. The student will also be required to add some new instance variables representing the body's current velocity in the x- and y-direction.

The second assignment is meant as a first assignment after discussing how one writes subclasses and overrides methods. It adds a Spaceship class to the mix of the preceding assignment, in order to represent a ship that is controlled via the keyboard.

* Copyright is held by the author/owner.

Because the spaceship is itself a body in the system, subject to the same gravitational forces, it is naturally a subclass of the Body class of the previous assignment. However, the instance method that updates a body's velocity must be overridden to account not only for gravity but also for the force exerted by the ship's rockets.

DEMONSTRATING THE NEED FOR PRECISION TO FRESHMEN*

NIFTY ASSIGNMENT

David Middleton

*Department of Computer and Information Science,
Arkansas Tech University,
Russellville, AR 72801
479 968-0628
dmiddleton@atu.edu*

This assignment has been used in our Orientation to Computing course to show first-time college students the extent to which their usual narrative is imprecise: their usual descriptions omit crucial details and convey others ambiguously. Demonstrating that tangible discrepancies inevitably follow motivates them to pay more attention to detail.

The in-class, manual, activity, described at <http://www.ccsc-ms.org/nifty/10/mid/> in more detail, takes about 10 to 15 minutes to complete, although it requires about one or two hours of preparation beforehand (possibly by a student worker).

The class is divided into pairs, whose task is to reproduce a diagram projected on the screen, using paper shapes and glue sticks which are provided. The member closer to the front turns around to face their team-mate. It is emphasized that every student will only see one diagram: those behind see and describe the projected image, those turned around duplicate the described pattern, hiding their work with the provided manila folder.

Even with the opportunity for intelligent agents to query ambiguous directions, their results usually show significant errors in reproduction. Shapes and colors are deliberately chosen to increase the probability of this occurring: for example, the letter 'R' is easily named and recognized even when it is back to front; the builder will naturally place it correctly if the teller fails to note its reversal. Using several asymmetrical shapes, and placing shapes so they interleave illustrates to students their difficulty in describing intricate situations.

The debriefing discussion afterwards is usually low-pressure, and hence quite animated. As team efforts, the final results are no particular person's fault. Discrepancies noticed by the instructor can be described without naming specific teams. Invariably, one or two students will have peeked. This provides a lead into discussions of ethics and the

* Copyright is held by the author/owner.

honor code, and the idea that students are here to gain experiences and skills, not to produce better paper patterns than other students.

FUNCTION COMPOSITION IS A GOOD PRACTICE*

NIFTY ASSIGNMENT

Cong-Cong Xing

Department of Mathematics and Computer Science

Nicholls State University

Thibodaux, LA 70310

cong-cong.xing@nicholls.edu

INTRODUCTION AND MOTIVATION

Functional programming (FP) is an important programming paradigm that differs dramatically from imperative programming paradigm. It was a unanimous agreement and a strong recommendation by the program committee in the First ACM SIGPLANWorkshop on Undergraduate Programming Language Curricula [1] that FP is to be included in the undergraduate computer science curriculum. In this abstract, we introduce a FP assignment that, we believe, would be helpful for students to grasp some key features and flavors of FP.

One of the distinguishable characteristics of FP is that functions are “first class citizens”, in the sense that functions can be treated in the same way in which data values are treated. In particular, functions can be passed as arguments to (other) functions and/or can be returned as results of (other) functions. This feature, especially the latter part, is generally considered beyond the programming domain that the imperative programming can handle.

Given this fact and considering that function-composition is a familiar to topic to computer science students (those who have taken discrete math), we believe that implementing the function-composition function in some FP language (e.g., ML [2]) is a good programming practice, and have given such an assignment to our students many times (using the free system Standard ML of New Jersey [3]). Interestingly, through correcting the work turned in by the student, we have found that some instances of the (incorrect) programs written by the student can actually be used to clarify some FP issues and to enhance the teaching of FP. Here, we consolidate the function-composition task with one typical instance of the student work, and propose an integrated FP assignment.

THE ASSIGNMENT

The complete assignment and hints for solutions can be found at
<http://www.ccsc-ms.org/nifty/10/xin/>.

* Copyright is held by the author/owner.

WHAT CAN BE GAINED

Through this (short) assignment, students can learn and reinforce several essential concepts and skills associated with FP, including

- *Higher-order function* – functions that take functions as arguments and/or returns functions as results.
- *Currying* – how to express functions that take multiple arguments as functions that only one argument.
- *Anonymous function* – how to define a function without naming it.
- *Type system* – a system that checks, assigns, and infers types for each expression.
- *Polymorphism* – polymorphism can be natural in FP (as in ML) due to the notion and use of type variables (comparable to generic types in Java).
- *λ -calculus and FP* – consistent with the application order of λ -terms, function application (in ML) associates to the left.

SCOPE OF USEABILITY

Obviously, this assignment can be used in an introductory FP course. For institutions where no such a course is offered, this assignment can be considered in a junior- or senior-level Programming Languages course where different programming paradigms (including FP of course) are briefly studied.

FINAL REMARK

Based on our teaching experiences, we have proposed a FP assignment which is centered around the function-composition function. We believe that this assignment involves several important aspects of FP and hope that it can be found useful by other colleagues.

REFERENCES

- [1] <http://www.sigplan.org/pl-workshop>, 2008.
- [2] Jeffrey D. Ullman, *Elements of ML Programming*, Prentice Hall, 1998.
- [3] <http://www.smlnj.org/>, 2009.

**Papers of the
Seventeenth Annual
CCSC
Central Plains
Conference**

**April 8-9, 2011
University of Central Missouri
Warrensburg, Missouri**

WELCOME — 2011 CCSC: CENTRAL PLAINS CONFERENCE

It is my pleasure to welcome you to the University of Central Missouri for the 17th annual Consortium for Computing Sciences in Colleges Central Plains Region Conference.

The conference this year boasts an appealing and diverse program. I am honored to have two distinguished speakers, an industry expert, Mr. Stuart Thomas from Hallmark as the keynote speaker and an educator, Dr. John McCormick from the University of Northern Iowa as the banquet speaker. Additionally, the program will include a broad range of paper presentations, panels, tutorials, workshops, lightning talks, nifty assignments, and a substantial collection of student research posters. The usual programming contest should be enjoyable for everyone, as well.

The paper acceptance rate for this year was 64%. Moreover, each paper was reviewed by at least three reviewers. This ensures that the papers accepted in this program continue to be first-rate. I am certain that the conference program will benefit both computer science educators and students.

I am privileged to have worked with a group of dedicated people. Without the devoted committee members, reviewers, session chairs, and many other volunteers, this conference would not be achievable. I would also like to express my gratitude to the administration, staff, and my colleagues in the Department of Mathematics and Computer Science at the University of Central Missouri who helped make this conference a success. Finally, thanks to the numerous individuals, vendors, and organizations whose support helped make the conference possible.

I am pleased to be hosting this year's conference in Warrensburg, a beautiful city with many conveniences. And I am especially pleased to have the conference at the University of Central Missouri, which features a large and diverse student body.

I hope you find the conference both enlightening and enjoyable. I look forward to seeing you next year at the Ozarks Technical Community College in Springfield, Missouri.

Mahmoud Yousef
University of Central Missouri
CCSC-2011 Central Plains Conference Chair

2011 CENTRAL PLAINS CCSC REGIONAL BOARD

MEMBERS

Scott Sigman, Regional Representative and Board Chair.....
..... Drury University, Springfield, MO
Gary Schmidt, Registrar and Membership Chair.. Washburn University, Topeka, KS
Dean Sanders, Regional Editor. Northwest Missouri State University, Maryville, MO
Scott Bell, Secretary. Kansas State University, Manhattan, KS
Michael Rogers, Webmaster. . Northwest Missouri State University, Maryville, MO
Judy Mullins, Regional Treasurer.
..... University of Missouri-Kansas City, Kansas City, MO
Mahmoud Yousef, Conference Chair.....
..... University of Central Missouri, Warrensburg, MO
Wen Hsin, Past Conference Chair..... Park University, Parkville, MO
George Gibeau, 2012 Conference Chair. . Ozark Technical College, Springfield, MO

2011 CENTRAL PLAINS CCSC CONFERENCE STEERING

COMMITTEE

Mahmoud Yousef, Conference Chair, Local Arrangements, Publicity.....
..... University of Central Missouri, Warrensburg, MO
Mustafa Kamal, Conference Co-Chair.
..... University of Central Missouri, Warrensburg, MO
Bob Neufeld, Papers Chair. McPherson College, McPherson, KS
Baochuan Lu, Papers. Southwest Baptist University, Bolivar, MO
Ron McCleary, Panels and Tutorials Chair..... Avila University, Kansas City, MO
Tom Mertz, Panels and Tutorials..... Kansas State University, Manhattan, KS
Carol Spradling, Nifty Course Assignments Chair.
..... Northwest Missouri State University, Maryville, MO
George Gibeau, Nifty Course Assignments.....
..... Ozark Technical Community College, Springfield, MO
Wen-Jung Hsin, Lightning Talks Chair..... Park University, Parkville, MO
Henry Walker, Submission/Review System. Grinnell College, Grinnell, IA
John Dooley, Submission/Review System. Knox College, Galesburg, IL
Tim DeClue, K-12 Teachers Committee Chair.
..... Southwest Baptist University, Bolivar, MO
Carol Spradling, K-12 Teachers Committee.
..... Northwest Missouri State University, Maryville, MO
Scott Bell, K-12 Teachers Committee Kansas State University, Manhattan, KS
Rick Barker, Student Posters Chair. Washburn University, Topeka, KS

BaoQiang Yan, Student Posters.. Missouri Western State University, St. Joseph, MO
Brian Hare, Student Programming Contest Chair.....
..... University of Missouri - Kansas City, MO
Scott Bell, Student Programming Contest... Kansas State University, Manhattan, KS
Mustafa Kamal, Student Programming Contest.
..... University of Central Missouri, Warrensburg, MO
Michael Rogers, Regional Vendors.....
..... Northwest Missouri State University, Maryville, MO
Shing So, Graduate Schools..... University of Central Missouri, Warrensburg, MO
Curtis Cooper, Registration. University of Central Missouri, Warrensburg, MO
Scott Sigman, At-Large Member..... Drury University, Springfield, MO
Dean Sanders, At-Large Member.
..... Northwest Missouri State University, Maryville, MO
Judy Mullins, At-Large Member..... University of Missouri - Kansas City, MO

REVIEWERS — 2011 CCSC CENTRAL PLAINS CONFERENCE

Mohammad Alanazi..... Imam University, Riyadh, Saudi Arabia
Charles Ashbacher..... Mount Mercy College, Cedar Rapids, IA
Pavel Azalov. Penn State University, Hazelton, PA
Sambit Bhattacharya. Fayetteville State University, Fayetteville, NC
Beverly Bohn. Park University, Parkville, MO
Nick Breems..... Dordt College, Sioux Center, IA
Carol Browning. Drury University, Springfield, MO
John Buerck. Saint Louis University, Saint Louis, MO
Chia-Chu Chiang. University of Arkansas at Little Rock, Little Rock, AR
Monica Costa..... Polytechnic Institute of Castelo Branco, Castelo Branco, Portugal
Tim DeClue. Southwest Baptist University, Bolivar, MO
Jeffrey Edgington. University of Denver, Denver, CO
James Feher. McKendree University, Lebanon, IL
Ernest Ferguson. Northwest Missouri State University, Maryville, MO
Charles Frank. Northern Kentucky University, Highland Heights, KY
Nadeem Hamid. Berry College, Mount Berry, GA
Cindy Hanchey. Oklahoma Baptist University, Shawnee, OK
James Harris. Pittsburg State University, Pittsburg, KS
Phillip Heeler. Northwest Missouri State University, Maryville, MO
Dennis Herr. Missouri Southern State University, Joplin, MO
Wen-Jung Hsin. Park University, Parkville, MO
David Klappholz. Stevens Institute of Technology, Hoboken, NJ
Myungsook Klassen. California Lutheran University, Thousand Oaks, CA
Janet Kourik. Webster University, St. Louis, MO
Srinivasarao Krishnaprasad. Jacksonville State University, Jacksonville, AL

Ingyu Lee.....	Troy University, Troy, AL
Noel LeJeune.....	Metropolitan State College of Denver, Denver, CO
Hui Liu.....	Missouri State University, Springfield, MO
Baochuan Lu.....	Southwest Baptist University, Bolivar, MO
Rick Massengale Sr..	University Arkansas Fort Smith, Fort Smith, AR
Ron McCleary.....	Avila University, Kansas City, MO
Robert McCloud.....	Sacred Heart University, Fairfield, CT
Jim McKeown.....	Dakota State University, Madison, SD
Bruce Mechtly.....	Washburn University, Topeka, KS
Jose Carlos Metrolho.....	Instituto Politenico de Castelo Branco, Castelo Branco, Portugal
Judy Mullins.	University of Missouri - Kansas City, Kansas City, MO
David Naugler.....	Southeast Missouri State University, Cape Girardeau, MO
Robert Neufeld.....	McPherson College, McPherson, KS
Crystal Peng.....	Southwestern College, Winfield, KS
Chuck Pheatt.....	Emporia State University, Emporia, KS
David Pope.....	Ozarks Technical Community College
Dean Sanders.....	Northwest Missouri State University, Maryville, MO
Jamil Saquer.....	Missouri State University, Springfield, MO
Cecil Schmidt.....	Washburn University, Topeka, KS
Terry Scott.....	University of Northern Colorado, Greeley, CO
Onkar Sharma.....	Marist College, Poughkeepsie, NY
Ching-Kuang Shene.....	Michigan Technological University, Houghton, MI
Carol Spradling.....	Northwest Missouri State University, Maryville, MO
John Stamey.....	Coastal Carolina University, Conway, SC
Keith Tookey.....	Eureka College, Eureka, IL
Timothy Urness.....	Drake University, Des Moines, IA
Gary Ury.....	Northwest Missouri State University, Maryville, MO
Nancy Van Cleave.....	Eastern Illinois University, Charleston, IL
Ian van der Linde.....	Anglia Ruskin Univ., UK
Ken Vollmar.....	Missouri State University, Springfield, MO
Henry Walker.....	Grinnell College, Grinnell, IA
George Whitson.....	The University of Texas at Tyler, Tyler, TX
Paul Wiedemeier.....	The University of Louisiana at Monroe, Monroe, LA
John Wilson.....	Regis University, Denver, CO
Baoqiang Yan.....	Missouri Western State University, St. Joseph, MO
Mahmoud Yousef.....	University of Central Missouri, Warrensburg, MO

KEYNOTE ADDRESS

Friday, April 15, 2011

DEVELOPING A WAREHOUSE CONTROL SYSTEM IN C# *

**Stuart A. Thomas
Principal Analyst
Hallmark Cards, Inc.
Kansas City, MO
sthoma4@hallmark.com**

This presentation will focus on the development cycle of a service oriented solution to handle warehouse process control systems at the Hallmark Cards Liberty Distribution Center. Insights will be shared from the design through the development, testing and implementation phases. You will be guided through the storage, pulling and picking of product by stacker cranes and conveyor systems. You will discover what exactly is a VAX program and more importantly how to convert it to .NET. You will be given a review of the challenges and trade-offs caused by budget, schedule, resources and business constraints. Finally, you will learn about the type of skills and traits that one should have to become a successful software developer.

Mr. Stuart A. Thomas is currently employed as Principal Analyst at Hallmark Cards, Inc. Stuart graduated from University of Central Missouri in 1981 with a B.S.B.A. in Data Processing. Following his graduation he spent the next 8 years working for the Kansas City Power & Light company as a COBOL programmer. In 1989, he joined Hallmark Cards as a VAX process control programmer at the Liberty Distribution Center in Liberty, Missouri. For the past 22 years, his responsibilities within the I/T organization have included programming, technical leadership, project management, and application architecture design. He has gained a vast amount of experience in developing applications using VAX C, C++, Java, and C# languages running on various platforms. He is now serving as the technical lead/architect for a project that is in the process of converting the current Shipping, Warehouse and Carousel control systems written in the 1990's on a VAX/VMS platform to a Windows platform using the .NET technologies.

* Copyright is held by the author/owner.

BANQUET ADDRESS

Friday, April 8, 2011

HOW TO TRAIN YOUR COMPUTER: THE NEED FOR QUALITY EMBEDDED SYSTEMS EDUCATION *

John W. McCormick

**Computer Science Department
University of Northern Iowa
Cedar Falls, IA 50614-0507
319-273-6056
mccormick@cs.uni.edu**

Nearly every electronic device we use contains an embedded processor. Our entertainment, communication, transportation, finances, and health are all dependent on the quality of software in these devices. Today, much of the software in these devices is written by electrical engineers with a course or two in C programming. The inevitable increase in software complexity and society's ever growing dependence on embedded software requires development team members with skills beyond basic C programming - the skills of a computer science graduate. As computer science educators, we are tasked with the job of preparing our graduates to play a part in the development of safe, reliable software.

One of the problems with teaching embedded real-time systems at a small school is in obtaining the necessary resources. I will begin by describing my experiences developing a laboratory for a junior level course in real-time embedded software development based on a large model railroad. This laboratory has been duplicated at small schools in North and South America, Europe, and Australia. We will look at thirteen years of data comparing the different programming languages used in this course and at what language features had positive effects on student programming team success.

Next we'll look at some of the strengths and weaknesses of Ada in both education and industry concentrating on correctness and reliability. Motivated by the increased use of formal methods for creating safe, secure software, we'll conclude with a brief look at SPARK, an annotated subset of Ada. The annotations allow the SPARK Examiner to generate conjectures which can be proven by the SPARK Simplifier or Proof Checker to

* Copyright is held by the author/owner.

verify that the program meets its specification. SPARK has successfully proven correctness in systems with over a million lines of code contradicting the common belief that proof is only viable for toy programs.

THREE HOURS, TWO APPS, NO PROBLEM! A QUICK INTRODUCTION TO IPHONE/IPAD DEVELOPMENT*

PRE-CONFERENCE WORKSHOP

*Michael Rogers
Computer Science/Information Systems
Northwest Missouri State University
800 University Drive
Maryville, MO 64468
mprogers@mac.com*

The iPhone and iPad are revolutionary technologies that have transformed the computing landscape. Computer Science students own these devices, and naturally want to learn how to write apps for them. For faculty interested in obliging, there is, unfortunately, a dauntingly steep learning curve. The purpose of this workshop is to help flatten that curve. It will start with an introduction to the language, frameworks and development tools in which all of the apps that are available on the App Store - from Angry Birds to Zulu Time - are developed. Participants will then create two modest apps that will illustrate the major design patterns common to all apps.

REQUIREMENTS

A knowledge of OOP, and a willingness to learn new technologies!

* Copyright is held by the author/owner.

AN EFFECTIVE WAY OF INTRODUCING IPHONE

APPLICATION DEVELOPMENT TO UNDERGRADUATE

STUDENTS*

Baoqiang Yan and Deborah Becker and Connie Hecker
Department of Computer Science, Mathematics and Physics
Missouri Western State University
Saint Joseph, MO 64507 USA
{byan, dbecker, hecker}@missouriwestern.edu

ABSTRACT

Since its release in 2007, the iPhone has not only become a novel mobile device but also turned into a popular application development platform. Many universities offer courses to introduce iPhone application development. As an advocate of applied learning, MWSU (Missouri Western State University) provides the same opportunity for students to delve into the exciting world of iPhone programming. Based upon our successful experience, we briefly describe in this paper an effective way of teaching iPhone application development. This includes what topics to cover in the underlying programming language, the core frameworks, what development tools to use, the design patterns that help the students grasp the essentials of iPhone programming and the sample applications that were developed.

1. INTRODUCTION

As of the writing of this paper, Apple Inc. has released four generations of its iPhone since 2007. By April 2010, 51.15 million devices have already been sold worldwide in less than 3 years [1]. The existence of such a huge body of users spurs the prosperity of iPhone application development, especially after Apple released its SDK for native iPhone application development. These native iPhone applications no longer require a

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

remote server as the web-based ones do, and can run on an iPhone as any built-in application.

Apple has weaved a beautiful touchable dream for a Computer Science student. Not only can they own such a fantastic device, but also they can design and build their own creative applications then sell them via Apple's app store. Many universities want to offer courses to introduce iPhone application development at an undergraduate level so that the students can start delving into the exciting world of iPhone programming on their journey to such success. As an advocate of applied learning, MWSU provides students with this same opportunity. This course was first offered as an 8-week course during Summer 2010 and it turned out to be a huge success. Most students not only grasped the essentials of iPhone programming, but also created functional real-world iPhone applications.

In this paper, we would like to share our experience of how to quickly and effectively introduce iPhone application development to undergraduate students. Typically, issues tackled in teaching new programming courses include things such as syntax of the underlying language, available frameworks, development tools and specific applications to develop. iPhone programming requires more. Students also need to get familiar with the design patterns that are widely used in iPhone project templates, such as MVC (model-view-controller), delegation, singleton and target-action.

Section 2 describes the role of Objective-C in iPhone programming and its new features that we need to cover compared with Java/C++. Section 3 discusses iPhone SDK and the frameworks used in iPhone programming. Section 4 talks about the tools used for iPhone application development and testing. Section 5 discusses the design patterns employed in iPhone application templates that students should know of. Section 6 briefly describes the real-world applications completed by our students. Section 7 makes conclusion remarks and discusses the possibility of integrating iPhone programming with other courses.

2. THE LANGUAGE - OBJECTIVE-C

Objective-C is the underlying programming language for iPhone application development. It follows object-oriented programming (OOP) paradigm. Students who take the iPhone course must have some background in OOP, usually either C++ or Java. This way, course content on the language itself focuses only on the new features in Objective-C that are essential to OOP, such as how to define a class, instantiate an object, access the data fields of an object, define and invoke an action, represent the relations between classes and objects through inheritance and polymorphism, and how to restrict the behaviors of classes using protocols.

2.1. Class Definition

In OOP, a class basically represents a type or category of objects that share the same set of data fields or properties, and perform the same set of actions. Different from Java but quite similar to C++, the definition of a class in Objective-C is composed of two sections, interface and implementation [2]. These two sections can be put in either the same file or separate files. The data fields of a class are declared within curly braces

inside the interface section. The interface section also contains the method headers for the actions available in that class. The implementation details of each method are put in the implementation section.

Like Java and C++, the method header in Objective-C is composed of modifier, return type, method name and a parameter list. However, the modifier is denoted using either - for instance methods or + for class methods. The return type and the parameter types are enclosed within parentheses. The parameters are specified following colons. Optionally but desirably, each parameter except the first one could have a descriptive argument name in front of the colon.

2.2. Object Instantiation

The creation of an object in Objective-C involves two steps, memory allocation and data field initialization [2]. These are realized by passing the message *alloc* to a class and then the *init* message to the newly created instance. These two steps can be combined into one step by just passing a *new* message to a class. We will talk about message syntax in section 2.4.

There exists a special type of object called *class object* in Objective-C. In other words, classes are themselves objects. A class object can be instantiated by passing the message *class* to the corresponding class.

2.3. Data Access

Objective-C provides convenient dot syntax for accessing the instance variables of an object. However, the *dot syntax* assumes that the corresponding set and get methods have already been defined with certain name formats. These methods could be either explicitly provided by the application developer or automatically generated by the compiler when the corresponding data fields are declared as properties using the *@property* directive in the interface section and synthesized using the *@synthesize* directive in the implementation section. The synthesized methods are more efficient and thread safe [2].

2.4. Method Invocation

Method invocation is called message passing in Objective-C. A message represents an action to be performed by the receiver. The receiver of a message could be either a class or a specific object. Objective-C uses a special messaging syntax expressed within a pair of square brackets.

2.5. Class Extension

There are two different ways to extend existing classes in Objective-C. Besides the inheritance mechanism common to all object-oriented programming languages, Objective-C also supports *category*. Category allows the developer to just extend the

behavior of a class without adding new data fields. It is also useful to divide the methods of a class into several groups at the very beginning of the class design [2].

2.6. Polymorphism

Polymorphism allows objects of different types (or rather of different classes) to be referenced by the same kind of variable. In Objective-C, variables of the generic type `id` can reference any objects. These objects can receive the same message but respond differently.

There are two issues involved in polymorphism, *type checking* and *method binding*. Type checking is the determination of the class that an object belongs to. Method binding is the determination of the method that will eventually be invoked. Objective-C always performs dynamic typing and binding in polymorphism, which means no error messages will be generated until run time [2]. To avoid runtime errors, Objective-C allows the developer to manually check the type (namely class) of an object and whether it responds to a certain message (namely method invocation) or not before the method is actually called. This involves using a new selector type `SEL` to represent a message.

2.6. Protocol

Protocols in Objective-C are similar to interfaces in Java. They both specify what methods need to be implemented in the implementing class. However, the methods in Java interface must be all implemented for a class to be a concrete class that can be used to instantiate an object, while the methods in an Objective-C protocol could be optional. The Delegation design pattern, which is widely used in iPhone application development and will be talked about in section 5, is actually implemented using protocols. Therefore it is important for the students to understand how protocols work in Objective-C.

2.7. Memory Management

Memory management is critical in iPhone application development because of two facts. First, the iPhone has limited memory. Second, garbage collection is not supported in the iPhone OS. So iPhone application developers should release the memory of unneeded objects promptly instead of waiting for the whole application to terminate. Students should be aware that, an application that runs fine in the iPhone simulator might freeze or even crash the whole system when deployed on a real device.

Objective-C uses reference counting to manage the memory occupied by an object. Sending an object the `release` message only decrements its reference count by one. The object's memory will not be reclaimed until its reference count reaches zero. The method that eventually reclaims the memory is called `dealloc` instead of `release` [2]. The `dealloc` method is inherited from `NSObject` and can be overridden in application specific classes.

3. IPHONE SDK AND THE FRAMEWORKS

To develop iPhone applications, the students must register with Apple as a developer and download the iPhone SDK (now called iOS SDK). Luckily, the registration fee for students can be waived through Apple's iPhone Developer University Program.

Many application development frameworks can be used in iPhone programming. At least, the students should know the core frameworks and their relationships. The Foundation Framework provides basic classes such as wrapper classes and data structure classes. All the classes in Foundation Framework use the prefix NS. Both Cocoa and Cocoa Touch Frameworks include the Foundation Framework, but they have different frameworks for GUI. Cocoa has AppKit in it while Cocoa Touch uses UIKit. Cocoa is for Mac application development while Cocoa Touch is for iPhone application development [3, 4]. Other frameworks, such as MapKit, AddressBook, can be imported into your iPhone project if needed.

4. THE TOOLS - XCODE, INTERFACE BUILDER AND SIMULATOR

XCode is the first tool students should learn since they can use it purely for practicing Objective-C. When it comes to iPhone application development, the Interface Builder would be required for creating GUI interface. The iPhone simulator is a free software that replicates the iPhone OS on a computer and can be used to test iPhone applications before they are deployed on real devices.

XCode provides a rich set of project templates for different kinds of applications. If the application is iPhone based, some NIB files with the extension .xib will also be included in the template's Resources folder. Clicking NIB files inside XCode will launch the Interface Builder where the developer works on the GUI interface of the application. The whole application development process involves switching between the Interface Builder and XCode many times.

If the iPhone simulator is selected in a project's active configuration as the target platform, once built successfully, the application will be launched in the simulator for testing. Eventually, the students will be excited to see their own applications running on a real device. For that purpose, the students must visit Apple's Developer Portal, request a development certificate signed by Apple and generate a provisioning profile for each application they want to deploy. The provisioning profile basically ties an application to the devices listed in their developer profile. Once the provisioning profile is ready, it can be used to install the corresponding application on the device through XCode's Organizer.

5. DESIGN PATTERNS

The first time the students create a new iPhone application, they are faced with several complex template files in XCode and UI elements wired into the Interface Builder. The students need to know how these files and UI elements are related to each other so that they can do further customization or add new components and features. In fact, the layout of template files and UI elements, and their relationships reflect the design patterns employed in iPhone SDK to facilitate application development. Design pattern is a software engineering technique to make your application more architectural.

Object-oriented design patterns typically show relationships and interactions between classes or objects. Understanding the patterns in iPhone programming helps the students in rapid application development.

5.1. Singleton Design Pattern

In singleton pattern, there exists a class that can be instantiated only once and that object is used to coordinate actions across the system. The UIApplication, a class in UIFramework to start an iPhone application in the main.m, follows the singleton design pattern. Every iPhone application has only one instance of UIApplication at run time to orchestrate the lifecycle of that application [3].

5.2. Delegation Design Pattern

In delegation pattern, an object, instead of performing one of its stated tasks, delegates that task to an associated helper object. Delegation pattern is implemented using protocol and is widely used in iPhone application development [3,4]. For example, the default delegate class in all iPhone applications conforms to theUIApplicationDelegate protocol. This class can be customized for application cycle events by implementing the various methods that UIApplication will call in prescribed order. UITableView and UITextField also use delegation pattern for table presentation or extracting user inputs.

5.3. Model-View-Controller (MVC) Design Pattern

In MVC pattern, model means application data or state represented in various formats, such as a class object, a database, or a network stream. View means how the data is presented to the user. The same data could be presented in different ways. The same view could be used to present different data. There is no coupled relationship between Model and View. Model and View never directly talk to each other. The interaction between Model and View is carried out through the Controller. The Controller is responsible for presenting the data in a view and allows the user to manipulate the data through the view.

The iPhone application development relies heavily on the MVC design pattern. In iPhone application, the connection between Model and Controller is established by creating an instance of the Model in the controller class. Creating IBOutlets (in the controller class) for the UI elements and wiring them into a NIB file sets up the connection between View and Controller. A NIB file contains the view (represented by the UI elements), the controllers (one for each view), and their relations through outlets and event-action mappings. A NIB file never has a model file in it.

5.4. Target-Action Design Pattern

View controllers use the target-action design pattern to notify your application of user interactions. When the user interacts with a UI element in a predefined way, the control sends a message to the specified object. Upon receiving the action message, the target object can then respond in an appropriate manner.

View controls usually allow users to initiate some type of action through some events such as touch event, value changed, or editing event. Actions (messages) are invoked (sent) on (to) certain target objects (the controllers not the data model). The whole operation involves three components: target, action and event. The developer sets up their connections either through IB or programmatically [3,4].

So far, we talked about the key technical topics that help the students quickly grasp the essentials of iPhone programming. They must be explained with well-crafted examples. It would be even better if the students can apply their newly acquired skills to specific projects, which may require additional topics. For example, our student projects also need to deal with database, networking and push notification, etc.

6. STUDENT PROJECTS

Our students had the opportunity to work on real world projects needed by the community. A local Heartland Health in Saint Joseph, MO sponsored a contest for our iPhone-programming course. Their IT department provided a list of possible iPhone applications that they wish could be implemented by our students and used by their physicians. They offered cash prizes totaling \$3,000 for the top three teams. Over the eight-week summer session, our students successfully developed functional prototypes for several of the requested applications.

The app that won the first place is called SurveySays created by Noah Hendrix. It allows the user to quickly create and deploy a survey to their organization or audience and receive their response in real-time. As shown in figure 1, SurveySays follows the client-server model. The administrator of some organization can connect to a centralized server, either from a regular computer or from their iPhone, to create a survey. The system will then utilize push notification mechanism to notify their

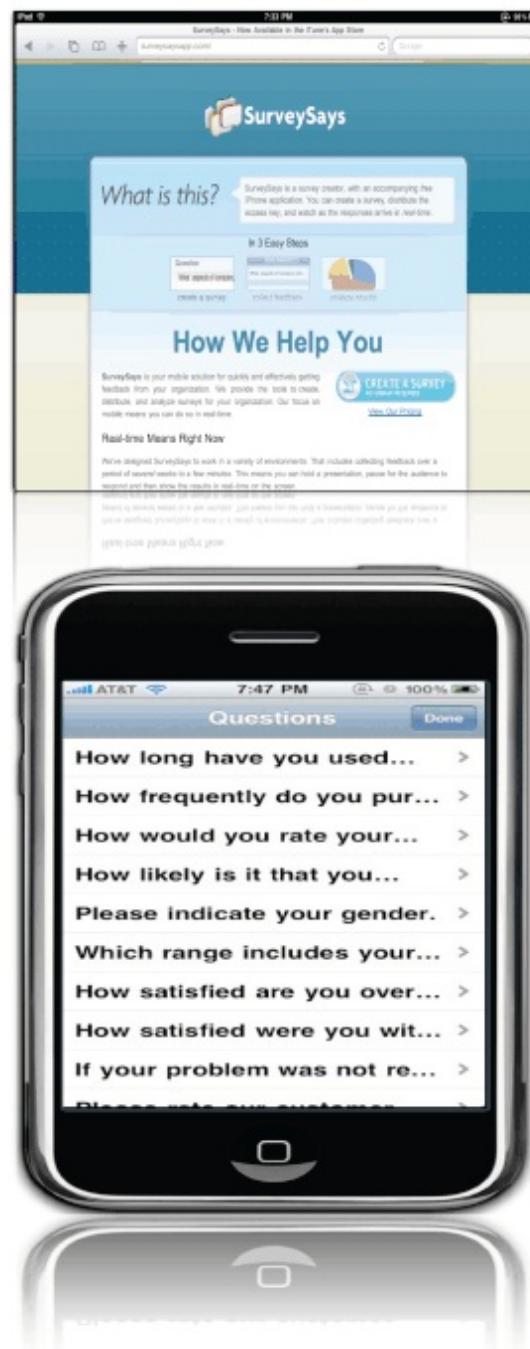


Figure1: SurveySays. These two screen shots are courtesy of Noah Hendrix.

employee of this new survey for them to take, again either on a regular computer or through their iPhones. Every time a new survey entry is inserted, it will be displayed on the client side. Every time a new response is completed, the statistics about this survey will be automatically available to the administrator. SurveySays works so elegantly that the Heartland Health would like to use it immediately, saying this course exceeded their expectation.

7. CONCLUSIONS AND FUTURE WORK

iPhone programming is getting popular. Based upon our successful teaching experience, we summarized the topics to cover so that the students can quickly grasp the essentials in iPhone application development. These include the unique syntax features in Objective-C, the iPhone SDK and core frameworks, development tools and design patterns. We briefly described the best project accomplished by our students.

We pointed out that working with local organizations and businesses on "live" student software project is very challenging but worthwhile. iPhone application development is essentially a kind of software development that involves not only the technical issues, but also communication with the client to meet their requirements and documentation to help the client use the software. These clearly should not be a major part of an iPhone course. We see opportunities to integrate the iPhone course with other courses such as our Analysis and Design, and Software Implementation. Actually this collaboration has been underway.

8. REFERENCES

- [1] <http://www.mobilecrunch.com/2010/04/20/apple-q2-earnings-million-iphones/>
- [2] Stephen G. Kochan, Programming in Objective-C 2.0 (2nd edition), Addison-Wesley
- [3] Bill Dudney and Chris Adamson, iPhone SDK Development: Building iPhone Applications, Pragmatic Bookshelf, ISBN-13: 978-1-93435-625-8, 2009
- [4] Apple, Inc. The iPhone Application Programming Guide. Apple Inc. 2009

THERE AND BACK AGAIN: LEVERAGING iOS

DEVELOPMENT ON MAC OS X*

Michael P. Rogers
Northwest Missouri State University, Maryville, MO
309-825-6454
mprogers@mac.com

ABSTRACT

More and more universities have been offering courses on iOS Development. With this knowledge, students can develop apps on iPhones, iPads, and iPod Touches, but they can do more: most of what they have learned applies directly to Mac OS X development, and so by expending just a little more effort they can expand their horizons into a novel and empowering realm. This paper describes the relationship between iOS and Mac OS X, a transition path for iOS-savvy students wishing to learn how to write Mac OS X apps, and the motivation for undertaking such a journey.

1. INTRODUCTION

With the commercial success and huge amount of publicity that has surrounded the release of the iPhone, iPad, and iPod Touch, it is not surprising that numerous universities are now offering courses on iOS development [1-4]. Students enrolled in these courses face a steep learning curve. The majority likely have no experience with the language used to develop iOS apps, Objective-C, as it remains one of the more obscure dialects of C [5]. Likewise, the frameworks (Cocoa Touch) and Integrated Development Environment (IDE) of choice (Xcode) are also likely to be unfamiliar.

Once these topics have been mastered, however, students are able to develop apps in any number of areas, and enjoy the recognition and possible financial gain that comes from having them accepted to and distributed via the App Store.

By virtue of the platform on which they run, though, these apps are somewhat constrained. They run on smaller screens, on slower processors, and employ keyboards

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

that, while ingenious, are less than ideal for extended text input [6,7]. Finally, the apps, while running on a UNIX variant, do so in a sandbox, and consequently cannot tap into much of the power of the underlying OS.

Some students may find themselves chafing at these limitations, and yet remain eager to develop using the same IDE, frameworks, and languages. It is not simply that they will have invested so much time in acquiring, and therefore be reluctant to set aside, their hard-won knowledge. The technology is intrinsically appealing: the combination of the Cocoa Touch frameworks and the IDE (Xcode and Interface Builder) make it possible to wire together powerful apps with great ease and alacrity.

Happily, there is a route for these students, and it is, in a sense, a return to roots: iOS is an offshoot of Mac OS X, and there are compelling reasons for encouraging these students to come full circle, to explore iOS's progenitor.

In this paper, we explore these reasons. We begin by outlining the differences between iOS and Mac OS X, from a student developer's perspective. We then consider in detail the numerous benefits of developing for the latter platform. Finally, we describe the creation of a single app, TyperTimer, for both platforms, to illustrate just how much overlap there is between the two APIs.

2. A COMPARISON OF iOS AND MAC OS X: A STUDENT DEVELOPER'S PERSPECTIVE

For many Computer Science students, languages are a major concern, so it is therefore fortuitous that iOS and Mac OS X share one. Objective-C - a superset of ANSI C that includes object-oriented extensions based on SmallTalk [8] - is Apple's preferred language for writing general-purpose Mac OS X applications, and the only rigorously supported language for iOS development.

Moving from iOS to Mac OS X, students will discover at least one reason to celebrate: garbage collection, which is disabled on iOS device runtimes (for the sake of battery life and performance), is supported on Mac OS X. For those who dread Objective-C's seemingly byzantine memory management system, this will come as a particularly welcome change.

There is just one set of software development tools for both iOS and Mac OS X development, namely Xcode (an IDE), Interface Builder (for laying out interfaces), and Instruments (for performance monitoring). When a student launches Xcode, they will merely have to choose a Mac OS X template rather than an iOS template, to begin creating a Mac OS X project. The project organizer, editor, debugger and documentation bookshelves are identical. Xcode includes 5 Mac OS X templates, and just 2 iOS templates, an inkling that there are more avenues open to Mac OS X developers.

In iOS, student developers spend most of their time interacting with the Cocoa Touch frameworks, which incorporate two key sub-frameworks. Foundation is comprised of the most commonly used non-visual classes (NSStrings, NSNumbers, NSArrays, NSThreads, etc.). UIKit consists of classes that have a visual representation on the screen (UIView, UIImageView, UIButton, UITextField, etc.).

Shifting from iOS to Mac OS X, student developers will find that existing Foundation classes are largely unchanged [9]. Several additions appropriate for a computer-based application have been added, the most significant of which is bindings: controls can be linked either programmatically or in Interface Builder, so that a change in one control automatically changes the other.

Changes in the user interface classes are more extensive. The Mac OS X framework, known as AppKit rather than UIKit, can display multiple windows simultaneously; menu support is built-in; an entire sub-framework devoted to managing documents is included; NSTableViews can support multiple columns (UITableViews, in use on devices with smaller screens, support only one); even coordinate systems are oriented differently (the origin is at the bottom-left in an NSWindow, as opposed to top-left in a UIWindow).

Adjusting to these changes is not as daunting as it might sound, because for the most part they add functionality. As we discuss in section 4, a simple iOS app, consisting of a single view with multiple controls, can be migrated over to Mac OS X with little more than a few changes in class names and methods. The most difficult challenge is to design an application that can take advantage of all the extra features available on Mac OS X.

Up until recently, one huge advantage of iOS over Mac OS X development had been the existence of the App Store. It provides a convenient system for app distribution and a means for students (and instructors) to obtain an outside and rigorous [10] assessment of their efforts. Apple has now introduced a Mac OS X analogue. Students can still distribute Mac OS X applications via other channels, but the Mac App Store provides greater visibility, an implicit endorsement by Apple, and an exciting target for students to aim for.

3. THE BENEFITS OF MAC OS X DEVELOPMENT

3.1 HARDWARE BENEFITS

The most obvious hardware benefit of Mac OS X development can be seen at a glance: larger screens mean that more data or detailed imagery can be portrayed in its entirety, and, if necessary, windows can be expanded or multiple windows displayed simultaneously. When it is necessary to have multiple applications open at once, the screen size makes it possible to view windows in each, and transfer data back and forth.

In contrast, on iOS devices there is a single, fixed, small window. Large amounts of data must be divided into several views, and some mechanism provided so that the user can page through these views; images may be enlarged to show more detail, and iOS device displays are high resolution, but then only a small fraction of the image is visible. Everything feels slightly confining. iOS 4 does support multitasking, but since the devices only support one window, transferring data among applications takes more effort.

iOS devices are marvels of engineering, but they are not nearly as speedy as the computers that run Mac OS X. Table 1 [11] shows that, in Geekbench 2 Tests, the iPad is much slower than even the MacBook Air.

Device	Processor	Score
MacBook Pro (13", Early 2010)	Intel Core 2 Duo P8800, 2.66 GHz	3655
MacBook Air (13", Late 2010)	Intel Core 2 Duo L9400 1.86 GHZ	2695
iPad	Apple A4	453

Table 1

In spite of all this, iOS devices typically do not feel sluggish, nor do their screens feel inadequate: successful developers design and target their apps for the most appropriate hardware. A developer considering a word processing app would probably be better off on a computer; a developer considering a GPS-related app would likely prefer an iOS device (one with a built-in GPS).

One final hardware-related advantage of Mac OS X development is that getting an app to run on an actual device is easy; the program that is built in Xcode is ready to execute and distribute. For an iOS app, however, the last steps, in order to transfer an app to a physical device, require the student become a registered iOS Developer (\$99/year), download several certificates from Apple, upload one of these certificates to the device, and code sign their app [12].

To summarize, students interested in creating an app with the sort of immersive experience that require larger screens; that is more computationally demanding; in which data management is key; which lends itself naturally to multitasking; and requires no extra effort to run on its targeted platform, would be better off with a Mac OS X device.

3.2 SOFTWARE BENEFITS

Mac OS X comes with more mature and diverse frameworks than iOS. Students studying computer graphics will appreciate that a full OpenGL implementation is available (iOS ships with a pared down version, OpenGL ES); they may also have reason to use Accelerate, a powerful framework for doing image manipulation and vector-based mathematical calculations in hardware. Students studying parallel computing can solve embarrassingly parallel problems using Xgrid[13], a technology that, because it ships with all Mac OS X computers, makes it possible to set up a computer cluster in a lab setting with very little effort. At a lower level, Mac OS X supports a distributed objects architecture that allows transparent inter-process and inter-machine communication: writing servers and clients using NSDistributedObjects is straightforward. Students in operating systems have the option to write daemons (system-wide background processes) and agents (background processes designed for a particular user). Finally, all students will enjoy NSSpeechSynthesizer and NSSpeechRecognizer, two versatile but delightfully easy-to-use classes that are lacking (and sorely missed) in iOS. An aural version of "Hello, world" takes just two lines:

```
NSSpeechSynthesizer * speaker = [[NSSpeechSynthesizer alloc] init];
[speaker startSpeakingString:@<b>"Hello, world!"</b>];
```

Migrating from iOS to Mac OS X development, students will find the platform more open, allowing access to a plethora of UNIX tools, shells and languages. Students who are familiar with the likes of awk (a programmable text processing system), bc (a scriptable calculator), grep (a regular expression processor) and scripting languages (e.g., perl, python, and ruby), will instantly recognize and appreciate the tremendous power that this puts at their disposal.

Access to all this power comes through NSTask and NSPipe classes. These make it possible to execute a particular task, and pipe the output from one task to another, respectively. NSTasks are not just limited to UNIX tools. Many Mac OS X applications have a command-line interface (Podcast Producer, MySql) as well, and they can be controlled in a similar fashion. None of this is possible with iOS devices. iOS apps run in a tightly controlled sandbox, and have no means to invoke any applications directly.

4. A SAMPLE APPLICATION

In order to illustrate just how readily a student, versed in the iOS APIs, could adapt, we developed TyperTimer, a small app to assess a user's keyboarding prowess. We wrote it first in iOS and then ported it to Mac OS X. We did so by literally pasting the code from the completed iOS app into the skeleton-code of like-named classes in a Mac OS X application, and then began editing. The process was enlightening: encouraging students to perform a similar task with one of their iOS apps would make for an excellent project.

Code that did not involve GUI - calls to NSString, NSDate, NSTimer and NSArray objects - functioned identically on both platforms, without change. We chose to disable garbage collection on Mac OS X to be able to make the preceding statement: we could have saved a few lines had we done otherwise.

There were no major conceptual changes in the GUI code, but adjustments were necessary. There were numerous class and method name changes, a few of which are noted in Table 2:



	Classes				
	UITextField	UIButton	UIAlertView	AVAudioPlayer	UILabel
Mac OS X	NSTextField	NSButton	NSAlert	NSSound	NSTextField

	Methods				
iOS	text, setText	setTitle: forState:	initWithTitle: message:delegate: cancelButtonTitle: otherButtonTitles:otherButtonTitles	play	text, setText
Mac OS X	stringValue setStringValue	setTitle:	alertWithMessageText: defaultButton: alternateButton:otherButton:informativ eTextWithFormat:	play	stringValue setStringValue

Table 2

This small sampling of a few classes and methods may make the changes seem cosmetic and perhaps even arbitrary; in fact, the designers of the iOS APIs have carefully streamlined and refined the classes to make them easier to use, without alienating Mac OS X developers.

Most iOS and Mac OS X apps use Interface Builder to lay out the user interface in what are known as .xib files. These .xib files contain an XML specification of the interface's objects, including their attributes and connections to other objects. One side effect of the redefinition of classes is that these .xib files must be recreated from scratch - one cannot simply copy an iOS .xib file into a Mac OS X project.

The iOS APIs use properties, invoked with dot notation, much more so than in the Mac OS X APIs. This tends to result in pithier code. For instance, to assign a value to a textfield, we would write, in iOS and Mac OS X, respectively:

```
iOSTextField.text = @ "Hello, Bilbo";           // UITextField * iOSTextField
[macOSXTextField setStringValue:@"Hello, Bilbo"]; // NSTextField *
macOSXTextField
```

Other changes include slight variations in the life cycle of UI objects created in .xib files; and a diminution of the importance of view-controllers (in iOS, UIViews and UIViewController are intertwined, fundamental, and created automatically in Xcode templates; in Mac OS X, the default templates provide an NSWindow, but the user must supply all other necessary classes).

5. CONCLUSIONS

Students writing iOS apps, by virtue of their position in front of a Mac OS X computer running Xcode, have all that they need to begin exploring a variety of exciting new dominions, from numerical computation to computer graphics to distributed computing, using languages, IDEs and frameworks that they have already mastered. The transition is relatively painless, and by exploring Mac OS X they will also learn more about iOS; they will see the changes that expert API designers have implemented; and by porting their own iOS apps over to Mac OS X, they will, in one project, learn more about the importance of MVC than they would by any other means.

6. REFERENCES:

- [1] Zardon. University Classroom Courses, 2010,
www.stanford.edu/class/cs193p/cgi-bin/drupal/.
- [2] Chen, B.X. UC Davis Teaches iPhone App Development, Too, 2009,
www.wired.com/gadgetlab/2009/12/uc-davis-teaches-iphone-app-development-too/.
- [3] iPhone Programming, 2010,
www.uh.edu/continuingeducation/professional/iphone.php.
- [4] Johnson, P.D. University of Maryland Introduces iPhone Programming Course, 2010, www.newsdesk.umd.edu/uniini/release.cfm?ArticleID=2077.
- [5] TIOBE Programming Community Index for November 2010, 2010,
www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
- [6] Gyford, P. Pen v keyboard v Newton v Graffiti v Treo v iPhone, 2010,
www.gyford.comphil/writing/2010/01/18/input.php.
- [7] Hoggan, E., Brewster, S.A., Johnston, J. 2008. Investigating the effectiveness of tactile feedback for mobile touchscreens. *CHI 2008 Proceedings*, 26, 1573-82, 2008.
- [8] Kochan, S.G. *Programming in Objective-C 2.0*. Upper Saddle River, NJ: Addison-Wesley, 2009.
- [9] Apple Inc., Migrating from Cocoa, 2010, tinyurl.com/2byjpc.
- [10] Apple Inc., App Store Review Guidelines, 2010,
developer.apple.com/appstore/resources/approval/guidelines.html.
- [11] OSXDaily, New MacBook Air 11" and 13" Benchmarks, 2010,
osxdaily.com/2010/10/25/new-macbook-air-benchmarks/.
- [12] Apple Inc., 2010, iOS Developer Program, 2010,
developer.apple.com/programs/ios.
- [13] Apple, Inc., 2010,
developer.apple.com/library/mac/#documentation/MacOSXServer/Conceptual/Xgrid_Programming_Guide/Introduction/Introduction.html.

A TOOL FOR TEACHING PETRI NETS*

Chao Mei, Xiaoyang Zhang, Wenfei Zhao, Kasi Periyasamy and Mark Headington

*Department of Computer Science
University of Wisconsin-La Crosse
La Crosse, WI 54601
Phone: (608) 785-6823
Email: periyasa.kas2@uwlax.edu*

ABSTRACT

Petri nets are a notation used to model and analyze the behavior of concurrent and distributed systems. They are used in many applications such as work flow descriptions and, recently, in certain medical applications. This paper describes the development of a Petri net tool that is mainly intended to teach Petri nets to beginners. Consequently, the authors focused more on making the tool relatively simple to use yet sufficiently detailed to aid in understanding the important concepts of Petri nets. The tool was developed in Java and uses XML notation for storing the underlying representations; hence, it is easily portable to several platforms. The paper includes a simple case study on the model of a vending machine to illustrate the expressive power of the tool. A brief discussion on continuing work on the tool is also included.

INTRODUCTION

A Petri net is a design notation used for modeling systems behavior, particularly concurrent and distributed systems. Petri nets are complementary to state transition diagrams in describing the dynamic behavior of a system. However, it is beneficial to observe the dynamic behavior in a Petri net with graphical tool support because the tokens in a Petri net can be used to represent the flow of data, information or entities around the system. In contrast, a state transition diagram represents only the architecture of the system in terms of the states of the system, but it is hard to execute a state diagram and observe the dynamic behavior.

Petri nets were presented by Carl Adam Petri in 1962 as a modeling tool to study parallel computation [5]. Recently, they have been used in a number of applications

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

including manufacturing processes, medical applications, office automation, formal languages and circuit design [6]. This list of applications of Petri nets is not exhaustive.

It is evident that teaching Petri nets in a computer science curriculum is almost inevitable. While upper division courses on topics such as Parallel and Distributed Systems, Concurrent Systems, and Computer Networking can certainly benefit from teaching Petri net as a modeling tool, courses on Information Systems can also use Petri nets. Graduate students can use Petri nets for analyzing models rather than simply modeling the system itself.

In this paper, the authors describe the design and implementation of a Petri net tool that is targeted primarily for teaching. It has a graphical editor with which the user can create a Petri net, save it and load it again. A tabular representation of the Petri net is also created that can be viewed separately.

Simple Petri Nets

A simple Petri net is represented as a triple $P_{net} = \langle P, T, C \rangle$ where P denotes a set of places, T denotes a set of transitions, and C is a set of directed arcs connecting places and transitions [4]. A place is a container that holds tokens. Each token in a place represents an item of significant information in the application domain such as a data element, an object or an entity. A transition represents a process that is fired when it is enabled. A transition is said to be enabled when the input places (discussed later) of the transition contain a predefined number of tokens. Hence, every transition is augmented with a set of conditions that enables the transition. These conditions are modeled as rules associated with a transition. For example, a rule for a transition may indicate that it should receive two tokens from its two input places, one token from each place, and then send one token to its output place. The arcs in C must connect places and transitions alternately so that no two places and no two transitions are connected directly by an arc. Hence, a Petri net is actually a bipartite graph. Accordingly, if a connection is directed from a place to a transition, the place is said to be an input place of the transition. If a connection is directed from a transition to a place, then the place is said to be an output place of the transition. Initially, every place may have some tokens. A net with no tokens in any of its places is called an *unmarked Petri net*.

Figure 1 shows a simple Petri net modeling the manufacturing process of a chair. Every token in the place P_1 represents a leg of the chair, a token in P_2 represents the seat, and a token in P_4 represents the back support of the chair. The rules for the transitions are indicated below:

- T_1 fires when there are four tokens in the input place P_1 , and one token in the input place P_2 . It puts one token into the output place P_3 after firing.
- T_2 fires when there is one token in the input place P_3 and one token in the input place P_4 . It puts one token into the output place P_5 after firing.

Thus, Figure 1 illustrates how a Petri net can be used to model the work flow in the manufacturing process of a chair.

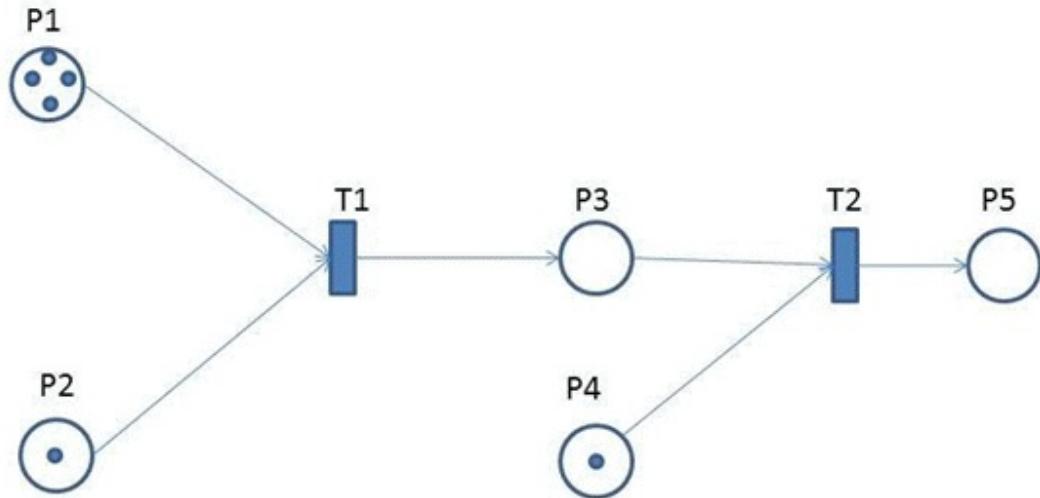


Figure 1: Representation of the manufacturing process of a chair.

Colored Petri Nets

A colored Petri net is an extension of a simple Petri net by adding colors to its components [2]. Using colors, one can distinguish the properties of different elements in the diagram. For example, the place P5 in Figure 1 can be assigned a different color in order to show that it is the final destination of all the tokens. It is up to the user to choose colors and their meanings in the diagram for a given application. In our work, we use colored tokens to distinguish different entities in the application. For example, the legs of the chair in Figure 1 may be represented by green colored tokens, while the seat can be represented by red colored tokens. When we use such colored tokens, the rules for the transitions will also be changed. Accordingly, the rule for transition T1 may be written as

T1 fires when there are four green colored tokens in the input place P1, and one red colored token in the input place P2. It puts one gray colored token into the output place P3 after firing.

Colored Petri nets are thus useful to quickly identify the movements of a particular entity throughout the diagram.

Timed Petri Nets

A timed Petri net is an extension of a simple Petri net by including time constraints [2]. One can add timing constraints to the transitions that can be used to fire them in synchronous time intervals. It is also possible to delay the movement of a token from an input place or to an output place in order to observe the departure or arrival of tokens at the respective places. In our tool, we have included the facilities to accommodate timing constraints in both places and transitions.

Colored and Timed Petri Nets

Since the purposes of using colors and timing constraints in a Petri net are relatively independent of each other, it is possible to use both extensions at the same time, leading to colored and timed Petri nets. Our tool supports such a hybrid approach as well.

THE DESIGN OF THE TOOL

The authors have developed a Petri net tool that is primarily targeted toward teaching Petri nets. However, the tool can also be used by researchers, modelers and others. There are two important goals we wished to achieve in developing this tool: modeling a Petri net and executing a Petri net.

Modeling a Petri Net

The first goal is achieved if the tool can provide sufficient mechanisms to draw a Petri net with different characteristics (simple, timed or colored). Like any design tool such as a UML tool, the Petri net tool provides the following functionalities:

- Create, edit, save and load a Petri net (simple, colored, timed, colored and timed).
- View the tabular representation of a Petri net.
- Add, delete and modify the rules of each transition.
- Execute a Petri net in a step-by-step manner, firing one (chosen) transition at a time.
- Execute a Petri net in auto-fire mode in which the transitions are fired either when they are enabled or synchronously when their respective timing constraints are satisfied.

The user interface of the tool is shown in Figure 2. The Petri net in this diagram models a vending machine that accepts 5? and 10? coins, represented by red colored and blue colored tokens, respectively. This example is a slight modification of the vending machine example given in [3]. The machine delivers a product for 15? (represented as a green colored token) and returns change, if any. The operations of this net can be briefly described as follows. Place 1 represents coin insertion into the vending machine; it accepts 5? and 10? coins (red and blue tokens, respectively). The transition T1 fires when there is at least one blue token in place 1, accumulating the 10? coins in place 2. Similarly, T2 fires when there is at least one red token, accumulating the 5? coins in place 3. A product can be purchased if (i) there is one red token in place 3 and one blue token in place 2, (ii) there are two blue tokens in place 2, or (iii) there are three red tokens in place 3. In case (i), transition T3 fires, resulting in a green token in place 4; the green token represents the product. In case (ii), transition T7 fires, leaving a green token in place 4 and a red token in place 3. The red token represents change in the amount of 5?. In case (iii), the transition T6 fires, resulting in a green token in place 4. Thus, the net accommodates all possible combinations of inputs to buy the product. In addition, place 6 represents the coin return button on the vending machine. When the user presses this button, the vending machine is supposed to return all coins in the machine. This is shown by firing the transition T4, which collects all 10? coins from place 2, if any, and returns

them in place 5. Also, the transition T5 is fired to collect all 5? coins from place 3, if any, and return them in place 5.

The tool supports basic verification of the diagram, which includes the following:

No element (place, transition or connection) can stand by itself without being connected to the rest of the diagram. It should be noted that this verification is done only at the time of creating a Petri net.

A connection cannot join two places or two transitions, thus enforcing the property that a Petri net is a bipartite graph.

A token can only be inserted inside a place.

The tabular representation of the Petri net also shown in Figure 2.

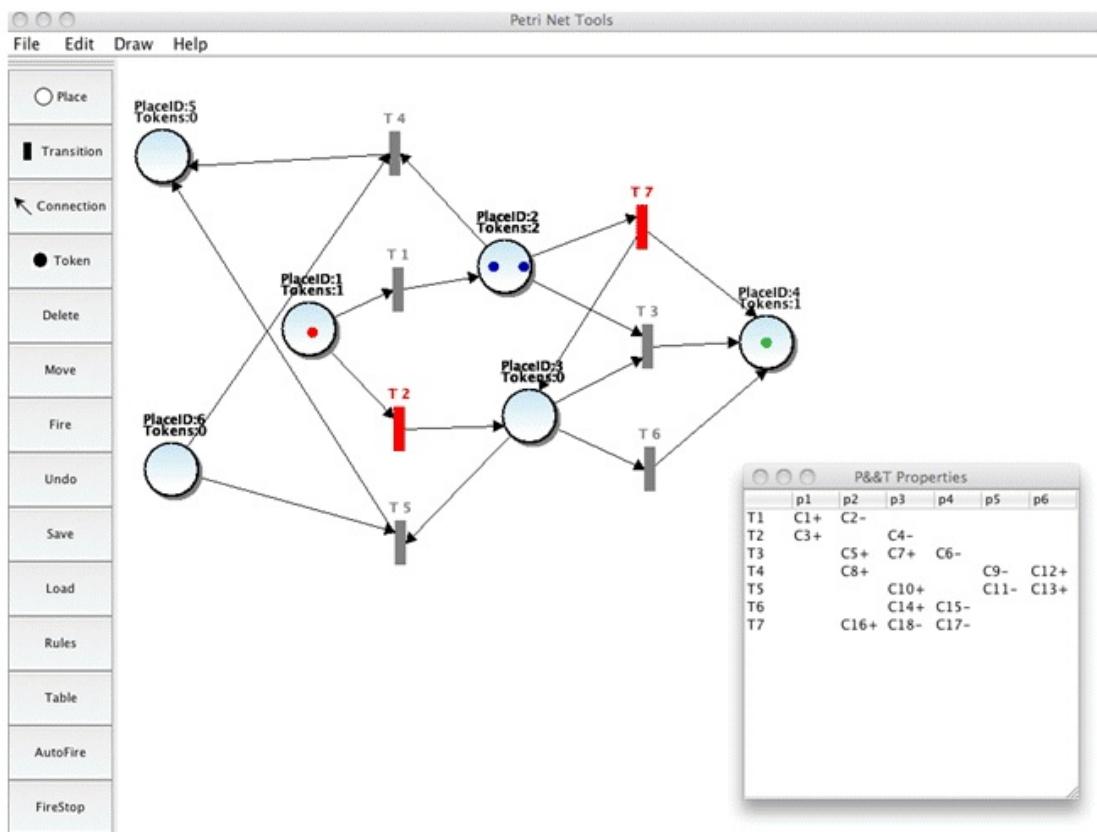


Figure 2: User interface for the tool

Every entry in this table represents a connection between a place and a transition. The '+' sign at the end of the connection label indicates a connection from a place to a transition, and a '-' at the end of the label indicates a connection from a transition to a place. The tabular representation is not only useful to quickly visualize a complex Petri net but can also be used for processing the information.



Figure 3: Rules for transitions T2 and T7 in Figure 2.

The rules editor is shown in Figure 3. As seen in the rules editor, a designer can choose different types of Petri nets - simple, colored, timed or hybrid (colored and timed).

Executing a Petri Net

Once a Petri net is drawn and verified for correctness, it can be executed manually, one step at a time, by firing a particular transition, or it can be executed synchronously by firing every transition that is enabled. Figure 2 shows one instance of synchronous firing in which the transitions T2 and T7 fire concurrently. It should be noted that a transition will fire only if it is enabled, irrespective of which mode of execution is selected. The left side palette on the user interface in Figure 2 shows the appropriate commands for both types of actions (Fire and AutoFire). As seen in Figure 2, a synchronous execution can be stopped at any time by choosing the FireStop button.

IMPLEMENTATION DETAILS

The Petri net tool was developed using Java and is portable to any platform. It has been designed in such a way that each entity such as place, transition, connection and token is modeled as a separate class. In addition, each rule for a transition is modeled as a separate class in order to keep it independent from its associated transition. The XMLEncoder and XMLDecoder classes in the Java API are used to store and retrieve the elements of a diagram. The set of rules for each transition is stored as an XML file. The XML representation is easier to manipulate and can be easily ported across platforms.

A major advantage of the tool presented here is the inclusion of the rule editor. The visual interface of the rule editor makes the creation and modification of the rules easier for anyone. Further, depending on the type of Petri net chosen (as seen by the selection tab on the top of the rule editor in Figure 3), the rules appropriate for the chosen Petri net type can be included. That is, the tabs provide a different interface for different sets of rules based on the type of Petri net.

In implementing the transitions, a separate Java thread is created for each transition so that several transitions can fire concurrently. Thus, concurrent modeling behavior can be observed by watching the concurrent thread executions (shown in red color in Figure 2). By including appropriate delays, one could watch how concurrent executions are serialized in a single processor environment. This visualization is further aided by the use of colored tokens in the colored Petri net model.

CONTINUING WORK

Modeling a Petri net is somewhat easier than executing a net because the main purpose is to draw the diagram with some built-in verification. However, executing a Petri net poses several challenges. Currently, the rules for a transition only allow the movement of tokens but do not support any computation. For example, in the vending machine example, it may be desired to include a rule that allows multiple options such as the following: Fire transition T3

- if there is one blue token in place 2 and one red token in place 3, and deposit one green token into place 4 OR
- if there are three red tokens in place 3, and deposit one green token into place 4 OR

- if there are two blue tokens in place 2, and deposit one green token into place 4 and one red token into place 3.

This rule would result in simplifying the three transitions T3, T6 and T7 in Figure 2 into a single transition. In this case, the rules editor should be modified to include expressions, not just selecting the components. Further, in many cases it is not known how complex the rule will be. Typically, the complexity of a rule will vary depending on the application. However, the tool must be sufficiently general to support all possible options. Currently, we have implemented most of the features that will be used for teaching Petri nets. We continue to work on the enhancements of the tool.

CONCLUSION

This paper describes the development of a Petri net tool that can be used for teaching Petri nets. The tool provides both modeling and execution of Petri nets. An extensive collection of Petri net tools is reported in [1]. We believe our tool is comparable to several of the simpler tools reported in that site. An interesting feature of our tool lies in the effectiveness and easy-to-use interface in introducing the rules for transitions. This interface does not require the user know any format to introduce the rules, thus making it user-friendly. We continue to work on enhancing the features of the tool so that it can be used for even more interesting purposes.

REFERENCES

- [1] Complete Overview of Petri Nets Tools Database, Nov 2010.
http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.html
- [2] David, R., Alla, H., *Discrete, Continuous and Hybrid Petri Nets*, Springer-Verlag Berlin, 2005.
- [3] Geholt, V., Sloane, E.B., Application of the Petri Net to Simulate and Validate Complex, Multi-Vendor, Heterogeneous Networks of Wireless Medical Device Systems, *Proceedings of the Tenth Americas Conference on Information Systems*, New York, 4631-4636, August 2004.
- [4] Girault, C., Valk, R., *Petri Nets for Systems Engineering*, Springer-Verlag Berlin, 2003.
- [5] Murata, T., Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, 77(4), 541- 580, April 1989.
- [6] Zurawski, R., Zhou, M., Petri Nets and Industrial Applications: A Tutorial, *IEEE Transactions on Industrial Electronics*, 41(6), 567-583, Dec 1994.

LESSONS LEARNED IN THE DESIGN OF AN UNDERGRADUATE DATA MINING COURSE*

Cecil Schmidt

Department of Computer Information Sciences

Washburn University

Topeka, KS 66621

785-670-1161

cecil.schmidt@washburn.edu

ABSTRACT

The ability to analyze and construct knowledge from data is a fundamental talent that is in high demand. Developing this ability requires the understanding of how to solve problems that are data oriented and continually evolving in both application and complexity, i.e. broadly speaking it is the understanding of data mining concepts and techniques. This suggests the need for an educational component at the undergraduate level that should provide the foundational background for solving such problems. These problems are multi-disciplinary by their nature, require both technical and non-technical expertise, and are wonderful resources for further research. This paper describes the design and delivery of such a course and the lessons learned in its design and delivery. Although the course content did not match the recommended curriculum, a set of interesting outcomes was produced. In particular students that completed this course created significant data mining research, and in some cases, was publication worthy.

1. INTRODUCTION

The ability to leverage and create knowledge from the data is a key ingredient towards the success of most any organization. Examples of knowledge creation range across all disciplines from the social and natural sciences to business. As noted by [9] "the convergence of computing and communication has produced a society that feeds on information". With such a broad spectrum of applications, it requires an educational

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

component that has the ability to attack problems from an overarching perspective. In April 2006, a data mining curriculum proposal was produced by [2] that identified a curriculum that would address this educational need.

Since the publication of the proposal in [2], a number of published papers have described data mining courses such as in [4] and [5], which are modeled after many of components that [2] identified. Other research describes courses that are directly related to data mining but do not specifically address the proposed curriculum model [3], [1]. The common threads among all of these publications is the understanding that data mining covers a broad field of applications and disciplines, can be taught a number of different ways, appeals to students that are both technical and non-technical, and the demand for such a course, or curriculum, is ever-increasing.

The objective of this paper is to describe the data-mining course that was taught in the fall semester of 2010 at an undergraduate masters-level institution. This paper is organized into three major components: the evolution of the course description; the course goals, their realization and a contrast between this course and a proposed curriculum model; and finally a discussion of the lessons learned.

2. EVOLUTION OF COURSE AND ITS DESCRIPTION

In spring of 2004 a special topics course in data mining was first created and delivered by the author. Special topics courses could be offered up to three times before going through formal course approval. Eleven students completed the initial course. Of those students, all were majors in either computer information science or computer information systems. The range of topics covered emphasized basic data mining concepts such as data representation, preparation and cleansing, and the applications of various data mining techniques. Additionally a formal research component was required for each of the students. The class meetings consisted of round-table discussions of the required reading for that week. For about the last third of the semester each student walked the class through their research. All of the students were very excited about what they learned from the course; course ratings were extremely positive across the board. Two of the students were so affected by their research that they went on to study data mining related disciplines in their graduate studies at level-one research institutions.

As an outcome of the special topics course, a new course was created, CM332, as part of the computer information sciences offerings at the author's university. At that time, it was thought that the course would run on an annual basis, alternating with a course in computational intelligence. Unfortunately, it was not until the fall of 2010 that this course could be offered again. Two key factors influenced the course offering. The first key factor was that for scheduling purposes. The department needed to offer a course that would meet similar curriculum requirements as that of our computational intelligence course, but cover a distinct set of topics. The second key factor was that it was supported by this institutions' Actuarial Advisory Board as an elective that would support the actuarial program currently offered through the department of mathematics and statistics. Provided below was the course description from our 2010-2011 course catalog.

CM332. Data Mining is the study of problem solving through the analysis of data. Topics include theoretical issues, input design, knowledge representation, and basic data mining algorithms including decision trees, statistical and linear models, and clustering techniques.

Prerequisites: CM307 Data Structures & Algorithmic Analysis & MA140 Statistics, or Consent of Instructor

The CM307 prerequisite implicitly required one course in discrete mathematics. CM307 was a course typically taken at the end of a student's sophomore year or first semester of junior year. MA140 was an introductory statistics course offered through the mathematics and statistics department. Consent was typically given only if the student had taken one course in Java, C++, or C#, and had the appropriate mathematical and statistical background.

The desired prerequisites aligned with two of the four prerequisites identified by the model curriculum proposal [2]. The remaining two, database systems and linear algebra, were not required.

3. COURSE GOALS AND THEIR REALIZATION

The course goals listed in Table 1 represent a significant subset of those typically identified for data mining course. The proposed curriculum model [2] does not specifically state a set of goals. These course goals were identified through the analysis of the content of two texts specifically related to data mining including a practical, tutorial-oriented professional book [9] and an applied textbook [6].

Table 1: Course Goals

Goal	Description
G1	Gain an understanding of what data mining is all about.
G2	Be able to perform the data preparation tasks and understand the implications.
G3	Demonstrate an understanding of the alternative knowledge representations such as rules, decision trees, decision tables, and Bayesian networks.
G4	Demonstrate an understanding of the basic machine learning algorithmic methods that support knowledge discovery.
G5	Be able to evaluate what has been learned through the application of the appropriate statistics.
G6	Be able to discuss alternative data mining implementations and what might be most appropriate for a given data mining task.
G7	Become proficient in the use of a set of data mining tools.

3.1 CONTENT COVERAGE

The proposed model curriculum [2] suggests two courses, a senior undergraduate or first-year graduate level foundations course and an advanced-topics course. CM332 matched up with the foundations course. Table 2 depicts the course content, the weeks during which it was covered, the related homework, required readings from [9], and the project. A tutorial-oriented, professional book [9] was chosen for its gentle introduction to the field of data mining and its direct link to an open-source data mining tool-kit [8].

Other readings and a comprehensive final exam were required but are not listed for brevity. Grades were based on homework (25%), final exam (25%), and research project (50%).

Table 2: Course Content

WK	Description	HWK	READ
1	What's it all about? Examples, fielded applications, generalization, and ethics.	HW1	CHP1
2	Input: Concepts, instances, and attributes; data preparation	HW2	CHP2
3,4	Output: Knowledge representation: decision tables and trees, classification rules, association rules, rules with exceptions, rules involving relations, trees for numeric prediction, instance-based representation, clusters.	HW3	CHP3
5-7	Algorithms - The basic methods: inferring rules, statistical modeling, constructing decision trees, constructing rules, mining association rules, linear models, instance based learning, clustering.	HW4	CHP4 CHP6
8	Weka machine learning workbench: an introduction and the explorer.	CHP9	CHP10
8,9	Credibility: Evaluating what's been learned - train and test, predicting performance, cross validation, leave-one-out, bootstrap, counting cost	PJ1	CHP5
10 11	Transformations: Engineering the input and output: attribute selection, discretization, bagging, boosting, and stacking.	HW5 PJ2	CHP7
12	Running experiments with Weka	HW6	CHP12
13 14 15	Research presentations.	PJ3	

Table 3: Proposed Model Curriculum Topic Coverage

Topic	Description	CM332
T1	Introduction [to data mining]	100%
T2	Data preprocessing	80%
T3	Data warehousing	0%
T4	Association, correlation, and frequent pattern analysis	40%
T5	Classification	100%
T6	Cluster and outlier analysis	50%
T7	Mining time-series and sequence data	40%
T8	Text mining and web mining	25%
T9	Visual data mining	10%
T10	Data mining: industry efforts and social impacts	50%

The foundations course described in [2] identified ten major content topics. Table 3 provides a high-level list of these topics and rough percentages of what CM332 covered. The focus of CM332 emphasized the application of the theoretical aspects of data mining. A significant amount of time was spent on the application of Weka in solving a real-world problem as part of an individual research project leaving less time for the suggested topics.

3.2 HOMEWORK

Homework assignments were made specifically to ensure that the students actually read the assigned material. Quizzes were given over the first three homework assignments in lieu of grading the entire assignment. HW4 required working through two separate 3-Fold cross-validation classification problems including naïve-bayes and 1-Nearest Neighbor. Students developed their solutions in a spread-sheet program. This assignment worked out particularly well in that the students easily saw that small changes to their classification description had a significant impact on the results. HW5 was a specific lab that required the students to use a number of the tools in Weka. Of most significance, the students were required to apply a cost model as part of a classification problem and determine which of two classification descriptions perform better based upon the cost model. Discussion ensued about the real cost of a false negative result such as a medical misdiagnosis. HW6 required the students to each develop a rubric that could be used to "classify" their research project. The "best" rubric was selected and was used as the grading mechanism for the final research presentation. This turned out to be a great way to illustrate how domain experts can lend their expertise to building a classifier.

3.3 RESEARCH PROJECT

A key component of CM332 was the course project. In order to manage the project it was divided into three parts guided by the knowledge discovery in databases (KDD) process as defined in [10]. A suggested starting point for datasets was at [7]. The following was the project description.

1. PJ1: Define the problem in terms of actionable goals and objectives. Describe the dataset(s) and list appropriate references (minimum of two peer-reviewed articles).
2. PJ2: Based upon your goal(s) for your study, review the current literature that is relevant. Write up a one or two page review of the most relevant paper regarding your research.
3. PJ3: Report your experimental findings in a PowerPoint presentation. Create the presentation such that it could be put together for a formal poster presentation.

The most significant outcome of the research project was how well these projects were done. Of the thirteen research projects presented, with minimal rework seven were poster ready. Of these seven, four of these projects were worthy of further refinement for submission as a conference paper.

4. DISCUSSION OF LESSONS LEARNED

The CM332 course fits best with the foundations course as described in [2]. Due to time constraints and CM332's applied emphasis, only a subset of the proposed curriculum could be covered. Depending upon the depth of coverage and student background, it is problematic to cover all of the identified topics. However the guidelines in [2] provide a standardized starting point for data mining courses, and that is significant. The findings of this paper suggest that the proposed model might be refined to include modules that subset the foundations course, but meet the minimum requirements for a first course in data mining. Although course pedagogy is not emphasized, the author has twice found that having a significant research component is a great way to stimulate student research and potential graduate study.

5. REFERENCES

- [1] Chawla, N. V., Teaching data mining by coalescing theory and applications. *Proc. 35th ASEE/IEEE Frontiers in Education Conference*, Indianapolis, IN, USA, 17-23, IEEE., 2005.
- [2] Chakrabarti, S. et al., *Data mining curriculum: A proposal (Version 1.0)*. Intensive Working Group of ACM SIGKDD Curriculum Committee, April 2006, retrieved December 1, 2010, from <http://www.sigkdd.org/curriculum/CURMay06.pdf>.
- [3] Christen, P., Evaluation of a graduate level data mining course with industry participants, *Proc. 6th Australasian Data Mining Conference (AusDM'07)*. Gold Cost Australia, pp. 233-241, CRPIT Vol. 70., 2007.

- [4] Ponelis, P., Finding diamonds in data: reflections on teaching data mining from the coal face. *Issues in Informing Science and Information Technology*, Volume 6., 2009.
- [5] Stewart, B., Reflection on development and delivery of a data mining unit, *Proc. 6th Australasian Data Mining Conference* (AusDM'07). Gold Cost Australia, pp. 225-232, CRPIT Vol. 70., 2007.
- [6] Tan, P., Steinbach, M., and Kumar, V., *Introduction to Data Mining*, Pearson Education, Inc., 2006.
- [7] UCI KDD Archive, Information and Computer Science, University of California, Irvine. <http://kdd.ics.uci.edu/>. Accessed September, 2010.
- [8] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten, The WEKA data mining software: an update; *SIGKDD Explorations*, Volume 11, Issue 1, 2009.
- [9] Witten, I., and Frank E. *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed., Morgan Kaufmann, 2005.
- [10] Zhang, S., and Zhang, C. Data preparation for data mining, *Applied Artificial Intelligence*, 17:375-381, 2003.

AN INTRODUCTION TO DISTRIBUTED VERSION CONTROL SYSTEMS*

CONFERENCE WORKSHOP

John Cigas

*Department of Computer Science, Information Systems, and Mathematics
Park University
Parkville, MO 64152
cigas@acm.org*

ABSTRACT

This workshop introduces participants to a distributed version control system (DVCS), such as *Mercurial* or *Bazaar*. A DVCS uses a peer-to-peer model as opposed to a client-server model. Two of the benefits of a DVCS are having a complete copy of the repository for each user and the decoupling of committing changes from publishing them. Participants will start with an established repository so they can focus on daily-use concepts, such as committing, publishing and merging. Using a web site as the initial project provides a familiar environment and allows viewing of all changes during the workshop exercises. With multiple participants, we will also create conflicts between the repositories and then see how to resolve them. As time permits, we will also cover issues such as permissions, web access, and different work flow possibilities.

This workshop is for anyone with an interest in distributed version control systems. Even those familiar with client-server systems, like CVS or Subversion, will find something useful. A laptop running Unix, OS X or Windows is required.

* Copyright is held by the author/owner.

CRITICAL THINKING AND MODELING IN CS0: THE PRISONER'S DILEMMA*

*David Reed
Department of Computer Science
Creighton University
davereed@creighton.edu*

ABSTRACT

Fluctuating computer science enrollments and the realization that many disciplines require basic computing knowledge are leading to an increased interest in non-majors (CS0) computing courses. Currently, there is no single dominant model of CS0 that simultaneously teaches the big ideas underlying computing, provides practical experience with programming, and makes computing relevant to a wide spectrum of students. What has become clear, however, is that meaningful, real-world applications are needed to drive student interest and learning. This paper describes one particular application, the Prisoner's Dilemma, that bridges game theory and evolutionary biology and provides a framework for critical thinking, modeling, and interdisciplinary problem solving within a CS0 course.

1 INTRODUCTION

Several factors in industry and academia have led to increased attention on non-majors (CS0) courses in computer science curricula. The drop in CS enrollments in the middle of the decade encouraged faculty to reconsider how they might reach out to non-traditional computer science students and perhaps interest them in taking additional computing courses. Demand for computing skills in other disciplines has encouraged interdisciplinary collaboration and has helped articulate the fact that many students need a broad understanding of computing, but not necessarily the programming depth typically found in a CS1 course. In recent years, articles at computer science education conferences (e.g., [1][2]) have emphasized the importance of critical thinking for non-majors. User-friendly environments, such as Alice, Scratch, and Greenfoot, are

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

increasingly being adopted to engage students and teach basic programming/computing concepts [3]. Likewise, the College Board and National Science Foundation are currently funding the development of a new first course in computing, CS Principles, that may eventually be adopted as an Advanced Placement course in high schools [4][5].

While most CS educators agree that a broad and engaging introduction to computing concepts is needed, it is still not entirely clear how to design a course that contains meaningful programming (since programming is central to so many computing concepts) while still focusing on broader themes (such as critical thinking, real-world applications of computing, and societal impact). The CS Principles project is currently in the pilot phase, where instructors at a handful of institutions are developing courses using a variety of pedagogies and tools in order to determine best practices. Central to the success of these and other balanced CS0 courses is the identification and integration of applications that are relevant to today's students, that encourage critical thinking and analysis, and that demonstrate the power of basic programming in solving interesting problems.

Since 2005, the computer science program at Creighton University has offered a popular CS0 course that balances the breadth of computing with practical programming depth using JavaScript [6][7]. The course, Computers and Scientific Thinking, meets the science core requirement for the College of Arts & Sciences and thus reaches a wide spectrum of students. One application that has been used successfully in this course to integrate critical thinking, modeling, and programming is the Prisoner's Dilemma.

2 THE PRISONER'S DILEMMA

The Prisoner's Dilemma is a classic game theory scenario first formulated by Merrill Flood and Melvin Dresher in 1950. The following is a slight variant of the description found on Wikipedia [8]:

Two suspects are arrested by the police. The police have insufficient evidence for a conviction, and, having separated the prisoners, visit each of them to offer the same deal. If one testifies for the prosecution against the other (defects) and the other remains silent (cooperates), the defector goes free and the silent accomplice receives the full five-year sentence. If both remain silent, both prisoners are sentenced to only one year in jail for a minor charge. If each betrays the other, each receives a three-year sentence. Each prisoner must choose to betray the other or to remain silent. Each one is assured that the other would not know about the betrayal before the end of the investigation. How should the prisoners act?

While this colorful story has a long history, it raises concerns both for its lack of relevancy and its questionable ethics. Hopefully, very few students will ever be in such a situation and the ethical concerns over crime and betrayal cloud the point of the scenario, which is to rationally determine which action is in a person's best interest. As such, an alternative formulation of the Prisoner's Dilemma can be used, which taps directly into their own sense of self interest.

An instructor is currently teaching two sections of a course and is offering bonus credit on the final exam to each section. As a single body, each section of students must decide whether to cooperate with the other section or defect.

If one section defects and the other cooperates, then each student in the defecting section will receive five bonus points on his/her final exam grade, while students in the cooperating section receive nothing. If both sections cooperate, students in both sections will receive three bonus points. If both sections defect, then everyone receives one bonus point. The sections are not allowed to communicate in any way. How should each section act?

This version of the scenario is accessible to students and never fails to raise heated discussion. Some students immediately recognize the incentive to defect, while others cling to the desire for cooperation and mutual benefit. Many are surprised by the argument that defecting is the only rational action to take. Since one section's choice does not affect the other's, it follows that defecting will always yield the better result: if the other section cooperated, then defecting will yield five points while cooperating would only yield 3; if the other section defected, then defecting will yield one point while cooperating would yield none. The fact that logic leads each section to choose to defect and receive fewer points than if they had both cooperated is what makes this scenario a dilemma.

An even more interesting variant of the Prisoner's Dilemma is the Iterated Prisoner's Dilemma, where the act of cooperating or defecting occurs repeatedly. In the essay *The Prisoner's Dilemma and the Evolution of Cooperation*, Douglas Hofstadter describes the Iterated Prisoner's Dilemma in terms of a diamond broker and a buyer who arrange to secretly drop off money and diamonds simultaneously each month at agreed upon locations [9]. Again, the background story raises ethical concerns that are best avoided in a classroom setting. Instead, the bonus credit version can be extended to capture this notion of repeated transactions:

Suppose at the beginning of the course the instructor announced that he/she would be making this same bonus credit deal on every quiz, test and assignment in the course. Knowing that the opportunity to earn bonus points will be repeated over and over throughout the course, how should each section act each time?

Under this scenario, it is in the best interest of the students to foster trust and cooperation, since receiving three bonus points on every assignment is to their advantage. The five bonus points that could be gained by defecting are offset by the likelihood of retaliation on the next assignment, which could quickly devolve in each section repeatedly defecting and earning only one bonus point. During discussions, students recognize this distinction from the one-shot Prisoner's Dilemma, and quickly propose strategies that reward cooperation and hopefully lead to both sections repeatedly cooperating for the common good.

3 CRITICAL THINKING AND MODELING

While discussions of the Prisoner's Dilemma and Iterated Prisoner's Dilemma force the students to think critically about the consequences of actions in each scenario, they also lead naturally into the use of computers to model the transactions. A computer model of the Iterated Prisoner's Dilemma allows the student to test his or her understanding of the process, to form hypotheses as to how each strategy would interact

with other strategies, and thus determine which strategies would perform the best overall. Initially, students consider four basic strategies for acting in the Iterated Prisoner's Dilemma:

<i>coop</i>	always cooperates, no matter what the opponent strategy does
<i>defect</i>	always defects, no matter what the opponent strategy does
<i>randomness</i>	chooses randomly between cooperating and defecting
<i>tit-for-tat</i>	cooperates in the first round, but then does whatever the opponent strategy did in the previous round

In a set of exercises, students are asked to predict how different strategies would fare against each other. For example, suppose *coop* and *defect* were to interact 50 times - how many points would each strategy earn? In this extreme case, *defect* would earn five points each time for a total of 250 points, while *coop* would earn no points. However, *coop* and *tit-for-tat* would both earn three points each time for a total of 150 points. Once student have made their predictions, they are provided with an interactive Web page that enables them to test those predictions (Figure 1). As a result, they experience the use of a program as a tool in the scientific process of studying a system, forming a hypothesis, and conducting an experiment to test that hypothesis.

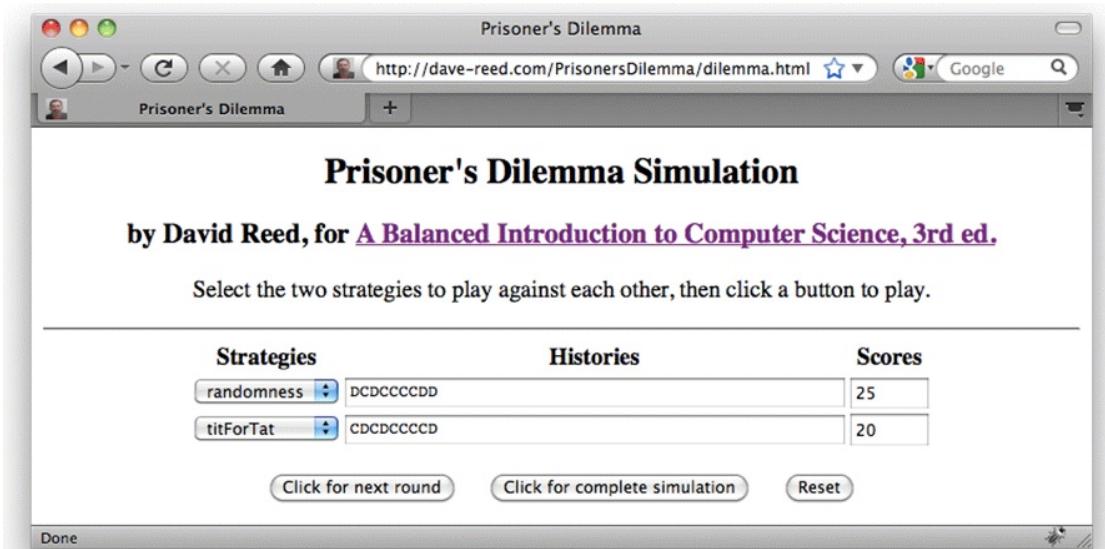


Figure 1: Screen shot of a Web page for modeling Iterated Prisoner's Dilemma transactions.

4 INTEGRATING COMPUTING ACROSS DISCIPLINES

In his classic book, *The Evolution of Cooperation*, Robert Axelrod uses the Prisoner's Dilemma as a metaphor for how cooperative behavior can evolve in nature [10]. In nature, some creatures are passive (e.g., a dove), others are aggressive (e.g., a hawk), and many are a hybrid of the two (e.g., a sparrow, which rarely attacks other birds but will fight if provoked). There are many instances in the natural world where weaker creatures will work together to defend themselves from a stronger predator, such as a pair of sparrows that will team up to drive a hawk from their nesting area. By associating Prisoner's Dilemma strategies with creatures in nature, the Iterated Prisoner's Dilemma provides a model for understanding how such cooperation might evolve.

For example, we may associate the aggressive *defect* strategy with a hawk and the hybrid *tit-for-tat* strategy with a sparrow. By simulating Iterated Prisoner's Dilemma interactions between the two strategies, for say 50 rounds, we can draw parallels with how they might interact in nature. In this case, the hawk (*defect*) would earn a total of 54 points while the sparrow (*tit-for-tat*) would earn only 49. That is due to the fact that the hawk earns five points in the first round (due to the sparrow's initial cooperation), then each bird earns one point in the subsequent 49 rounds as they mutually defect. Thus, our model would conclude that the hawk defeats the sparrow in a close battle. However, if we add a second sparrow to the mix and total the points that each bird receives in its interactions with the other two, a different result is achieved. While the hawk defeats each sparrow in its individual confrontations (54 vs. 49 points), the sparrows both earn 150 points when they interact with each other due to 50 rounds of mutual cooperation. The final result is 199 points for each sparrow and only 108 points for the hawk. In a sense, the hawk wins all of its battles but loses the war with the sparrows. The cooperative nature of the sparrows gives them a survival advantage as long as there are other sparrows around to cooperate with.

	hawk (defect)	sparrow1 (tit-for-tat)	sparrow2 (tit-for-tat)	TOTAL POINTS
hawk (defect)		54	54	108
sparrow1(tit-for-tat)	49		150	199
sparrow2 (tit-for-tat)	49	150		199

Figure 2. Interactions between a hawk and two sparrows.

This model of natural behavior opens numerous possibilities for exploration and experimentation. For example, throwing a dove (coop) into the mix shifts the balance of power back in the hawk's favor. If there is a passive dove for the hawk to exploit, the hawk will gain strength from its interaction with that dove. As Figure 3 shows, in an interaction between a hawk, two sparrows and a dove, the hawk comes out victorious - earning 358 points as opposed to 349 for the sparrows and 300 for the dove. While the sparrows and dove do cooperate with each other (earning 150 points each time), those points cannot offset the 250 earned by the hawk when it interacts with the dove.

	hawk (defect)	sparrow1 (tit-for-tat)	sparrow2 (tit-for-tat)	dove (coop)	TOTAL POINTS
hawk (defect)		54	54	250	358
sparrow1(tit-for-tat)	49		150	150	349
sparrow2 (tit-for-tat)	49	150		150	349
dove (coop)	0	150	150		300

Figure 3. Interactions between a hawk, two sparrows, and a dove.

	hawk (defect)	sparrow1 (tit-for-tat)	sparrow2 (tit-for-tat)	dove (coop)	sparrow3 (tit-for-tat)	TOTAL POINTS
hawk (defect)		54	54	250	54	412
sparrow1 (tit-for-tat)	49		150	150	150	499
sparrow2 (tit-for-tat)	49	150		150	150	499
dove (coop)	0	150	150		150	450
sparrow3 (tit-for-tat)	49	150	150	150		499

Figure 4. Interactions between a hawk, three sparrows, and a dove.

Interestingly, while two sparrows are not enough to protect the dove from the hawk, three sparrows suffice. As Figure 4 shows, by having three sparrows cooperating with each other and the dove, all of those birds earn more points than the lone hawk.

Examples such as these provide opportunities for students to experience computer programs as models of real-world systems and as tools for solving problems. They are able to form hypotheses, design experiments, and carry out those experiments using the program as a tool. They are also able to make the connection from computing to other disciplines, in this case the use of computer models to study evolutionary behavior in nature.

5 MODELING STRATEGIES AS FUNCTIONS

The activities and explorations described so far could be conducted in any CS0 course, regardless of the amount or type of programming taught in that course. To fully experience the power of this and similar applications, however, a student needs to see "under the hood" of the model. That is, a student who has learned the basics of programming in some language will benefit greatly from seeing how that language is used to create or modify the model. In this way, the student is empowered by knowing that they are capable, or could become capable with more practice, of developing such a model to solve real problems.

The Iterated Prisoner's Dilemma simulator (Figure 1) that is used in the CS0 course at Creighton University is written in HTML and JavaScript, the languages that the students are taught over the course of the semester. Once they have utilized the simulator as a tool for answering questions, students are able to view the source code and see how the individual strategies are encoded as JavaScript functions. Figure 5 shows the definition of the *tit-for-tat* strategy, which cooperates in the first round (when the history strings are empty) and subsequently acts however the opponent strategy did in the last round.

```

function titForTat(myHistory, yourHistory)
// Assumes: myHistory and yourHistory are strings of C's and D's
// (same length)
// Returns: whatever opponent did last (last character in
yourHistory)
{
    if (yourHistory == '') {
        return 'C';
    }
    else {
        return yourHistory.charAt(yourHistory.length-1);
    }
}

```

Figure 5. JavaScript function that implements the tit-for-tat strategy.

Once the students see how their programming skills can be applied to this problem, there are many opportunities for exploration. The instructor may introduce new strategies, such as a *tit-for-2-tats* strategy that requires two consecutive defections from the opponent strategy before it will retaliate, or a *random-tat* strategy that randomly defects 10% of the time and behaves as *tit-for-tat* the remaining 90%. Students must implement these new strategies as JavaScript functions, then integrate them into the simulator and perform experiments to characterize the performance of that strategy. In some years, students have been asked to design their own strategies from scratch and then run a class-wide tournament where all of the student strategies compete head-to-head. This activity is always popular with students and serves to integrate the programming and problem-solving skills that the students have developed within an interdisciplinary application.

6 CONCLUSION

The Prisoner's Dilemma, and the Iterated Prisoner's Dilemma variant, provide a rich framework for encouraging CS0 students to think critically, explore and experiment with computer models, and apply programming skills to interdisciplinary applications. As new models for CS0 courses are developed, there will be increasing demand for similar applications that integrate all of these characteristics and serve to engage and motivate students. While the particular implementation of the Prisoner's Dilemma simulator described in this paper utilized JavaScript, it could easily be adapted to other CS0-appropriate programming languages.

REFERENCES

- [1] Astrachan, Owen, Kathleen Haynie, Chris Stephenson, Lien Diaz and Amy Briggs. The Present and Future of Computational Thinking, *Proceedings of the 40th ACM Technical Symposium on Computer Science Education, ACM SIGCSE Bulletin* **41**(1), 2009.
- [2] Qualls, Jake A. and Linda B. Sherrell. Why Computational Thinking should be Integrated into the Curriculum, *Journal of Computing Sciences in Colleges*, **25**(5), 2010.

- [3] Utting, Ian, Stephen Cooper, Michael Kolling, John Maloney and Mitchel Resnick. Alice, Greenfoot and Scratch - A Discussion, *Transactions on Computing Education* **10**(4), 2010.
- [4] *CS Principles: A New First Course in Computing*, accessible online at <http://csprinciples.org>.
- [5] Astrachan, Owen, Susanne Hambrusch, Joan Peckham and Amber Settle. Re-imagining the First Year of Computing, *Proceedings of the 41st ACM Technical Symposium on Computer Science Education, ACM SIGCSE Bulletin* **42**(1), 2010.
- [6] Reed, David. *CSC 121: Computers and Scientific Thinking*, accessible online at <http://dave-reed.com/csc121>.
- [7] Reed, David. *A Balanced Introduction to Computer Science*, 3rd edition, Prentice Hall, New York, 2011.
- [8] *Prisoner's Dilemma*, Wikipedia: the Free Encyclopedia, accessible online at http://en.wikipedia.org/wiki/Prisoner's_dilemma.
- [9] Hofstadter, Douglas R., *Metamagical Themas: Questing for the Essence of Mind and Pattern*. Basic Books, New York, 1985.
- [10] Poundstone, William. *Prisoner's Dilemma: John von Neumann, Game Theory, and the Puzzle of the Atom Bomb*. Bantam Doubleday Dell Publishing Group, New York, 1992.

REQUIRING OUTREACH FROM A CS0-LEVEL ROBOTICS COURSE *

Janice L. Pearce
Department of Mathematics and Computer Science
Berea College
Berea, KY 40404
859 985-3569
pearcej@berea.edu

ABSTRACT

Since Fall 2006, Berea College has offered a CS0-level course entitled Introduction to Robotics. This course is one course among many which students can choose to satisfy a practical reasoning requirement in our institution's general education program. In each of the last two years of course offerings, the course has included a required outreach assignment in which students demonstrate the robots and their programming to children in the larger community. The positive college student response to this added outreach requirement is explored in this article.

NATIONAL CONTEXT

According to the National Science Board, for nearly two decades about one-third of all U.S. college freshmen have indicated plans to pursue a degree in science and engineering fields. They cite data which shows that the number of undergraduate degrees awarded by U.S. colleges and universities has been increasing over the past two decades both inside and outside of science and engineering fields. But, as all educators in computer science know, enrollments in computer science have followed a boom-bust cycle since 1998. In the field of computer science, the number of bachelor's degrees increased sharply from 1998 to 2004 and have continued to drop since that time [9].

Although computer science bachelor's degrees have continued to decline nationally, some good news seems to be on the national horizon. According to the Computing

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Research association (CRA), the number of new students in the U.S. majoring in computer science increased in both 2008 and 2009. The CRA indicates that the total enrollment among U.S. majors and pre-majors in computer science increased 4.2 percent [18]. The National Science Board finds that freshman intentions are a leading indicator of the future degree distribution, so one would expect bachelor's production in the field of computer science to begin rising in a few years when these students graduate [9].

LOCAL CONTEXT

Because of Berea College's unusual institutional mission, nearly 100% of our student body is economically disadvantaged, so very few of our students come from high schools which offer courses in computer science. It is for this reason, that we began offering a CS0-level offering in 1996. This original CS0-level offering was focused on HTML and JavaScript. It has been updated several times, and we continue to offer it as a CS0-level course today.

From 2001 to 2006, our CS1 enrollments remained fairly flat, so we made a departmental decision to expand our CS0-level offerings with the hope of attracting more students into computer science. In Fall 2006, we first offered the CS0-level Introduction to Robotics, and in January 2007, we first offered a CS0-level course on 3D animation using the Alice programming language. By expanding our CS0 course offerings, we were able to more than double our CS1 enrollments in 2007. Thus, it was in 2007 during the dismal decline in national CS bachelor production that our faculty unanimously approved a new major in Computer and Information Science [11].

Since its inception in 2006, the Introduction to Robotics course has served a number of purposes. It is one course among many which students can choose to satisfy a practical reasoning requirement in our institution's general education program, and it is one course among four which mathematics majors can choose to satisfy a collateral computing requirement in the mathematics major. Of course, its primary purpose, from a departmental perspective, is to interest students in taking additional computer science courses.

As mandated by our institutional mission, essentially all Berea students are economically disadvantaged and are drawn from a very wide geographic area. With a student body comprised of 25% ethnic minorities, our student population is racially much more diverse than our surrounding community, which is 93.2% white [16]. Thus, the addition of the outreach component to the Introduction to Robotics course was intended primarily to add educational value to the course rather than to attempt to directly increase or to diversify the computer science pipeline into our institution.

RELATED WORK

Computer science programs in many large research universities often employ outreach to the larger community, and many of these initiatives also include the laudable additional goal of increasing diversity in computer science. Programs such as Purdue's place computer science within a broader science, engineering, and mathematics context [12] while university outreach programs focused squarely on computer science outreach

have also proliferated. Outreach from Carnegie Mellon and the University of Illinois exemplify two of many such fine efforts [4, 15]. The University of Minnesota and Northern Kentucky University number among many institutions which use robotics in outreach through summer camps [3, 7]. At Duke University, undergraduates mentor teams of middle and high school students on robotics projects which is carried out co-curricularly rather than as a component of a course [10]. Dodds and Karp point out some of the difficulties involved in transferring the kind of outreach programs possible from a university to a small college environment because of potentially considerable differences in financial resources as well as in human capital [6].

Outreach from smaller institutions is often embodied in service-learning components of upper-level courses such as database systems or software engineering [14]. Even at larger institutions, the outreach which occurs in the area of robotics is frequently carried out co-curricularly or from upper-level robotics courses. Like other many institutions, Berea College defines service-learning to be an educational experience based upon a collaborative partnership between college and community which meets genuine community needs. In addition, a course which satisfies a service-learning requirement in Berea's general education program must have service-learning as a substantial portion of the course. At Virginia Commonwealth University (VCU), service-learning has been incorporated into an upper-level robotics course taught under an engineering rubric. In this course VCU students work with students from local high schools to design and construct a mobile robot which the high school students take to compete in the FIRST Competition [8]. Because of the needs of our student population, we do not believe that substantive outreach as a service-learning component would fit well into our Introduction to Robotics course as it is currently conceived. Instead, we have chosen to include a more modest outreach component in the course.

Relatively little has been written about outreach undertaken from CS0-level computer science courses. The Artbotics program at University of Massachusetts Lowell is an unusual collaboration between artists and computer scientists, taught at the CS0 level, and culminating in outreach through public exhibitions at a local museum [17]. At the University of North Carolina at Asheville, instructors have built outreach into their CS0-level technology course. In this course, college students work with local middle school students over multiple class sessions, and the service project in the course has changed over time from helping the children develop an algorithm for creating a drawing using a robot [1] to having the children create mazes to be fabricated by the university students [13].

Researchers have recently begun looking at the efficacy of outreach programs in computer science. For example, researchers at Indiana University used attitudinal surveys to study the outcome of their outreach roadshow which is designed to address stereotypical attitudes of high-school students towards computing [5]. However, most studies of outreach study the increase in interest or the change in attitude of those to whom the outreach is directed. Far fewer have looked at the impact on or the responses from those doing the outreach. The latter is the purpose of this paper.

GOALS AND METHODOLOGY

Many studies have shown that active learning as a model of instruction is superior to transmission models in which students are more passive receivers of knowledge [2]. One well-known instructional model which uses this perspective is the five step learning cycle: engagement, exploration, explanation, elaboration, and evaluation.

The primary course goal desired from these outreach experiences is to include a natural way to engage the college students in more components of the five step learning cycle. In particular:

- To reinforce course concepts by revisiting past labs while considering the concepts for use in a new application and for a different audience.
- To engage students in consideration and discussion of how to best explain the robotic and programming concepts to non-programmers.
- To create a structure in which students reflect on their own learning.

In both outreach experiences from the CS0-level Introduction to Robotics course at Berea, students worked in teams in class to develop programs in advance of the outreach. However, in 2010 additional reading assignments were given to help students better understand the needs of children and the purposes of outreach.

In 2009, students in the course demonstrated the robots at a local childcare center. These demonstrations were held during the class period on a regular class day. The college students worked in teams of three students to choose sensors and to design interactive programs which they could describe in anthropomorphic terms to children ranging in ages from 2 to 4 years old. During the demonstration, the college students sat on the floor in their teams. Teachers at the childcare center rotated the children from one student team to another team every 5-10 minutes.

In 2010, students in the course demonstrated the robots at the local public library on two non-consecutive Saturdays. The targeted age groups were pre-teens aged 10 and older on one Saturday, and children under 10 on the other Saturday. The college students worked in teams of four students to choose sensors and to design interactive programs which they could describe in anthropomorphic terms to these age groups. In order to allow the college students some flexibility, the teams were required to send only two of the four college students to each of the two Saturday outreach events. The basic set-up was similar, with the college students on the floor in their teams and interacting with the children.

RESULTS

Following each of these experiences, the college students were surveyed. In both years, the response rates exceeded 85%, and students overwhelmingly reported that the experience was worthwhile. In each year, only a single respondent reported that they did not believe it was sufficiently worthwhile to be repeated in a future offering. This means that 95% of the responding students in these two years self-reported that the outreach was worthwhile.

Though in both years, the vast majority of students reported that the experience was worthwhile, the responses were markedly less enthusiastic in 2009 than in 2010. Students

were asked to identify the most disappointing or frustrating part of the experience using an open ended format, and these responses were grouped through category analysis. In 2009, all of the respondents were able to identify ways to improve the experience. Some students identified more than one challenge, so the results do not sum to 100% (See Table 1.)

Table 1. Challenges reported by the more than one 2009 student

needed more preparation time/ better organization	56%
challenges in understanding/interacting with the age group	39%
difficulties moving sensors and programs to a new environment	22%

Following the 2009 offering, these student comments were used to improve the course structure: giving students more in-class preparation time prior to the outreach events, adding related out of class readings, and spreading the outreach over multiple days. In addition, the shift to the public library setting was motivated by the desire for finding an older group of children to reach.

In 2010, 20 of the 23 students responded, giving an 87% response rate. In 2010, the single student who reported that the experience was not worthwhile categorized the experience as fun but did not see the outreach as a learning experience. Organizational challenges seem to have been addressed in 2010 via the changes made in the course, but the shift to an older groups of children apparently created a new challenge, namely that of communication (See Table 2). Surprisingly, only a single student expressed dismay at having been required to dedicate a Saturday afternoon to a course assignment.

Table 2. Challenges reported by more than one 2010 student

difficulties moving sensors and programs to a new environment	40%
communication challenges with child, parent, or partner	35%

When asked what they saw as a stated learning goal and how well this learning goal was achieved, the responses from the students in 2010 were grouped into three areas using category analysis. All 2010 respondents reported that this goal was achieved, though the amounts varied from "somewhat" to "very well", with the majority of students indicating the goals were achieved "very well". Some students identified more than one stated learning goal, so again the results do not sum to 100% (See Table 3.)

Table 3. College student identification of primary learning goal achieved

to interest children in robotics and computer science	65%
to explain programming to non-programmers	30%
to interact with children in a positive way	22%

CONCLUSIONS

The primary goals of the Introduction to Robotics course are to interest and to engage the college students so they consider enrolling in additional computer science courses. So, the outcome of having 95% of the students report that they found the

outreach experience to be worthwhile, particularly when it took place on a Saturday afternoon, seems a direct and significant result.

It would be problematic to formally correlate continuation into C++ to a single assignment in the Introduction to Robotics course. Nevertheless, the inclusion of the outreach component is the most significant change which has been made to the course content in the last two years, so it seems worthwhile at least commenting on the increases in the continuance rate. In 2006, the first year of the Introduction to Robotics, more than a third of the students in the course went on to take C++ in the next term. Over the next two years, the percentage of students continuing from Introduction to Robotics into the C++ course in the following term increased over the first year, but then remained constant. In 2009, a modest, but perhaps not significant, increase was realized. Then in 2010, the increase in the percentage of students who elected to continue on to C++ was truly dramatic (See Table 4).

Table 4. Percentage of students in Introduction to Robotics continuing to C++

year	Robotics to CS1
2006	35%
2007	41%
2008	41%
2009	46%
2010	70%

Though our C++ enrollments had been almost flat for four years, our current C++ offering has seen a 53% enrollment increase, only half of which are students continuing from Introduction to Robotics. Clearly this remarkable enrollment increase is far in excess of the 4.2% of increased computer science enrollments seen nationally by the CRA [18]. We fully intend to explore this phenomenon further to gain a better understanding of it, but in the meantime we are pleased to know that our students are gaining interest in computer science.

Since many CS0 programs are developed in an attempt to interest students in computer science, it is hoped that the lessons learned in this experiment in outreach in a CS0 course at a small college will encourage others to consider incorporating outreach in their courses.

ACKNOWLEDGEMENTS

The author gratefully acknowledges:

- o The Lego, iRobot, and Sony Corporations for the Mindstorm, Create, and AIBO robotic platforms and well as to the Center for Distributed Robotics at the University of Minnesota for their development work on the Scout and eROSI robots.
- o Carnegie Mellon for the development of the RoboLab programming language.
- o The National Science Foundation for the support received through the following grants: #0420836, #0529529, #0511304, and #0531859.

REFERENCES

- [1] Bruce, R., Reiser, S., Introduction to technology for general education, ACM-SE 43: *Proceedings of the 43rd annual Southeast regional conference*, 43(1), 2005.
- [2] Bybee, R.W., *Achieving scientific literacy: from purposes to practices*, Heinemann Publishing, Portsmouth, NH , 1997.
- [3] Cannon, K., LaPoint, M., Bird, N., Panciera, K., Harini Veeraraghavan, N. P., and Gini, M., No fear: University of Minnesota robotics day camp introduces local youth to hands-on technologies. *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, 363-368, 2006.
- [4] Carnegie Mellon University Women@SCS. Women in computer science: Carnegie Mellon's women@SCS Outreach roadshow
<http://women.cs.cmu.edu/What/Outreach/Roadshow/> retrieved November 23, 2010.
- [5] Cottam, J., Foley, S. and Menzel, S., Do Roadshows Work? Examining the effectiveness of Just Be, *SIGCSE Bulletin*, 41(1), 17-21, 2010.
- [6] Dodds, Z. and Karp, L. The evolution of a computational outreach program to secondary school students, *SIGCSE Bulletin*, 38(1), 448-452, 2006.
- [7] Fox, R., Newell, G., and Frank, C. Experiences hosting high school summer camps to promote university outreach. *Proceedings of the 2nd Mid-south CCSC*, (April 2004), ACM Press, 166-172, 2004.
- [8] Hobson, R. S., The changing face of classroom instructional methods: service learning and design in a robotics course, *Proceedings of the Frontiers in Education Conference*, 2, F3C20-F3C25, 2000.
- [9] National Science Board, Science and engineering indicators: 2010,
<http://www.nsf.gov/statistics/seind10/pdf/seind10.pdf> retrieved November 23, 2010.
- [10] Osborne, R. B. Thomas, A. J. and Forbes J. R. N., Teaching with robots: a service-learning approach to mentor training, *SIGCSE Bulletin*, 41(1), education, 172-176, 2010.
- [11] Pearce, J. Nakazawa, M., The funnel that grew our CIS major in the CS desert, *SIGCSE Bulletin*, 40(1), 503-507, 2008.
- [12] Purdue University College of Science, Computer Science K-12 Outreach,
http://www.cs.purdue.edu/external_relations/k-12_outreach/ retrieved November 23, 2010.
- [13] Reiser, S. L. and Bruce R. F., Fabrication: a tangible link between computer science and creativity, *SIGCSE Bulletin*, 41(1), 382-386, 2009.
- [14] Sanderson, P. Vollmar, K., A primer for applying service learning to computer science, *SIGCSE Bulletin*, 32(1), 222-226, 2000.

- [15] University of Illinois at Urbana-Champaign Department of Computer Science, Outreach and diversity, <http://cs.illinois.edu/outreach> retrieved November 23, 2010.
- [16] U.S. Census Bureau, State and County Quickfacts, Madison County, KY, <http://quickfacts.census.gov/qfd/states/21/21151.html> retrieved December 13, 2010.
- [17] Yanco, H., Kim, H., Martin, F., Silka, L., Artbotics: combining art and robotics to broaden participation in computing, *Proceedings of the AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*, Palo Alto, CA, 2007.
- [18] Zweben, S., Computing degree and enrollment trends: From the 2008-2009 CRA Taulbee Survey, http://www.cra.org/govaffairs/blog/wp-content/uploads/2010/03/CRA_Taulbee-2010-ComputingDegreeandEnrollmentTrends.pdf retrieved Nov

A NEEDS AND REQUIREMENTS ELICITATION SIMULATION*

NIFTY ASSIGNMENT

*Edward J. Mirielli
Computer Science Department
Westminster College
Fulton, MO 65251
ed.mirielli@westminster-mo.edu*

Integrating theoretical and practical knowledge is essential for the developing software engineering professional. And, discriminating between "needs" and "requirements" is one such domain. Gaining experience and confidence using observational techniques to conduct needs and requirements elicitation during the analysis phase for software engineering projects is a necessary exercise in any system analysis and design or software engineering course. Elicitation is all about determining the needs of stakeholders and discovering what the user wants; it is considered one of the most critical activities in the software life cycle [1].

This nifty-assignment builds on a project-centered perspective where teams engage in needs and requirements elicitation sessions as both "designer" and "client" via a 4-stage role playing simulation. Each stage is allocated approximately 30 minutes, but time intervals can be adjusted as required. This simulation exercise is conducted after student teams become knowledgeable of their assigned software or system development project mission and vision. They take on the roles of both "designer" and "client" and engage other teams in order to better comprehend and demonstrate how needs and requirements elicitation is conducted. The outcome of this activity is that teams are better prepared when meeting with their real client thus producing more consistent and focused system requirement specifications.

Stage I of the simulation focuses on review of the various observation techniques used in needs and requirements elicitation. The material reviewed is provided by the instructor and corresponds with the course texts and other resources. Teams strategize and develop a plan for how they would conduct a needs and requirements elicitation for their own assigned project. In other words, each team has to imagine how they would go about the elicitation process using their project as a basis and then generalize their methods so as to be useful in other situations. The goal of this stage is for the teams to develop an interview schedule of questions to ask their "clients" and to be able to probe

* Copyright is held by the author/owner.

for other information such as physical documents to inspection in order to better understand a client's needs.

Stages II and III are designed for the different teams to interact with each other through the roles of "designer" and "client". In Stage II, one-half of the teams are designated as "designers" and the other one-half as "clients". They use the strategies developed in Stage I to investigate the other team's assigned project. At the end of the time interval, teams are given a few minutes to confer about their experience and then team roles are reversed; "designer" teams become the "client" and vice-versa. At this switch point (Stage III), it is important that each team now interacts with a completely different team in their new assigned roles. The aim of these two stages is for the project teams to become immersed in the elicitation process and to be confronted with the issues of exposing the needs of a client through dialog and the challenges of expressing one's needs from a client's perspective.

Stage IV consists of each team reconvening and reflecting on the processes and roles in which they just engaged. They are to use the initial list of strategies from Stage I and identify which worked and did not work well while also providing recommendations for how the elicitation process could be improved from both the "designer" and "client" perspectives. After about 10 minutes, each team presents a summary of their list of items and recommendations as part of a class discussion. The last activity in Stage IV is to have the teams present what they determined to be the top 2 or 3 greatest needs expressed by their "client". This list is transcribed to the "white-board" in the form of a simple list. Once all content is gathered from the teams, a class discussion ensues focusing on this list of items and their designation as a "need" or a "requirement". The outcomes of Stage IV are to 1) draw out the strategies that work well for needs and requirements elicitation and how being exposed to the different roles can help us be more focused and intentional in our approaches and more productive in our efforts as designers; and 2) better differentiate between a client's needs (e.g. goals to be accomplished) from the requirements (e.g. objectives required to accomplish a goal) of a system which ultimately lead to technical specifications.

This assignment was used is a Research & Development Laboratory course which is linked to an upper-level Systems Analysis and Design course. Nearly all the students indicated on the end of course evaluations the value of their participation in this simulation and how it they felt better prepared when conducting needs and requirements election with their real client.

- [1] Fox, Christopher, *Introduction to Software Engineering Design: Processes, Principles and Patterns with UML 2*, 1st Edition, Boston, MA: Pearson Education, Inc., 2007.

BOID SWARMS*

NIFTY ASSIGNMENT

*Andrew Mertz and Nancy Van Cleave
Eastern Illinois University
Mathematics and Computer Science Department
Charleston, IL 61920
(217)581-3828, (217)581-5228
aemertz@eiu.edu, nkvancleave@eiu.edu*

Graphics programs can be used to spark student interest and enthusiasm, and if they are interactive, even better. By supplying skeleton programs (with sliders) to CS I students, they are able to write such programs by mid-semester. CS II students add further event driven programming by utilizing mouse clicks.

One straightforward and easily understood concept is to treat the graphics window as a grid of blocks to be colored according to a given set of rules. From checkerboards, we advance to fractals (Julia Sets), a Lite-Brite simulation, and then to Conway's Game of Life. The fractals and Game of Life are based on very simple rules, which can result in unexpectedly complex relations and behavior.

Emergence is the name for such outcomes - the way complex systems and patterns arise out of a multiplicity of relatively simple interactions. Continuing this theme of **emergent behavior**, we also look at *boids*, which create swarming behavior in a simulated flock.

Basic models of flocking behavior are controlled by three simple rules:

1. Separation - avoid crowding neighbors (short range repulsion)
2. Alignment - steer towards average heading of neighbors
3. Cohesion - steer towards average position of neighbors (long range attraction)

With these three simple rules, the flock moves in an extremely realistic way, creating complex motion and interaction that would be extremely hard to create otherwise. Several mechanisms for controlling the strength or weakness of these rules governing the movements of the flock are implemented. By varying these values, interesting behaviors may be observed.

These assignments have piqued student interest and creativity, resulting in higher levels of enthusiasm.

* Copyright is held by the author/owner.

SKIP-3 SOLITAIRE*

NIFTY ASSIGNMENT

David Reed
Department of Computer Science
Creighton University
davereed@creighton.edu

SUMMARY Students modify and implement interacting classes in order to simulate a solitaire game involving a row (list) of cards.

TOPICS The key topics for these assignments are interacting classes, user interaction, and the use of ArrayLists for storing and manipulating sequences of objects.

AUDIENCE This assignment is given near the end of a CS1 course using Java, after students have had experience with interacting classes, Strings, and ArrayLists.

STRENGTHS This assignment involves working with interacting classes, as students must build upon provided classes for modeling cards and decks of cards. Given these classes, they must implement a class that models a row of cards, providing methods that correspond to the various actions of the game (e.g., adding a new pile, moving one pile on top of another). This forces them to think abstractly and call appropriate card and deck methods without necessarily knowing the internal behavior of those methods. The assignment provides good experience with class design and ArrayList manipulation, as most of the common ArrayList methods are utilized, including add, remove, indexOf, get, and set. The rules of this particular solitaire game are simpler than most, making it possible for a first-semester programmer to design and implement a simulation. Students enjoy creating an interactive game, and take great pride in playing the game that they themselves implemented (sometimes to the point of compulsion).

WEAKNESSES While the student is only required to implement one class and make modifications to another class, there is a considerable amount of code that comprises the project and that they must comprehend. Weaker students may initially be overwhelmed and need assistance in reviewing the provided code framework.

* Copyright is held by the author/owner.

Skip-3 solitaire is a solitaire game in which cards are dealt one at a time from a standard deck and placed in a row. If the rank or suit of a card matches the rank or suit either one or three cards to its left, then that card (and any cards in the pile beneath it) can be moved on top of the matching card. For example, suppose the three of clubs, five of hearts, nine of spades, and jack of hearts are initially dealt. They would be placed in a row, in left-to-right order, as shown below.

[3C, 5H, 9S, JH]

If the next card dealt is the five of spades, then that card is placed to the right and is found to match the five of hearts three spots to its left. Thus, the player can move the five of spades on top of the five of hearts, shortening the row of cards. The consolidation of these piles then allows the nine of spades to match with the five of spades that is immediately to its left. Thus, the player can move the pile topped by the nine of spades on top of the five of spades, shortening the row even further. Pictorially, the consolidation steps are:

[3C, 5H, 9S, JH, 5S]

[3C, 5S, 9S, JH]

[3C, 9S, JH]

Note that the addition of a single card can start a chain reaction in which many piles of cards, both to the left and right, are moved. The goal of the game is to end with as few piles of cards as possible. Theoretically, the game could end with a single pile of 52 cards, although that outcome is extremely unlikely.

For this assignment, students are given `Card` and `DeckOfCards` classes, which implement the basic operations on a card (`getRank`, `getSuit`, `matches`, `equals`) and deck of cards (`shuffle`, `dealCard`, `cardsRemaining`). Building upon these provided classes, they must implement a `RowOfPiles` class that models the row of individual card piles that are used in the game. This class must have the following methods:

```
void addPile(Card top)
    Adds a new pile at the end of the current row of piles.

boolean movePile(Card fromTop, Card toTop)
    Moves a pile on top of another pile either 1 or 3 spots away.

int numPiles()
    Determines the number of piles in the row.

String toString()
    Returns a string representation of the row.
```

Implementing the `RowOfPiles` class involves maintaining an `ArrayList` of `Card` objects (since only the tops of piles must be stored). Each method of the class involves calling the appropriate `ArrayList` methods to update the row, as well as the appropriate `Card` methods to compare individual cards. A driver class is used to allow read in moves from the user (such as dealing a card from the deck or moving one pile on top of another) and control the game.

A full version of the assignment, along with source code, can be found online at <http://dave-reed.com/csc221.S10/HW/HW7.html>.

USING SAP ERP SOFTWARE IN ONLINE DISTANCE EDUCATION*

*John L. Wilson and Ed Lindoo
College for Professional Studies
Regis University
Denver, CO 80221-1099
920-869-3446
jwilson@regis.edu, ed.lindoo@scripps.com*

ABSTRACT

This paper describes the design, delivery and outcomes at one university when using SAP's enterprise resource planning (ERP) software in a management information systems (MIS) course. The course provides students "hands on" experience using SAP software to learn how an ERP system could help a company streamline business processes, improve communication among departments, and facilitate sharing of information. Topics include joining SAP's University Alliances (UA) program, using SAP's University Competency Center (data center), instructors' education and preparation for students, administration during class, and learning outcomes. Discussed are successes and failures when incorporating SAP ERP software in an online distance education class. Also discussed is a more successful alternative using a simulator for SAP ERP. Other universities and colleges considering, or having implemented, SAP ERP into their curricula may benefit from lessons learned.

INTRODUCTION

Systems Analysis and Program Development (SAP) is the world's largest and most well known vendor of ERP software. SAP created the University Alliances (UA) program to have their SAP ERP software included in higher education programs. The UA program provides universities and colleges with access to a full suite of SAP ERP software, "hands

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

"on" application exercises, data center support, training seminars and educational resource materials. SAP ERP would appear to be a great tool to assist teaching integrated process theory and concepts within an online management information systems (MIS) course.

At a U.S. university, School of Business faculty members realized their MIS course needed to be revamped. The MIS course was part of their online distance education MBA program. Business school faculty members were aware the global economy was driving businesses to adopt enterprise resource planning (ERP) systems. Faculty members understood companies in their geographic area were expecting prospective employees to have a knowledge base of the various roles an ERP system could play in a company to increase communication and efficiency. With additional constructive feedback from students, faculty members encouraged revision of their online MIS course to include SAP ERP software, as many other business schools have done [1, 2, 3, 4, 6]. Recently, published guidelines for information systems curricula also proposed an understanding of the theoretical and practical issues related to business process management and enterprise systems, with the inclusion of an ERP system [5].

SAP ERP SOFTWARE & UA PROGRAM

To get started, the lead MIS faculty member completed the application for the School of Business to join SAP's University Alliances (UA) program. The UA program gives access to SAP's full suite of ERP software and "hands on" application software training exercises - both hosted from university competency centers (UCCs). The School of Business petitioned SAP's UA program to become members. After acceptance and payment of membership fees, the school arranged for the MIS course instructors to receive SAP training. The instructors completed introductory training on SAP ERP software and an application training exercise called Fitter Snackers (FS). Training programs and "hands on" business process-oriented exercises (like FS) are plentiful through the UA program. An annual UA Curriculum Congress distributes these materials to its members. UA coordinators work diligently year-round to communicate materials to members through Websites, instructor-led seminars, etc. Faculty members from universities and colleges around the world, including seven MIS instructors from this School of Business, annually attend the UA Curriculum Congress.

The initial project team consisted of three MIS instructors. The lead faculty member was the project manager. As mentioned, all team members were trained on SAP ERP software and FS exercises at a UA Curriculum Congress. Immediately after training, faculty members anxiously started writing guidelines and procedures to incorporate SAP ERP into the new MIS course. They included Concepts in Enterprise Resource Planning (Third Edition) by Monk and Wagner [8], a companion textbook to the "hands on" Fitter Snacker (FS) exercises. Students would perform the FS exercises online after connecting with UA's UCC data center using a graphical user interface (GUI) on their PC and the Internet. The original author of the FS exercises [8] wrote step-by-step instructions for "how to" use SAP ERP to execute business transactions. Using these instructions and screen snapshots, students would be expected to perform the business transactions on their own and take a final snapshot of their PC's screen to prove they successfully reached the end of an exercise.

Difficulties Begin

Difficulties with design and development of the online MIS course began almost immediately. Writing installation procedures for online students became more difficult when faculty members received operating instructions from the UCC's data center. The UCC provided a ZIP file containing the SAP front-end GUI. The ZIP file was large (over 600 MB). Not all students would be able to download it onto their PCs. The UCC data center sent account ID's and passwords to the MIS instructors for distribution to students in their classes.

The Help Desk at the School of Business was allowed to opt out of supporting the SAP GUI, administration of account ID's, and problem solving for the FS exercises. The Help Desk agreed to only reset passwords. MIS instructors were on their own to support and solve any problems students would encounter installing the SAP GUI and/or performing FS exercises.

The lead MIS faculty member used the "big bang" approach to implement the newly revamped MIS online course. Only the MIS course using SAP ERP software was offered to students, which meant students in all classes (approximately 300 students in eight classes) would encounter all design flaws and instruction errors.

When classes started, students began installing the 600 MB GUI on their PCs and connecting through the Internet with SAP ERP software running in the UCC's data center. The installation process for students, however, was most arduous. Many unforeseen problems occurred. Not anticipated were all the different versions of computer hardware and system software students used. The GUI was certified to run on Windows XP Professional; however, many students used Windows Vista, XP Home edition, or Windows 98, as well as a variety of service packs. After discovering a great deal of unnecessary information within the GUI installation file, one of the instructors reduced the 600 MB ZIP file to 164 MB - one-fourth the original size. The instructor also rewrote installation procedures, and included screen shots of installation steps, to make installations more reliable on the large variety of Windows operating systems. However, some students were unable to get the smaller Windows GUI to work and were given a Java version of the GUI. Java was a good fit for these students, as few issues were encountered after using it. However, the step-by-step snapshots from the Windows GUI instructions appeared differently when running the Java GUI, creating frustration and confusion for Java users.

Some students had Macintosh (MAC) computers. The instructor had to provide a MAC version of the GUI for MAC users, as they could not use the Windows GUI. Although this worked well for MAC users, the instructor needed to rewrite instructions for MAC users, because the instructions didn't look the same as those used with the Windows GUI.

In the end, four different versions of the GUI were loaded onto a central server within the University, and links to these were provided in the online classrooms. As mentioned, one of the instructors wrote detailed installation instructions with screen snapshots for each GUI version. All instructors distributed the installation materials, along with detailed FS exercises, to students in their online classrooms. Data for the Fitter Snacker Company resided in an Oracle database at the UCC's datacenter in Milwaukee,

Wisconsin. Students accessed the data through the Internet using the SAP GUI client they installed on their PC.

Problems Continue

While access through the Internet was quite consistent, the UCC computers were not always available, which hindered students' attempts to complete their exercises before assignment deadlines. (Sunday night due date, for example.) Solving students' problems was a challenge for instructors, because the UA did not provide an SAP help desk for instructors to obtain answers for difficult problems. Creation of an "SAP Help" forum on each classroom's discussion board was found to be most useful. Instructors told students to use the "SAP Help" forum to post questions, so that fellow students could assist with their responses. This worked well, because many students experienced the same problems and were able to help others solve common problems.

As previously mentioned, the School of Business' Help Desk did agree to reset students' passwords. This proved quite valuable. Some students did forget their passwords, and some students would try logging into a wrong account. After three failed attempts, students logging into the wrong account would lock the rightful owner's password. For example, if a student were assigned account FSUSER-22 but accidentally-keyed FSUSER-23, then after three failed logon attempts, they would lock out account FSUSER-23.

One of the biggest issues with the SAP program was the inability to keep students from contaminating other students' datasets. For example, a student could login to their assigned account FSUSER-22 and then use the dataset belonging to account FSUSER-23, in addition to any other datasets belonging to accounts FSUSER-00 through FSUSER-99. There was no apparent way to prevent this from happening. Even though instructors advised students of these issues, and the inappropriateness of using another account's dataset, students continued using other accounts' datasets. Of course, this corrupted other datasets, causing students to become frustrated.

Following the step-by-step instructions for FS exercises, often some 60+ pages long for one exercise; it was quite easy for a student to miss a step, or even key in the wrong data. Once a step is missed, or wrong data entered, it's akin to closing the books at month end. Once closed, the average user cannot go back and make changes. Because the students knew what results they were supposed to achieve (as provided in the step-by-step instructions), when they couldn't get these results, they became quite frustrated. Their emails for help from instructors became quite demanding. Students were willing to do the work, and to do it over and over to get it right; however, students' datasets that were corrupted with incorrect data could not be reset. There really was no way to "fix" a mistake. Resetting datasets was a frequent request from frustrated students. One way instructors accommodated this was by giving students an unused account, if one was available. That was not always the case. This led to additional administrative work for instructors - keeping track of accounts that had been switched for students.

As twenty percent of their final grade was contingent on scores from their FS exercises, students would complain when their FS scores were low. Often their complaints were directed to the lead MIS faculty member, the Help Desk and others in

the administrative section of the School of Business. Some very frustrated students directly approached the Office of the Dean of the School of Business to complain about the SAP ERP software and FS exercises in their MIS online classes.

Giving Up

As described above, the implementation of SAP ERP and FS exercises failed miserably. There was no buy-in from anyone within the School of Business, except from the lead and MIS faculty members teaching classes. The implementation also failed, because the SAP GUI was a major install on students' PCs. All students could not successfully use the ERP software to complete the complex FS exercises without a lot of handholding by the instructors. With no support from the School of Business' Help-Desk, the start of every subsequent semester was hectic for instructors and students. Something had to be done to rectify all the frustrations and very verbal complaints from students. As a result, the School of Business opted to quit using SAP ERP software, FS exercises, and the UA Program. Instead, they decided to use an SAP simulator created by Simha R. Magal and Jeffrey Word [7] in partnership with SAP.

AN ALTERNATIVE - SAP ERP SIMULATOR

The first step in getting started was to contact WileyPLUS and obtain access to Websites to setup a simulator for each MIS class. Once an instructor had access to a Website, setup was straightforward and very intuitive. The instructor created a new class by assigning five simulator exercises and five quizzes to specific weeks of the class. After setting up the simulator for a class, an instructor automatically received the URL to provide to students, so they could gain access to the simulator's Website. If students purchased a new Magal & Word textbook [7], they received an access code. If they purchased a used textbook, they had to purchase an access code during registration on the WileyPLUS simulator Website.

Once students had access to the Website, they self-registered to use the simulator. Throughout the MIS class, students completed pre-defined simulator exercises and quizzes, which were assigned on a week-by-week basis. The simulator Website had five process simulators and five quizzes. The simulator exercises were automatically graded as a percentage of completion. That is, if a student started a simulation and completed 75% of the exercise, then 75 was their grade. Most students successfully completed the simulator exercises and ended up with 100 on each of the five exercises. The quizzes, on the other hand, were challenging. Some quizzes only had four questions. If a student incorrectly answered two questions, they only earned 50% for that quiz. Instructors encouraged students to run simulator exercises over and over, and to take detailed notes during the exercises, until they became very familiar and comfortable with the simulated process flow. That way, they were better prepared to answer quiz questions.

The simulator was a Java-based program running in a browser. It was extremely easy to use. From start to end of the simulation, the simulator led the student from field to field, and screen to screen. The simulator instructed the student on what to do, and

even prompted with a red highlight on what fields or instruction set they needed to click to work their way through the process flow.

For instructors and students, the simulator was quite straightforward. As mentioned, the instructor created a new class by assigning five simulator exercises and five quizzes to specific weeks of the course. Students self-registered on the simulator Website, and then performed the assigned exercises - once with the simulator performing all the steps, and then three times on their own with the simulator guiding them through the processes. A quiz for each simulator exercise measured students' comprehension of process flow. Students' results were recorded in their grade book. All grading information could easily be downloaded to an Excel spreadsheet for performing calculations to determine students' overall weighted grades for the exercises and quizzes.

CONCLUSION

The authors have described how two different implementations of SAP's ERP system were incorporated into a business school's MIS course conducted online. In reviewing the objectives for the MIS course, the authors asked: "Is the objective to learn how to use SAP ERP software and reach the correct results found at the end of step-by-step exercises (as is done with the Fitter Snacker exercises), or is it to learn how an ERP system can improve processing flow and communication within an organization (as in the SAP simulation exercises)?" Students who used SAP ERP to complete FS exercises primarily concentrated on obtaining the correct results at the end of step-by-step exercises, with little focus on learning processing flow. On the other hand, students who used the SAP simulator better understood how communication and processing could successfully flow though a business organization.

One important lesson learned is that trying to implement SAP ERP into a single MIS course with no support from school resources (especially the help desk) is a plan for failure. SAP ERP is complicated software to use and covers a large variety of business processes from inbound logistics with suppliers to outbound logistics with customers. To obtain a thorough understanding of the software and business processes included, students would need to use SAP ERP software in a series of courses throughout an entire MBA curriculum. In each course of the curriculum, the subject matter of the course could be supplemented with an appropriate, accompanying SAP module. Working together, they would enhance a student's understanding of the course subject matter and ERP software functionality.

SAP ERP simulation software takes away the need for a student to know all the details about how to process transactions. Instead, the simulator guides students through processing steps with a concentration on explaining how ERP systems allow processing to flow across organizational functions, insuring communications between involved departments, and allowing processing to be completed efficiently, timely and with fewer errors, as compared to manual processing. Using a simulator allows students the opportunity to understand process flows, and appreciate the benefits that could occur, when using ERP systems in businesses.

REFERENCES

- [1] Becerra-Fernandez, I., Murphy, K.E., & Simon, S.J., Integrating ERP in the business school curriculum, *Communication of the ACM*, 43, (4), 39-41, 2000.
- [2] Cannon, D.M., Klein, H.A., Koste, L.L. & Magal, S.R., Curriculum integration using enterprise resource planning: An integrative case approach, *Journal of Education for Business*, 93-101, 2004.
- [3] Corbitt, G., Integrating SAP R/3 into a college of business curriculum: Lessons learned, *Information Technology and Management*, 1, (4), 247, 2000.
- [4] Davis, C.H., & Comeau, J., Enterprise integration in business education: Design and outcomes of a capstone ERP-based undergraduate e-business management course, *Journal of Information Systems Education*, 15, (3), 287-299, 2004.
- [5] IS 2010: Curriculum guidelines for undergraduate degree programs in information systems. *Communication of the Association for Information Systems*, 26, 359-428, 2010.
- [6] Kirkham, K., & Seymour, L., The value of teaching using a live ERP system with resource constraints.
<http://www.commerce.uct.ac.za/informationsystems/Staff/personalpages/lseymour/2007/2005-WCCE-376.pdf>, retrieved October 12, 2010.
- [7] Magal, S.R. & Word, J., *Essentials of Business Process and Information Systems*, Hoboken, NJ: John Wiley & Sons, Inc., 2009.
- [8] Monk, E. & Wagner, B., *Concepts in Enterprise Resource Planning*, Boston, MA: Course Technology Cengage Learning, 2009.

INFORMATICS IN THE CLASSROOM: AN EVALUATION OF INNOVATIVE AND EFFECTIVE USE OF TECHNOLOGY IN ONLINE COURSES *

*John P. Buerck, Patricia G. Bagsby, Stephanie E. Mooshegian, and Lisa M.
Willoughby*

*Saint Louis University
3840 Lindell Blvd., St. Louis, MO 63108
314-977-2320
buerckjp@slu.edu*

ABSTRACT

Informatics is a field of study that emphasizes the interaction between individuals, groups, or larger social contexts, and information and communication technologies (ICTs). Distance learning is emerging as one of the most common mediums for person-technology interaction, and therefore is of particular interest to informatics researchers. The current study explores the relationship between perceptions of effective use of technology in online courses with an administrator and student satisfaction in a sample of adult learners. These results show that satisfaction with online courses is significantly related to the instructors' ability to deploy the available course management tools in an innovative way. Additionally, while administrator and student ratings for overall course satisfaction were positively correlated, there was no statistically significant relationship between administrator ratings of innovation and student perceptions of effective technology use.

INTRODUCTION

Distance learning has become a staple in higher education [1]. Once believed to be a fad, distance learning is now common practice in Universities and educators want to know how they can efficiently use technology in a virtual classroom. In fact, while some

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

attention has been given to the types of content that should be included in an online course (e.g., online chats, discussion threads, video-streamed web casts, PowerPoint lectures, and multimedia lectures) [2, 3], there are many unanswered questions regarding the use of technologies in online courses and their impact on student satisfaction with the course. The purpose of the current study was to examine the relationship between administrator and student perceptions of the use of technology in online courses, as well as the evaluation of overall course satisfaction among nontraditional undergraduates.

Informatics in Higher Education

It has been noted that the traditional concept of what constitutes learning and the learning environment is changing; and with this change comes a required shift in paradigm for educators with regard to pedagogical practice (e.g., experiential and active learning practices) [2]. Consequently, this paradigm shift, coupled with rapidly advancing technology, necessitates the identification of new boundaries within which educators operate. These boundaries have extended beyond traditional classrooms with face-to face interaction to now include both asynchronous and synchronous learning in multiple distance modalities (e.g., learning management systems (LMS), webinars, and video-conferences). With the acceleration of technology, educators may benefit from clearly defined parameters with which to analyze and interpret success, and failure, in the virtual classroom. Thus, the field of Social Informatics is of particular relevance to educators.

Social Informatics has been described as "the body of systematic research about the social aspects of [information and communication technologies] ICTs" [4]; and further elaborated as "a field of research focusing on the relationship between them [ICTs] and the larger social context in which they ICTs exist" [5]. Furthermore, parameters for defining what types of research fall under this field include context-dependent, methodologically-plural, problem-oriented, and trans-disciplinary [5]. These broad boundaries for identifying Social Informatics research are equally beneficial and problematic. Casting a wide net allows the field to incorporate research from multiple disciplines, yet this also means that research findings are scattered in a wide variety of journals, a prevalent issue due to the slow adoption of the common term, "Social Informatics" [6]. Subsequently, there is a lack of congruence in how researchers classify the social context. Although social groups, such as work teams, organizations as a whole, and even the entirety of internet users, are more typically included in studies of social informatics [5], the educational setting is an emerging social context of interest. Given that distance learning is becoming more common practice, there is a need to examine the interaction between the end user (i.e., students) and online learning modalities employed.

Traditional and Nontraditional Student Satisfaction in Online Courses

Although there has been considerable effort made to compare ratings of course satisfaction among online and on-ground students [7, 8, 9], there is a paucity of research that directly addresses comparisons between traditional and nontraditional students [10]. Existing research suggests both traditional and nontraditional student samples value flexibility as an important factor for online courses [10, 11]. However, evidence largely

suggests these two populations have very different perceptions of ideal online courses. One difference is the desired amount of peer interaction. While interaction with other students has been shown to be a predictor of satisfaction in traditional student samples [12], nontraditional students have been shown to desire less interaction among peers within the course (e.g., discussion boards) and less interference with their full-time work to be among the most important elements of an online course [10]. Furthermore, nontraditional students have been shown to have more overall satisfaction with distance learning than traditional undergraduate students [13]. Specifically, active communication with instructors has been shown to play a role in nontraditional students' positive ratings of course satisfaction [14]. These studies highlight a few of the very distinct differences between traditional and nontraditional students' preferences for online learning.

Unfortunately, most researchers include a mix of traditional and non-traditional students within a single sample, or they omit a full description, which means there is no way to distinguish those students who are in college at the typical age from those who are returning, or are first-time later-in-life students [9, 11, 15, 16]. Much of what is known regarding course satisfaction and the role of technology in the classroom with regard to satisfaction is based on samples of traditional-aged undergraduate students. Therefore, we believe that there is a need for more consideration of nontraditional undergraduate students' preferences for technology in the classroom and how these preferences relate to course satisfaction.

Role of Technology in Online Courses

Due to the introduction of technology in the classroom and the inherent shift in paradigms, instructors' views on the role of technology in the virtual classroom are mixed. Raths [17] described effective online instruction as balancing between a role that is similar to that of an instructor in an on-ground classroom and a "technical support representative." Through this lens, technology in the virtual classroom is interpreted more so as an element of maintenance rather than an outlet for innovation. In contrast, Olcott and Wright [18] described a need for technology coupled with creativity to create high quality teaching in online course settings. Along this line, researchers have identified effective uses of online methodologies as a way to facilitate student-instructor communication [2]. Although effective interactions in a virtual classroom are likely to be a more creative endeavor than interaction in an on-ground classroom, communication with the instructors has been shown to be an essential component for satisfaction in an online course for the nontraditional student [14]. Therefore, it would be expected that a more innovative approach to using the tools available in an online learning system to enhance communication between students and instructors is likely to lead to more positive overall perceptions of the course. Underlying this reasoning is the belief that innovative approaches to the presentation of course material can be treated as avenues to reach the multiple needs of online students. Non-traditional students will perceive innovation as an effective use of the tools that are available to instructors, which will create a more positive impression of the instructor and the course overall. Therefore, we posit that perceptions of effective use of technology (EUT) will be positively related to overall evaluations of the course by both administrators and non-traditional students.

METHODOLOGY

Sample. Data were collected from an adult education program at a Midwestern university. A variety of undergraduate core courses including, but not limited to, English, History, Math, and Psychology (N=38), were evaluated by both the students and the administrators.

Evaluation Process. The administrator course review was conducted on the sample of courses from two to four weeks prior to the start of the term. Courses were assessed with the "course review form" (see description below) and seven administrator raters were provided with frame of reference training for internal consistency of their ratings [19]. One rater assessed all classes to further add control to the rating process. Courses were also rated by students at the end of the term via online questionnaire. The response rate for student end-of-course evaluations was 66% (375/567). Overall Course Assessment. Student evaluation of the course consisted of 16 items rated on a Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). Items reflected a range of evaluative areas including clarity of expectations, delivery, and direct application of classroom concepts. The mean score for the entire 16 items, averaged across students, was used to measure students' overall evaluation of the course (N=38). Administrator evaluation of the course consisted of an assessment measuring five major areas: Instructional Design, Learner Support, Course Organization and Design, Assessment, and Innovative Teaching with Technology. The mean score for these five areas was computed to assess administrators overall evaluation of the course. Means and standard deviations of assessment items for students and administrators can be found in Table 1.

Table 1: Means and Standard Deviations of Survey Items

<i>Item</i>		Mean	SD
Administrator Evaluation of Course			
1. Instructional Design	3.45	.59	
2. Learner Support	3.10	.83	
3. Organization and Design	3.06	.74	
4. Assessment	3.35	.32	
5. Innovative Teaching with Technology	3.16	.72	
Average Administrator Evaluation Score	3.22	.47	
Means of Mean Scores for Student Evaluation Items			
1. Course objectives were clearly presented.	4.27	.57	
2. Course expectations were clearly stated.	4.33	.52	
3. The course was intellectually challenging.	4.44	.53	
4. The course increased my knowledge and understanding of the subject.	4.26	.69	
5. This course used Blackboard effectively to meet educational needs	4.25	.74	
6. The instructor was accessible and responsive to students in the on-ground or online classroom.	4.07	.79	
7. The instructor effectively engaged me in the on-ground or online classroom.	4.01	.76	

8.	Having taken this course, I am interested in learning more about this topic.	3.56	.81
9.	I can apply the knowledge I acquired in this course to real life situations.	3.95	.74
10.	This course improved my communication skills.	3.64	.79
11.	This course enhanced my ability to work collaboratively with others.	3.51	.71
12.	This course encouraged my respect and compassion for humanity and the dignity of each person.	3.81	.80
13.	This course fostered my understanding of the nature of leadership.	3.54	.78
14.	This class strengthened my commitment to service.	3.49	.84
15.	This class developed my understanding of my own values and beliefs.	3.70	.81
16.	<u>Overall, I was satisfied with what I learned in this course.</u>	3.98	.80
<u>Means of Mean Score for Overall Student Evaluation</u>		3.93	.62

Effective Use of Technology (EUT). The mean score of one item ("This course used Blackboard effectively to meet educational needs") from the student course evaluation form was used to measure student perceptions of effective use of technology in the classroom. (Blackboard is the online technology system utilized at this school.) Administrator evaluation of innovation was measured using the mean of the Innovative Teaching with Technology score from the administrator evaluation of the course conducted at the beginning of the term (which included the following items: (a) variety of tools used to facilitate communication, (b) teaching methods applied to enhance learning, engage students interactively, and (c) variety of multi-media elements and learning objects relevant to student learning.

RESULTS

Overall, student evaluation of the total course was in alignment with administrator evaluation of the course ($r = .56$, $p < .001$). In support of our hypothesis, administrator overall evaluation of the course was positively related to both the administrator review score for innovative use of technology ($r = .52$, $p < .01$), and students' perceptions of effective use of technology in the classroom ($r = .51$, $p < .01$). Student overall evaluation of the course was positively related to end-of-course ratings of effective use of technology in the classroom ($r = .86$, $p < .001$), as well as, administrator ratings of innovative use of technology in the classroom ($r = .41$, $p < .05$). However, the correlation between the administrator ratings of innovation and student ratings of effectiveness was positive, but did not reach statistical significance ($r = .30$, $p = .063$). Follow up analyses conducted with the EUT item removed from the overall student evaluation score resulted in the same pattern of relationships.

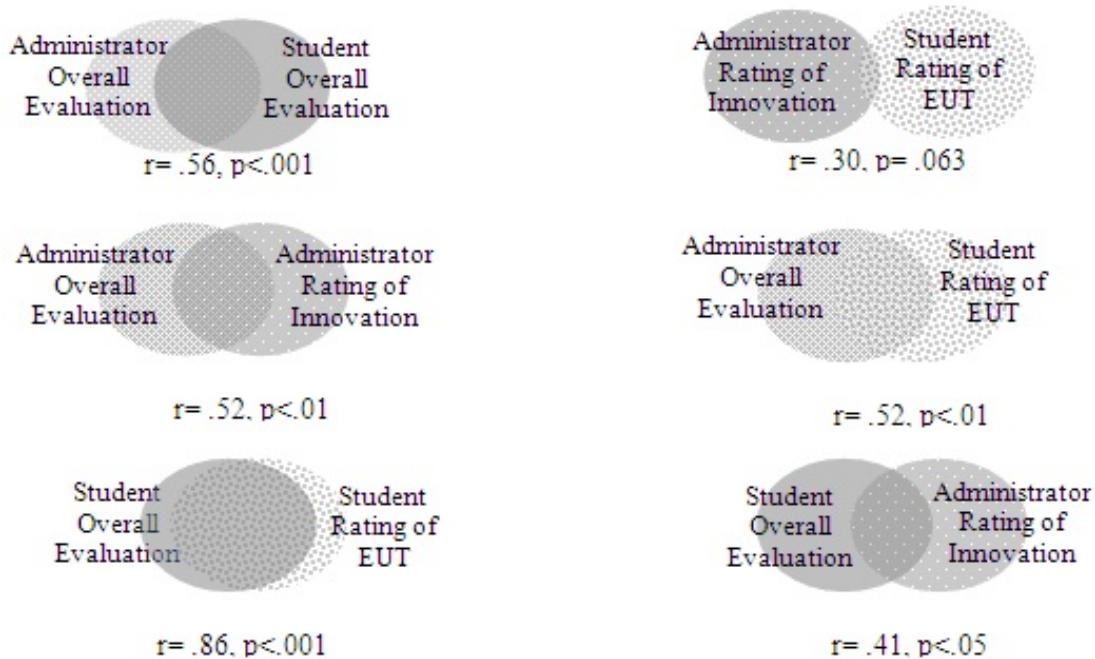


Figure 1: Visual Estimation of Relationships between Study Variables
(Note: EUT=Effective Use of Technology)

DISCUSSION

It appears that the administrator and students were in agreement with regard to the importance of the effective use of technology in online courses. Additionally, while administrators and students concurred on the overall evaluation of a course, there was not a statistically significant relationship between the two groups and their perceptions of what constituted effective use of technology. While the correlation between student perceptions of effective use of online learning technology and administrator ratings for innovative teaching with technology did not reach significance, the lack of significance may be due to the use of one-item measures or the low power in this study.

This study lends support to the idea that online teaching is more than incorporating basic content on PowerPoint slides from textbook publishers or supplemental readings into online courses. In a virtual classroom setting, the online learning system can be seen as the "physical" aspect of the classroom. It takes the place of the desks, chairs, and chalkboards with which students are familiar and allows instructors an outlet for building the atmosphere of their classrooms. While these physical aspects may not consciously be taken into consideration for evaluations of on-ground courses, students in online courses are more acutely aware of the learning atmosphere in which they are operating [20]. These results show that satisfaction with an online classroom is significantly related to an instructor's ability to deploy the options available to him or her in an effective and innovative way. Additionally, due to the implications for improving student satisfaction with online courses, programs that offer online courses may want to consider developing or expanding opportunities to train instructors to utilize technology in effective and innovative ways (providing demonstrations of innovative teaching with technology).

REFERENCES

- [1] McIsaac, M.S., & Gunawardena, C.N. Distance Education. In D.J. Jonassen, (Ed.) *Handbook of Research for Educational Communications and Technology: A project of the Association for Educational Communications and Technology*. New York: Simon & Schuster Macmillan, 1996.
- [2] Atkinson, T., Atkinson, R.H., & Smith, D. GOALS (graduate online active learning strategies), *Journal of Computing Sciences in Colleges*, 17, (3), 251-264, 2002.
- [3] Nelson, M., Settle, A., Bhagyavati, Shaffer, D., Miles, G., & Watts, J., Online teaching practices (both best and worst). *Journal of Computing Sciences in Colleges*, 21, 2, 223-230, 2005.
- [4] Kling, R., Crawford, H., Rosenbaum, H., and Sawyer, S., Learning from social informatics: Information and communication technologies in human contexts. *Indiana University Center for Social Informatics*, 2000. Informatics. http://rkcsi.indiana.edu/media/SI_report.pdf
- [5] Sawyer, S., & Eschenfelder, K. R. *Social informatics: Perspectives, examples, and trends*. *Annual Review of Information Science and Technology*, 36, (1), 427-465, 2002.
- [6] Kling, R., What Is Social Informatics and Why Does It Matter? *The Information Society*, 23, 205-220, 2007.
- [7] McCloud, R., Does an online course work on computer science, *Journal of Computing Sciences in Colleges*, 19, (5), 260-269, 2004.
- [8] Ury, G., A comparison of undergraduate student performance in online and traditional courses. *Journal of Computing Sciences in Colleges*, 19, (4), 99-107, 2004.
- [9] Kozak, M.M., A comparison of student perceptions regarding online learning versus traditional classroom learning. *Journal of Computing Sciences in Colleges*, 25, (1), 75-93, 2009.
- [10] Valenta, A., Therriault, D., Dieter, M., & Mrtek, R., Identifying student attitudes and learning styles in distance education. *Journal of Asynchronous Learning Networks*, 5, (2), 111-127, 2001.
- [11] Johnston, J., Killion, J., & Oomen, J., Student satisfaction in the virtual classroom. *The Internet Journal of Allied Health Sciences and Practice*, 3, (2), 1-7, 2005.
- [12] Fulford, C. & Zhang, S., Perceptions of interaction: A critical predictor in distance education. *The American Journal of Distance Education*, 7, (3), 8-21, 2003.
- [13] Skopek, T. A., & Schuhmann, R. A., Traditional and non-traditional students in the same classroom? Additional challenges of the distance education environment. *Online Journal of Distance Learning Education*, 11, (1), 2008.

- [14] Ashburn, L.J., Course design elements most valued by adult learners in blended online education environments: An American perspective, *Educational Media International*, 41, (4), 327-337, 2004.
- [15] Reeves, T., Baxter, P., & Jordan, C., Teaching computing courses- computer literacy, business microcomputer applications, and introduction to programming online utilizing WebCT, *Journal of Computing Sciences in Colleges*, 18, (1), 290-300, 2002.
- [16] Holcolm, L.B., King, F.B., & Brown, S.W., Student traits and attributes contributing to success in online courses: Evaluation of university online courses, *The Journal of Interactive Online Learning*, 2, (3), 1-17, 2004.
- [17] Raths, D. Is anyone out there? *Inside Technology Training*, 32-34, 1999.
- [18] Olcott, D. Jr. & Wright, S., An institutional support framework for increasing faculty participation in postsecondary distance education. *The American Journal of Distance Education*, 9, (3), 5-17, 1995.
- [19] Bernardin, H.J., & Cooke, D.K. A consideration of strategies in rater training. *Academy of Management Review*, 6, (2), 205-212, 1981.
- [20] Richardson, J.C., & Swan, K., Examining social presence in online course in relation to students' perceived learning and satisfaction, *Journal of Asynchronous Learning Networks*, 7, 64-88, 2003.

ORGANIZING AND HOSTING A STUDENT PROGRAMMING CONTEST*

PANEL DISCUSSION

*Brian Hare, Moderator
Computer Science Electrical Engineering Dept
School of Computing and Engineering
University of Missouri-Kansas City
450H Flarsheim Hall
5100 Rockhill Rd.
Kansas City MO 64110
816-235-2362
hareb@umkc.edu*

*James Cain
Department of Computer and
Information Science
Southwest Baptist University
Bolivar, MO 65613
jcain@sbu.edu*

*John Cigas
Department of Computer Science,
Information Systems, and Mathematics
Park University
Parkville, MO 64152
cigas@acm.org*

ACM hosts an annual student programming contest [1]. Contests are also offered regularly by several conferences [2], student organizations, and professional societies [3]. Although specific organizational features vary, there are common requirements for organizers and host sites to be aware of. This panel discussion will address computing and infrastructure requirements, developing contest problems, scoring and judging issues, organizational and technical challenges, and how much pizza and soda it takes to keep several teams of student programmers functioning. Confirmed panel members have each been site director for an ACM or conference-based programming contest.

- [1] Association of Computing Machinery International Collegiate Programming Contest. <http://cm.baylor.edu/welcome.icpc>, retrieved December 11, 2010.
- [2] Consortium for Computing Sciences in Colleges, Central Plains Division. <http://www.ccsc.org/centralplains/programmingcontest.html>, retrieved December 11, 2010.

* Copyright is held by the author/owner.

[3] IEEEExtreme. http://www.ieee.org/membership_services/membership/students/competitions/xtreme/index.html, retrieved December 11, 2010

SCRATCH THE WORKSHOP AND ITS IMPLICATIONS ON OUR WORLD OF COMPUTING*

*Judy Clark, Michael Rogers, Carol Spradling
Computer Science/Information Systems
Northwest Missouri State University,
Maryville, MO 64468*

1-660-562-1281, 1-660-562-1551, 1-660-562-1588

clarkj@nwmissouri.edu, michael @nwmissouri.edu, c_sprad@nwmissouri.edu

ABSTRACT

In an attempt to foster an interest in Computer Science, and help students appreciate the fundamental concepts in the field, faculty at universities, in conjunction with their teaching colleagues in K-12, have expended a great deal of effort trying to come up with appealing and educational products. In this paper we focus on one of these products, Scratch, providing a description, rationale, and a summary of our experiences using Scratch in a variety of educational settings.

1. INTRODUCTION

Much has been written on the need to inject Computer Science topics into the K-12 curriculum. All students need to know how to problem solve, how to think in a logical systematic fashion, and how to think abstractly, ideas that lie at the very heart of the discipline. Unfortunately, due to numerous challenging logistical difficulties, Computer Science is nearly invisible in K-12.

Many approaches have been suggested to rectify this problem, many of which involve software of one form or another. Among those that have been suggested in the literature, and tested in the field, Scratch, a delightfully simple and appealing software package from the Lifelong Kindergarten group at MIT, has recently risen to the fore.

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

In this paper we will begin with an overview of the dismal state of Computer Science Education in K-12, and its impact on higher education Computer Science programs; then, we will take a brief historical look at a number of approaches to enhancing Computer Science education in K-12; finally, we will report on our experience with Scratch. We have taught Scratch to K-12 teachers in workshops and classes, and, in limited numbers, to K-12 students. Our experience has been universally positive, and while there are no panaceas, Scratch has the potential to help turn around the decline in Computer Science Education.

2. STATUS OF COMPUTER SCIENCE IN K-12 EDUCATION

State-level curriculum standards for computer science are nearly nonexistent. The 2003 Model Curriculum for K-12 Computer Science report defines computer science as "the study of computers and algorithmic processes¹, including their principles, their hardware and software designs, their applications, and their impact on society" [2, p. 2]. Most K-12 state curriculums do not cover these topics. The result is that the general public is not well educated about computer science, which the ACM K-12 Task Force [1] reports may lead to a shortage of information technologists. The second edition of the ACM Model K-12 Curriculum sets the context for computer science within K-12 education today and provides a framework for state departments of education and school districts to address the educational needs of young people and prepare them for personal and professional opportunities in the 21st century [2]. There are current efforts to have this model curriculum adopted at national and state levels. However, school districts are under ever increasing pressure to reduce their budgets and offer only courses that are necessary.

In many high schools, computing courses are seen as a vocational rather than academic subject, emphasizing software such as word processing, presentation software and spreadsheets. The CSTA survey [6], which has been conducted every two years over a six year period, shows that the number of schools offering the introductory (or pre-Advanced Placement) Computer Science (AP CS) have declined in the past six years, 78%(2005), 73%(2007), & 65%(2009) respectively. The same survey reports that 71% of teachers completing the 2009 CSTA survey report that there are qualified students who are not taking Computer Science courses that are offered. In 2008, 56% of AP test-takers were female, yet only 17% of AP Computer Science test takers were female [5].

The state of Missouri does not certify computer science as a content area in teacher education [8]. In most school districts, computer science is viewed as an extension of business education where the focus is mainly on computer usage proficiency rather than on the actual computer science concepts [8]. The 2008 Outstanding Schools Act in Missouri recognizes technology as an important component in preparing students for life-long learning and for the future workplace. In this sense, the computer is seen as a tool to support classroom learning with the diverse student population [7].

3. HOW THIS IMPACTS HIGHER EDUCATION OR PIPELINE ISSUES

The dwindling computer science offerings in K-12 education [21] lead to smaller enrollments at most colleges and universities. Most computer science programs today have a smaller enrollment in computer science programs than ten years ago. However, nationwide enrollments began to increase modestly in 2007, and showed an overall 8.5% increase in undergraduate enrollments in 2009 [23]. Underrepresented groups - women, ethnic minorities, academically disadvantaged, disabled, and low socio-economic status students [12] - have particularly small enrollments in computer science nationally. Data taken from the National Center for Women in Technology [5] shows that women's participation in computing is lower than participation in other STEM areas. In 2008, 65% of Intel Science and Engineering Fair finalists in Biochemistry were female, 35% of the finalists in Math were female, and only 19% of the finalists in Computer Science were female. In short this change in the smaller number of schools offering Computer Science courses feeds smaller numbers of students that are entering Computer Science majors in colleges and universities.

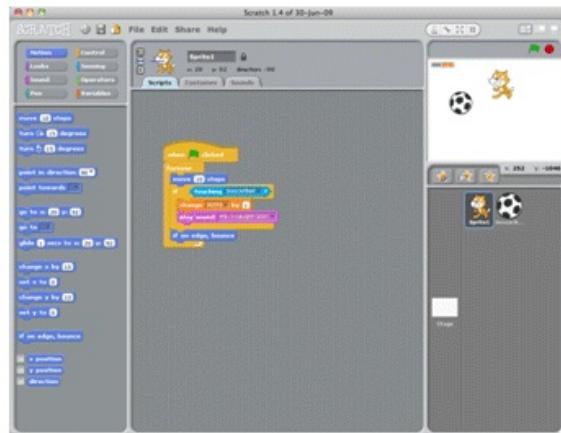
4. ALTERNATIVES TO ATTRACT STUDENTS TO COMPUTING

If more students in K-12 are not participating in Computer Science activities, it is certainly not for lack of effort on the part of their teachers and their colleagues in higher education: over the years scores of approaches and technologies have been created or adapted to entice students [4, 14, 17]. Some of the more notable in current use include: Alice [19], Computer Science Unplugged, Electronic Blocks, (blocks based on Lego Duple Primo that can be snapped together to create various interesting behaviors) [14], Media Computation [9], Nyquist, (software that merges computing and music) [14], Lego Mindstorms, [9], Logo [11], Pico Crickets (an electronic device similar to Lego Mindstorms, but with more of an emphasis on artistic endeavors) [16], Scratch, the subject of this paper (see section 5 for description) [18], and Storytelling Alice, (a variant on Alice that, as its name suggests, emphasizes story telling) [20]

5. SCRATCH OVERVIEW AND INTRODUCTION

Scratch is a visually appealing, extremely easy to use software package that makes it possible to program sprites in a 2D world without having to write any code. It is designed for school children (from grades 2 and up), but is flexible enough that it can even be adapted at the university level for non-majors.

To create a Scratch program, users drag color-coded puzzle pieces onto a scripting area. The puzzle pieces are grouped into categories (Motion, Looks,



Sound, Pen, Control, Sensing, Operators and Variables.) and their names make their function clear (move 10 steps, or turn 15 degrees in the Motion category; repeat 10 and wait 1 seconds in the Control category; etc.). The pieces may be snapped together, and then executed, driving whatever sprite is currently selected on the screen. Pieces can only snap together if the combination makes sense from a "programming" standpoint: their shapes provide visual cues as to whether a particular combination is permitted. For example, a boolean puzzle piece is shaped like an irregular hexagon, and can only be placed in a like-shaped receptacle (as is found in if and repeat-until puzzle pieces).

6. SCRATCH EXPERIENCE AND RESEARCH

Scratch can be used in many curricular areas and across most grade levels. Educators are finding that their students can use Scratch to demonstrate their understanding in a variety of content areas. Wong reports that young people learn "21st century skills that will be critical to success in the future: thinking creatively, communicating clearly, analyzing systematically, using technologies fluently, collaborating effectively, designing iteratively, learning continuously. Learning the fundamentals of programming is a secondary outcome." [22, p. 2].

Fesakis and Serafeim [10] conducted an eight week workshop for 35 prospective teachers in the areas of pre-school and educational design. They taught students Scratch over a six week period and asked students to create their own Scratch projects. They conducted a research study with these students about their attitudes regarding computing, Scratch and the Internet. The study found that 65% of the students thought that Scratch was easy to use and 85% thought that Scratch contained a user interface that used simple and understandable language.

However, an interesting fact is that 80% of the students thought that the use of technology in education was interesting, but after learning Scratch 95% of the students thought that technology in education was interesting. Additionally, 50% of the students reported feeling insecure about technology prior to learning Scratch, while this percentage decreased to 35% after learning Scratch. In summary, Fesakis and Serafeim [10] felt that learning Scratch increased the self-confidence of students toward technology, lessened students stress and anxiety about their ability to use technology and increased their desire to use technology in education.

7. OUR EXPERIENCES

7.1 Missouri Business Education Teachers Workshop (K-12)

In an effort to promote computer programming in the business education classroom, two Scratch workshops were conducted at the Missouri Business Education Association summer conference. Both workshops were well attended and the educators were excited to implement Scratch into their curriculum. Follow up with many of the attendees indicated a Scratch unit of instruction has been completed in their classrooms. The students enjoyed Scratch and are now creating their own projects on their own after they complete their daily assignments.

7.2 Kauffman Kids Presentation

The Kauffman Scholars program attempts to bolster university enrollment and graduation rates among urban youth through rigorous programs that begin in grade 7 and continue all the way through their senior year in a university. Two of the authors conducted a hands-on Scratch session with 48 Kauffman Scholars. The students were led through a series of small exercises involving Scratch. It was manifestly clear, based on the reactions of the students, that they greatly enjoyed using Scratch, and were highly motivated to complete the exercises that they were assigned. While anecdotal, it confirms the impressions of others - Scratch is straightforward enough that students can begin to use it with literally 1 minute of training, but is capable of creating elaborate programs that captivate student interest.

7.3. Summer Scratch Workshop for K-12 Teachers

During the summer of 2010, Northwest Missouri State University's Computer Science/Information Systems department offered a free one day (seven hours) workshop, "Scratch: The Workshop", to forty-five elementary and secondary teachers. The workshop was geared to teach K-12 teachers how to utilize Scratch to teach computer programming in a context that would allow students to create stories, animation, games, work with music and art.

The group of teachers attending our Scratch workshop was evenly split between elementary, middle school and secondary teachers. The teachers were from a variety of schools, with approximately 75% from rural schools, 15% suburban schools, and 10% urban schools.

Michele Starke, a third grade teacher at Eugene Field Elementary in Maryville said, "The Scratch workshop provided time and expertise in working with designing computer programming for the inexperienced like me. The objectives and instruction were well-presented with guided practice and peer tutoring throughout the day. The hands-on learning was, as we all know, the best way to do this workshop." Starke says that she hopes to incorporate the software into an afterschool program and eventually daily classroom activities [13, p. 1]

Teachers attending the Scratch workshop completed a pre-workshop survey and post-workshop survey. As Tables 1-3 clearly indicate, their attitudes toward computing were positively influenced by their experiences at the Scratch workshop.

Table 1: Students Capable of Writing Programs?

Responses	Pre Workshop	Post Workshop
No of responses	41	44
Yes	23 (56%)	40 (91%)
No	2 (5%)	1 (2%)
Don't Know	16 (39%)	3 (7%)

Table 2: How likely to incorporate these workshop materials into courses?

Responses	Post Workshop
No of responses	44
Very likely	39 (66%)
Somewhat likely	8 (18%)
Likely	7 (16%)
Not very likely	0 (0%)
Not at all	0 (0%)
Don't know	0 (0%)

Table 3: Motivated to learn more about teaching computer science topics?

Responses	Pre Workshop	Post Workshop
No of responses	41	44
Strongly agree	17 (42%)	22 (50%)
Agree	22 (58%)	19 (44%)
Neutral	1 (2%)	3 (7%)
Disagree	1 (2%)	0 (0%)
Strongly disagree	0 (0%)	0 (0%)

8. CONTEXTUAL LEARNING ELEMENTS

Many students are not able to make a connection between what they are learning and how the particular knowledge may be used. This is true in subjects, such as math and science where students often learn information in a particular context and do not know how to apply this knowledge to real world problems. Anderson, Rederm & Simon [3] provide examples of children who were able to perform a math task in a classroom environment and were not able to recognize that this same skill could be applied to a shopping, cooking, or work environment. Contextual or situational learning, which strives to provide a context in which to learn information, comes to the rescue.

Teachers who try to make a connection between new information (knowledge) and the application of this new knowledge, help students make sense of this new knowledge using their own frame of reference, such as previous experiences, memories or particular responses. This approach to learning allows a student to integrate the new knowledge into their existing knowledge and make sense of the relationship between the new knowledge

and existing knowledge. Students are then able to make sense of abstract ideas and concepts and internalize them into real world applications of this knowledge. They are able to develop a complex understanding of the concept in relationship to their social, physical, cultural, and psychological aspects of learning.

In short, learning occurs when learners process information in ways that make sense to them using their own set of references, such as past experiences, or their own memory of knowledge. This learning approach assumes that learners will search for relationships between the new knowledge and their current environment. Therefore, contextual or situation learning forces teachers to incorporate many different experiences for the learner with many experiences. Lee and Shute [15] propose "an integrated perspective that students' personal factors of the domains of behavior, affect, attitude and cognition as well as their social-contextual environment have to work in concert to produce optimal school performance" [15, p.1].

9. WILL SCRATCH WORK AS AN INTRODUCTION TO COMPUTING?

Scratch is an easy to learn programming environment that is ideally suited for teaching computer programming concepts to elementary and high school teachers, who can then pass on this knowledge to their charges. Students who use Scratch become more adept at thinking creatively, problem solving, communicating their ideas through games and stories, and applying concepts in the fields of math, science, art, history to a problem. Research has demonstrated that elementary and high school teachers were motivated to learn Scratch and that Scratch actually changed the attitudes of pre-service teachers and active teachers. We strongly encourage colleges and universities to use Scratch in a variety of education settings to promote computing to elementary and secondary audiences.

REFERENCES:

- [1] ACM K-12 CS model curriculum, 2nd edition: Preparing young people to excel in computer science (2006). Retrieved from <http://csta.acm.org/Curriculum/sub/CurrFiles/K-12ModelCurr2ndEd.pdf>
- [2] ACM K-12 Task Force Curriculum Committee. (2003) A model curriculum for K-12 computer science. Retrieved from <http://csta.acm.org/Curriculum/sub/CurrFiles/K-12ModelCurr2ndEd.pdf>.
- [3] Anderson, J.R., Rederm L. M. & Simon, H. A. (May, 1996). Situated learning and education. *Educational Researcher*, 25(4), 5-11. doi: 10.3102/0013189X025004005
- [4] Bruckman, A., Biggers, M., Ericson, B., McKlin, T., Dimond, J., DiSalvo, B., Hewner, M., Ni, L. & Yardi, S. (2009). Georgia computes!: Improving the computing education pipeline. Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09), 86-90. doi:10.1145/1508865.1508899

- [5] By the numbers. The National Center for Women & Information Technology. Retrieved on July 8, 2010, from <http://www.ncwit.org/pdf/BytheNumbers09.pdf>
- [6] CSTA national secondary computer science survey: Comparison of 2005, 2007 and 2009 survey results. (2009). CSTA Computer Science Teachers Association. Retrieved from http://csta.acm.org/Research/sub/Projects/ResearchFiles/CSTASurvey05-07_09Compar_DCarter.pdf
- [7] Curriculum Frameworks - Preface (October, 2008). Missouri Department of Elementary and Secondary Education School Improvement. Retrieved from <http://dese.mo.gov/divimprove/curriculum/frameworks/preface.html>
- [8] DeClue, T. (2008) Computer science in kindergarten? Of course!: the ABC'S of the K-12 CSTA model curriculum in computer science. *Journal of Computing Sciences in Colleges*, 23(4), pp. 257-262.
- [9] Ericson, B., Guzdial, M. & Biggers, M. (2007). Improving secondary CS education: progress and problems. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, 298-301. doi:10.1145/1227310.1227416
- [10] Fesakis, G. & Serafeim, K. (2009). Influence of the familiarization with "Scratch" on future teachers' opinions and attitudes about programming and ICT in education. *ACM SIGCSE Bulletin*, 41(3), 258-262.
- [11] Frost, D. (2007). Fourth grade computer science. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*, 302-306. doi:10.1145/1227310.1227417
- [12] Heisserer, D. L. & Parete, P. (2002). Advising at-risk students in college and university settings. *College Student Journal*, 36(1), pp. 69-83.
- [13] Hornickel, M. (2010, July 130. Workshop helps area K-12 teachers learn benefits of Scratch software. Northwest Missouri State University. Retrieve from <http://www.nwmissouri.edu/universityrelations/news/newsreleases/100715scratchworkshop.htm>
- [14] Kelleher, C & Pausch, R. (June, 2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Survey*, (37) 2, 83-137 doi:0.1145/1089733.1089734
- [15] Lee, J & Shute, V. (2010) Personal and social-contextual factors in k-12 academic performance: An integrative perspective on student learning. *Educational Psychologist*. 43(3). 1-19. doi: 10.1080/00461520.2010.493471
- [16] Marcu, G., Kaufman, S., Kate Lee, J., Black, R., Dourish, P., Hayes, G. & Richardson, D. (2010). Design and evaluation of a computer science and engineering course for middle school girls. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education (SIGCSE '10)*. 234-238. doi:10.1145/1734263.1734344

- [17] Murphy, L., VanDeGrift, T., Richards, B. & Wilson, B. (October, 2005). Models for computer science K-12 outreach activities. *Journal of Computing Sciences in Colleges.* (21) 1, 274-276.
- [18] Resnick, M., Malone, J., Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, a., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y. (November, 2009). Scratch: Programming for all. *Communications ACM,* (52) 11, 60-67. doi:10.1145/1592761.1592779
- [19] Rodger, S., Hayes, J., Lezin, G., Qin, H., Nelson, D., Tucker, R., Lopez, M., Cooper, S., Dann, W. & Slater, D.. (2009). Engaging middle school teachers and students with Alice in a diverse set of subjects. *Proceedings of the 40th ACM Technical Symposium on Computer Science Education (SIGCSE '09).* ACM, New York, NY, USA, 271-275. doi:10.1145/1508865.1508967
- [20] Werner, L., Denner, J., Bliesner, M. & Rex, P. (2009). Can middle-schoolers use Storytelling Alice to make games?: Results of a pilot study. *Proceedings of the 4th International Conference on Foundations of Digital Games (FDG '09),* 207-214. doi:10.1145/1536513.1536552
- [21] Wilson, C., Sudol, L.A., Stephenson, C. & Stehlík, M. (2010). Running On Empty: The Failure to Teach K-12 Computer Science in the Digital Age. The Association for Computing Machinery & The Computer Science Teachers Association. Retrieved from <http://csta.acm.org/runningonempty/fullreport.pdf>
- [22] Wong, A. (2010). Scratch: A creative programming tool for kids. Retrieved from <http://launchpadtoys.com/blog/2010/06/ltp-review-scratch/>
- [23] Zweben, S. (2010). Computing degree and enrollment trends from the 2008-2009 CRA Taulbee survey. Retrieved December 12, 2010, from <http://www.cra.org/uploads/documents/resources/taulbee/0809.pdf>

MULTI-STEP PROBLEM SOLVING USING SCRATCH: A PRELIMINARY REPORT*

*William Siever, Linda Heeler, Phillip Heeler
Northwest Missouri State University
Maryville, MO
660-562-1600
 [{siever,lheeler,pheeler@nwmissouri.edu}](mailto:siever,lheeler,pheeler@nwmissouri.edu)*

ABSTRACT

The researchers have conducted weekly after-school computer laboratory sessions with fifth and sixth graders using visual programming languages and both robots and a virtual environment. Following several individual exercises in the virtual environment, teams of students were formed and asked to perform a multi-step task in both the virtual environment they work with regularly and an unfamiliar version of the robot environment. Upon completion of the tasks it was clear that the students have a strong grasp of multi-step problems and are able to identify the salient steps of the multi-step problem and translate it to the two different environments.

INTRODUCTION

Computer programming with middle school students has evolved from the days of text based programming languages like Applesoft BASIC and Logo to visual programming languages like Alice and Scratch. The researchers have conducted weekly after-school computer laboratory sessions with fifth and sixth graders using robots with Lego Mindstorms and the Scratch programming environment, both of which use visual programming languages. This paper describes the preliminary results of multi-step problem solving activities that the researchers have carried out with sixth grade students using both Scratch and Lego Mindstorms.

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

BACKGROUND

The Horace Mann Laboratory School at Northwest Missouri State University has an enrollment of over 200 students in grades Pre-Kindergarten through sixth grade. Currently, there are 23 students in the fifth and sixth grade class, all taught by the second author.

During the 2007-2008 school year, a robotics project was initiated by selecting seven fifth and sixth grade students to participate in an after-school computer club. The students used Lego Mindstorm kits and completed various exercises under the guidance of the three faculty members, Bell, Heeler and Heeler.[1]

During the 2008-2009 school year the researchers decided to divide the fifth and sixth grade students into two groups. The fifth grader students used the Lego Mindstorm kits to complete exercises with robots. The sixth grade students, having already worked with the robots for a year, were ready for a different computer experience. The same three researchers decided to expose these students to the Alice development environment.[2]

During the 2009-2010 school year, the researchers continued the after-school computer club with all of the available fifth and sixth graders. The fifth graders completed exercises with the Lego Mindstorm kits as before. However, based upon the experiences of the sixth graders during the previous year, the researchers decided the Alice environment was too complicated for the sixth graders. An alternative environment, called Scratch, was used. Scratch also provides a visual programming language, but it is much simpler than Alice [5].

CURRENT STATUS

During the 2010-2011 school year the researchers have continued the weekly after-school computer club with the fifth graders using the Lego Mindstorm kits to investigate computer programming. The sixth graders have started to learn the Scratch programming language. Various projects, which typically require the students to implement interactive stories, emphasize specific programming concepts such as sequences, iteration, conditional statements, multiple processes/threads, synchronization, Boolean logic, event handling, and user interface design.[4]

A partial list of the programming projects that the students completed is described below.

1. After learning how to utilize the Scratch environment and learning a few programming statements, the students were asked to create a Total Turtle Trip project to make the Scratch cat character trace a polygon with a specified number of sides. This required looping and basic geometry.
2. The students were asked to implement a Scratch animation of the popular nursery rhyme "Hey diddle diddle, the cat and a fiddle, the cow jumped over the moon." The story was described as a list of several steps and the students needed to decompose the steps into a sequence of program statements and control the actions of multiple characters.

3. The students were asked to implement a Scratch animation of the "Red Rover" game, where each student's picture was pasted onto a sprite and then moved across the screen.
4. The students were asked to create a Halloween scene with animated objects moving across the screen. One goal of the project was the introduction of inter-process communication and event driven programming by requiring the characters in the story interact using basic message passing.
5. The students were asked to create a Thanksgiving scene with animated objects moving across the screen.
6. The students were asked to create a Holiday scene with animated objects moving across the screen.

RESEARCH PROJECT

As the semester continued, the sixth graders were quickly gaining experience with the Scratch programming environment and becoming more sophisticated in designing their projects. These sixth graders had already learned how to program in a visual environment using the Lego Mindstorms robots the previous year. Based upon those experiences and the researchers' interest in multi-step problem solving, a project was designed that would ask these students to perform two different problem solving tasks.

The two tasks differ in two aspects. First, they are performed in different, but comparable, visual programming languages, namely Scratch and Lego Mindstorms NXT. Second, one task is performed by a character in Scratch's perfect, virtual environment while the other is performed with real robots. Unlike the Scratch environment, the robots used didn't respond to geometric constructs like "turn 90 degrees clockwise". Instead the students had to identify parameters that achieved the required movement. For example, the student's needed to perform tests to see if turning a motor on for 5 seconds was sufficient to turn 90 degrees.

The nine sixth grade students were divided into three groups of three students each and were given instructions to work on two projects, one with robots and one with Scratch. For two of the teams, the Cardinals and the Bluebirds, the first project was to write a program to have their robots trace a large block letter 'L' on the classroom floor. The students were given specific instructions to first analyze the task, then to design a solution, and then to write a program using the Lego Mindstorms NXT environment. Although these students had worked with Lego Mindstorms RCX RoboLab the prior year, the NXT language and environment was new to them.

By requiring the teams to start with analysis and design they were forced to verbalize their approach, which helped elicit the appropriate angles needed to trace the block 'L'. Upon completion of the program, they alternated between testing their work and adjusting parameters. Both the Cardinals and the Bluebirds successfully completed the task of tracing a block 'L' over the course of two one-hour sessions on successive weeks.

After finishing the first part of this project the Cardinals and the Bluebirds were asked to repeat the same task in the Scratch environment, which both teams completed

in only a few minutes. Although this isn't surprising, as the students had been working with Scratch the entire semester, it confirmed that both teams had identified the salient steps of the process and could apply a known multi-step solution in Scratch. In addition, both teams were able to achieve the task with very little testing.

The other team, the Robins, was given the same two tasks but in the opposite order. This team quickly analyzed, designed, wrote, and tested their Scratch program to trace the letter 'L' on their computers. This task was completed in just one hour-long session. The following week the Robins started on the second part of their project: to design, write, and test a program to trace the letter 'L' on the classroom floor. This task took the Robins team two weeks to complete following on the success of the Cardinal and Bluebird teams.

THEORETICAL ANALYSIS

Rachel Wing DeMatteo answers the question of "how do you bring problem solving to the classroom?" with the following three recommendations. First, ask students to generate solutions that can be shared and generalized. Second, encourage students to use a variety of media to express their solutions. And third, require students to problem solve with their peers.[3]

The above described research project is an example of a viable problem solving exercise using Scratch and robots that answers the question of how to bring problem solving to the classroom. First, the students worked as teams to generate solutions that had to be generalized to the two different environments, the virtual environment in Scratch and the real robots. Second, the students expressed their solutions to the problems both verbally and in writing as part of the problem analysis, and then as a graphical program during the implementation phases on both the Scratch and NXT environments. Third, the students solved this exercise with their peers, including team members as well as interaction across teams.

The current Scratch computer science education literature repeatedly mentions the 21st Century Learning Skills.[6] Critical thinking and problem-solving skills are emphasized with special attention paid to systems thinking; problem identification, formulation and solution; and creativity and intellectual curiosity. The above-described research project directly includes these specific activities as the students were asked to identify the problem, formulate a solution, and include creativity in their solutions.

The success of these student teams was not a surprise for the researchers. Each spring these students take the Stanford Achievement Test that measures, among other things, their problem solving abilities. When they were in fifth grade, the current sixth graders scored at the 86 percentile compared with students in the same grade who took the test at a comparable time. The stanine score of 7 is two standard deviations above the mean.

CONCLUSIONS

For the nine students involved, the exercise was successful because of their problem solving skills that had already been developed from previous academic activities. The

collection of Scratch exercises described above was designed to further develop their problem solving abilities through technology including robot programming and Scratch programming. The researchers realize that this one exercise is not enough to enhance problem-solving skills for all students, but it did demonstrate the skills developed from prior work in comparable environments quickly translated to a new environment. Consequently, the students were able to quickly identify the salient steps of the multi-step problem and translate it to the two different environments. Now these students are ready for more sophisticated problem solving activities whether it be in mathematics, science, social studies or other academic or non-academic areas.

REFERENCES

- [1] Bell, S., Heeler, L., and Heeler, P., A preliminary report on the use of robots with elementary school students, *Journal of Computing Sciences in Colleges*, 23, (4), 263-268, 2008.
- [2] Bell, S., Heeler, L., and Heeler, P., Computer programming with middle school students, *Journal of Computing Sciences in Colleges*, 24, (5), 161-165, 2009.
- [3] DeMatteo, R.W., Informing practice: a model approach to problem solving, *Mathematics Teaching in the Middle School*, 16, (3), 132-135, 2010.
- [4] Lifelong Kindergarten Group, MIT Media Lab., Programming concepts and skills supported in Scratch:
<http://www.learninggameslab.org/documents/program-concepts-day-1.pdf>, retrieved December 13, 2010.
- [5] Mesnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Grennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y., Scratch: programming for all, *Communications of the ACM* 52,(11), 60-67, 2009.
- [6] Rusk, N., Resnick, M., and Maloney, J., Learning with Scratch 21st century learning skills.
<http://llk.media.mit.edu/projects/scratch/papers/Scratch-21stCenturySkills.pdf> retrieved December 13, 2010.

EMBEDDED AND REAL-TIME PROGRAMMING WITH ADA*

CONFERENCE WORKSHOP

*John W. McCormick
Computer Science Department
University of Northern Iowa
Cedar Falls, IA 50614-0507
319-273-6056
mccormick@cs.uni.edu*

ABSTRACT

This workshop provides instructors with knowledge that can help to bring their students with traditional backgrounds in sequential programming up to speed in the embedded real-time domain.

INTRODUCTION

As a result of the availability of extremely low cost microprocessors, actuators, and sensors the world is awash in new devices whose primary use is not traditional computing. Much of the software running on these embedded processors has timing deadlines. The consequences of missing a deadline range from sluggish sales to the loss of life or property. Some think that simply writing "fast code" or upgrading to a faster processor solves all our timing problems. The true solution is predictable code and proof that this code can meet required deadlines under worst case conditions.

Unfortunately, very few undergraduate computer science programs address the special issues involved in the specification, design, implementation, testing, or verification of real-time embedded software. As result, most embedded software is developed by electrical engineers with a course or two in C programming. As computing hardware continues to decline in price and increase in power, applications we had never dreamed of in the past are now possible. The inevitable increase in software complexity requires development team members with skills beyond basic C programming - the skills of a computer science graduate.

* Copyright is held by the author/owner.

WHY ADA?

Some may question the decision to use Ada as a vehicle for teaching real-time embedded systems. The short answer is the success of students in undergraduate real-time embedded systems classes over the past 25 years. Since 1984 students have developed the software to monitor and control a large model railroad. The minimum project requirements have remained the same throughout this time. The software developed by student teams contains over 10,000 lines. For the six years in which the C language was used, none of the student teams successfully completed the minimum requirements of the train project. With Ada, over 80% of the teams successfully complete the minimum project requirements. While the numbers of students attempting to implement the control software in C++ or Real-Time Java is too small to be statistically significant, no student has been successful with either of those languages. Using Ada in a teaching environment makes sense.

Two Ada myths stated by reviewers of the original proposal for this workshop are "... limited utility of the Ada language outside of DoD ..." and "Ada is not a very popular language in real-time and embedded systems due to its size and complexity." In fact, over the last decade the number of commercial Ada projects is nearly triple that of military products. Based on the size of the language standards and the size of reference books that cover the complete features and libraries of each language, Ada is significantly smaller than C++ and Java. Ada is about the same size as C when you include all of the specialized libraries needed to develop real-time embedded software in C. The success of students using Ada suggests that it is also less complex than C, C++, and Java. The work currently being done by the ISO / IEC JTC 1 / SC 22 Working Group 23 on Programming Language Vulnerabilities supports this suggestion. The difficulty of students using C to develop train control software was recently corroborated by Michael Barr, editor of Embedded Systems Design magazine. He and a small team of embedded systems experts developed a multiple choice quiz to access C knowledge pertinent to the embedded domain. He found that the average grade for the over 4,500 engineers whose primary languages are C and C++ was a pitiful 61%. Such a lack of understanding demonstrates the complexity of C/C++ semantics. While only a small percentage of the world's software is developed in Ada, there is a demand for graduates with Ada skills. This demand has seen a linear growth over the past decade.

WORKSHOP TOPICS

Most of the material for this workshop is drawn from the presenter's textbook, *Building Parallel, Embedded, and Real-Time Applications with Ada*, published by Cambridge University Press, 2011.

Why teach real-time embedded systems?

- 25 years of evolution of a successful course and laboratory
- Characteristics of real-time embedded systems
- The Ada type model - *In Strong Typing We Trust*
- Embedded architectures
- High level support for low level programming
- Manipulating bits - control over representation of types

- Device drivers
- Why use multiple threads of control?
- Defining tasks
- The lifecycle of a task
- Task hierarchies
- Task communication and synchronization
- Asynchronous events - processing interrupts
- Characteristics of tasks relevant to real-time scheduling theory
 - Characterization of Tasks
 - Scalar properties of tasks
- Real-Time schedulers and feasibility testing
 - Fixed priority scheduling
 - Dynamic priority scheduling
 - Priority inheritance protocols for shared resources
- Ada constructs for compliance with real-time scheduling theory
 - Expressing time
 - Implementing periodic tasks
 - Implementing priority inheritance protocols
 - Scheduling models

FIVE FOCUSED STRATEGIES FOR INCREASING RETENTION IN COMPUTER SCIENCE 1*

Tim DeClue, Jeff Kimball, Baochuan Lu, James Cain

Department of Computer and Information Sciences

1600 University Avenue

Southwest Baptist University, Bolivar, MO 65613

(417) 328-1704

tdeclue@sbuniv.edu, jkimball@sbuniv.edu, blu@sbuniv.edu, jcain@sbuniv.edu

ABSTRACT

Increasing retention of students in computer science—particularly in the first course—is a frequently studied and much discussed topic in the discipline. This paper presents a multi-faceted approach which is practical and which can be generalized to small college settings. In particular, the authors describe strategies for building a sense of community among the students, increasing opportunities for programming practice and success, adopting a high-quality supporting software set, and making use of motivating assignments. As a result, 95% of students (128 of 134) who began the course in the fall semester for the years 2005-2009 persisted to the point of receiving a final grade.

INTRODUCTION

A great deal of effort has been focused on increasing student retention in computer science. Clearly, the effort has significance. Retaining talented students is critical to meet industry demand [2]. Retaining talented students helps retain talented faculty. Retaining larger numbers of talented students contributes to the overall health of a department and helps at budget setting time. Other positive effects could be cited as well. Further, if there is a rationale for specific retention strategies, it makes sense to implement them as early in the curriculum sequence as possible, ideally in CS1 (computer science 1). Retaining younger students—especially freshmen—will lead to more upper-classmen

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

than retaining sophomores or juniors, so it makes sense to focus retention efforts on the first course.

Strategies cited in the literature include reducing the amount or type of programming in CS1 [9], emphasizing study skills [3], including particular educational approaches like active or group-based learning [4, 5], and implementing strategies to address poorly prepared students or poorly taught courses [1].

Between the fall of 2005 and the fall of 2009, the authors' university achieved over a 95% retention rate (134 students started the course and 128 completed) for students in CS1. Retained students is defined to be that set of students who were enrolled in CS1 on the census date for the university and who persisted to the point of receiving a final grade for the course. Retained students includes students who completed the course but received a failing grade.

Upon reflection, the authors could not cite a specific, singular practice which could account for the high retention rate. Rather, the authors cited several strategies which have found their way into the course over this period of time. This paper seeks to describe the strategies for the common benefit of all who teach CS1, particularly in the small college setting.

STRATEGY #1: BUILD A SENSE OF COMMUNITY

Several course and departmental practices are used to build a sense of community within the first course. Strategically, this effort at building community affects retention in at least three desirable ways; by encouraging positive relationships, through increasing feelings of connectedness with other students, and by fostering a sense of belonging for the students in the course.

Community Through Collaborative Learning

Collaborative learning strategies are implemented in the course to both promote learning and to encourage positive relationships within the course. The research-based educational value of collaborative/cooperative learning is now fairly well known. According to Joyce, Weil, and Showers, for example, "...classrooms organized so that students work in pairs and larger groups, tutor each other, and share rewards are characterized by greater mastery of the material than the common individual-study cum recitation pattern. Also, the shared responsibility and interaction produces more positive feelings toward tasks and others..." [5]. Examples of cooperative learning include pair programming, peer tutoring, table answers (negotiated answers produced during class by a single table of students), peer code reviews, and checking each other's work in non-graded assessments. Besides having a research-based effect of increasing learning in the cognitive domain, collaborative learning has also been shown to address affective learning by helping increase students' enjoyment of the learning process [4].

Community Through Service Projects

For approximately the past five years at the authors' university, CS1 students have participated in a service project together. Three of the years, the students adopted a three hour period of time in which to ring the Salvation Army bell at a local grocery store. Students signed up for 30-minute segments and generally there were 4-6 students available at any one time. While ringing the bell (or instead of it), the students sang Christmas carols and songs in front of the store. In 2009, a local food pantry needed help moving food and furniture to a new location and the CS1 students spent a Saturday morning helping the food pantry complete the move. In 2010, students participated in both bell-ringing and a program to help needy children through a Christmas food and toy distribution program.

The positive effects of group service projects are well-known and research is not cited here. Frequently cited effects, however, include feelings of greater self-efficacy, higher academic achievement, enhanced problem-solving skills and better planning abilities. It seems natural that such activities would increase the level of connectedness that students feel for each other simply due to the effect of a positive shared experience.

Community Through Group Identification

Clothing is used by a wide variety of groups to build a sense of community ranging from the military to schools, churches, athletic teams, even gangs. Few would dispute the sense of community high school and college students exhibit by wearing t-shirts that attest to their common experience of attending a rock concert.

Since the fall of 2006, it has become tradition to order T-shirts for students majoring or taking classes in the Department. The t-shirts are sold at cost to the students. Even though only 50% to 70% of students in the Department ever purchase the t-shirts, this number seems well above the critical threshold to be a recognized group on campus. The t-shirts become a sort of "badge of honor" which promotes a sense of belonging and identification. It is well-known many students have difficulty transitioning from the high school setting to the college setting because they do not feel a sense of belonging in their new home. The use of a t-shirt to symbolize their belonging to a new group can help ease this transition.

STRATEGY #2: LEVERAGE SOFTWARE TO SUPPORT LEARNING

Six useful software tools-in addition to ones normally found in a classroom setting-were used in the fall semester of 2010 to increase student learning. These tools were not consistent across all years, but they do represent the sort of toolset used in the course from year to year.

- The Angel Learning Management System (Blackboard.com), is used to manage the course, and to host videos of class sessions from previous semesters.
- Netbeans (Netbeans.org), a Java IDE sponsored by Oracle, was used as the integrated development environment for the course.
- Gliffy (Gliffy.com) is a free web-based drawing tool the students and faculty used to produce UML class diagrams.

- Zoomit (Microsoft.com) was used to assist in the examination of code and text when projected on the screen in front of a classroom.
- Irfanview (Irfanview.com), a free image viewer that allows students to convert images from one format to another, was used in two of the programming assignments.
- Codebat (Codingbat.com), an online practice coding website, was used to provide additional opportunities for coding practice.

STRATEGY #3: SUPPORT AUTOMATICITY THROUGH PRACTICE

Most learning theory accepts the idea of automaticity in learning, that is "...controlled processes [will] become increasingly automatic through repetition and practice" [7, p. 281]. To a certain extent, some of the programming skill students are expected to achieve in CS1 benefits from repetition and practice. For example, the sooner it is that students achieve automaticity with the basics of software development the faster they can move on to higher-order learning. Included in the basics would be working with an IDE, the more mundane aspects of programming language syntax and certain common flow constructs like loops and if statements. To this end, the faculty have instituted several opportunities for coding practice beyond the minimum required by the syllabus for the course.

Engaging in competitive programming has become a tradition for some students in the department. For the years covered in this paper, the competitive programming teams-along with one or more faculty coaches-have practiced every Monday evening for the first eight to ten weeks of each semester. The students enrolled in CS1 are invited to attend these practice sessions and to be prepared to work on class assignments while the competitive programmers are problem-solving. This serendipitous arrangement has allowed beginning computer science students an extra two hours of lab/instruction time per week.

Additionally, faculty have encouraged the use of Codebat (described earlier) by including its use in CS1 labs as enrichment activities and by basing test questions on problems found at the website. This practice website is available 24 hours a day, and can be a valuable tool to practice writing short and targeted code segments.

Learning theory strongly supports this type of practice to achieve success in learning. According to the Carnegie Mellon Center for Teaching Excellence, for example, "Learning and performance are best fostered when students engage in practice that focuses on a specific goal or criterion, targets an appropriate level of challenge, and is of sufficient quantity and frequency to meet the performance criteria" [10]. In other words, students will get better at whatever they practice, including writing software.

STRATEGY #4: USE ASTRACHAN'S LAW AND CONTEXT IN PROGRAMMING ASSIGNMENTS

Students who see no point to small, simple programming assignments may lack motivation to complete the assignment no matter how valuable the lesson. This observation is particularly true for the most talented problem-solvers in the class. For this

reason, Astrachan's Law [8] is followed when programming assignments are made. According to Astrachan, you should "...not give an assignment that computes something that is more easily figured out without a computer..." (p. 26). Astrachan's point is clear. If students believe using a computer makes problems harder to solve, students will not view the required computation with favor and motivation will sink. Assignments, therefore, are generally chosen if some aspect of solving the problem requires significant computation unavailable without using a computer. Even assignments given during the first week of classes are designed with Astrachan's law in mind.

Assignments in the course also rely on context to provide meaning and motivation. The principles taught from semester to semester have remained fairly stable, but the contexts into which the principles are placed have been updated each semester to have the greatest effect. For example, simple string assignments have been described in terms of texting, tweeting, or even online textbook ISBN numbers; file input/output problems have become rudimentary digital fingerprint analysis, and array-based linear searches have become red eye reduction in images.

STRATEGY #5: USE CARROTS

It is said that horses can be motivated to continue walking forward as long as a carrot is dangled just beyond the reach of the horse's mouth. Although students are not horses, it can be helpful to save an inherently motivating activity for the end of a learning unit or course. Looking forward to these "carrot" activities can have the effect of keeping students in the course until the activity is reached.

Recursion As A Carrot

Recursion is normally a topic covered in CS1 and can lead to some fun-try Googling "recursion" or look up the term in Wikipedia, for example. At times, however, the examples provided to the students portray recursion more as a mechanistic "parlor trick" than as a truly usefully approach to certain categories of problems. The parlor trick examples include the classics like n-factorial, towers of Hanoi, and the fibonacci sequence. While useful-these examples are still elegant and continue to be included in the course-none of these examples demonstrate the true power of recursion to solve certain difficult and highly intractable categories of problems.

Students who do not grasp the power of recursion are less likely to learn how it works and see its appropriate application. Lacking this knowledge, students can become confused, unmotivated, and consider quitting. For the semesters described in this paper, a particularly good example of recursion-storing a binary search tree in an array-has been used as a carrot to end the instruction on recursion.

First semester students normally have all of the necessary background to understand binary search trees except a thorough understanding of dynamic memory allocation. The basic idea for the array-based binary search tree is straightforward: store the root at location 0, left children at the parent's index*2+1, and right children at the parent's location*2+2.

By storing a binary search tree in an array, the students can implement a recursive insertion and recursive retrieval without the need for dynamic memory allocation. This example, while simple, becomes a powerful symbol for algorithms that are able to achieve $O(\log n)$ efficiency, certainly a pragmatic reason to use a recursive approach.

Robotics As A Carrot

In the fall of 2007, the authors began using robots and an introduction to the Python programming language as a way to finish the last two weeks of CS1. It was acknowledged that some loss of traditional course content might occur, but that this loss might be outweighed by the motivational aspect of using robots as a kind of carrot for staying in the course. As further justification, the authors point out that the last week of a traditionally-taught class often includes an introduction to topics covered in CS2 or content for which no programming assignments can be made due to the proximity to the end of the course. At most, then, only about one week of the traditional CS1 course content was lost and this loss of traditional content was deemed acceptable if replaced with an introduction to a second language and robots.

Not every school can use robots in CS1 of course, but this fact does not preclude the use of something at the end which is both different and inherently motivating. Principles of gaming and game design, or perhaps even a short course on writing Facebook apps could serve as carrots as well.

SUMMARY

Retention continues to be a challenging issue in computer science, especially with novice students enrolled in CS1. While the "silver bullet" solution continues to elude faculty who teach the course, the multi-faceted approach described in this paper which includes principles of learning theory, principles of group identity, and good teaching practices seems to produce results worthy of further study.

REFERENCES

- [1] Beaubouef, T. and Mason, J., Why the high attrition rate for computer science students: some thoughts and observations, *SIGCSE Bulletin*, 37 (2), 103-106, 2005.
- [2] Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook, 2010-2011 Edition*, Computer Software Engineers and Computer Programmers, <http://www.bls.gov/oco/ocos303.htm>, (visited November 27, 2010).
- [3] Gathers, E., One freshman studies program which improved student retention in the first year computer science sequence for majors, *Proceedings of the 1988 ACM Sixteenth Annual Conference on Computer Science*, ACM, New York, NY, USA, 739-.

- [4] Guo, J., Using group-based projects to improve retention of students in computer science, *Journal of Computing in Small Colleges*, 23 (6), 187-193.
- [5] Joyce, B., Weil, M., Showers, B., *Models of Teaching 4th Edition*, Boston, MA: Allyn and Bacon Publishing, 1992.
- [6] Norfleet, W., Impact of active learner involvement on achievement and retention, *Ph.D. Dissertation*, Wayne State University, Detroit, MI, USA. Advisor(s) Rita C. Richey, 2008.
- [7] Ormrod, J. E., *Human Learning: Theories, Principles, and Educational Applications*, New York, NY: Merrill Publishing, 1990.
- [8] Parlante, N., Astrachan's Law, *SIGCSE Bulletin*, 35 (4), 26-27, 2003.
- [9] Turner, Elise H., Albert, Erik, Turner, Roy M., & Latour, Laurence, Retaining majors through the introductory sequence, *SIGCSE Bulletin*, 39 (1), 24-28, 2007.
- [10] Office of Technology for Education, Eberly Center for Teaching Excellence, Theory and research-based principles of learning, <http://www.cmu.edu/teaching/principles/learning.html>, (visited November 27, 2010).
- [11] Urness, T., Teaching file input/output, loops, and if-statements via a red eye reduction assignment, *Journal of Computing in Small Colleges*, 23 (4), 286-289, 2008.

WORKING TO CHANGE PERCEPTIONS*

*Ken T. N. Hartness, Ph.D.
Sam Houston State University
936-294-3524
hartness@shsu.edu*

ABSTRACT

Students are avoiding high school and college courses that would prepare them for careers in technology, science, and engineering. Some of this behavior comes from the advice of authority figures. The authors describe some of their efforts to address misconceptions held by high school students, teachers, and counselors and how those efforts were received. We share some of what we have learned from teachers and counselors and offer suggestions on further actions that are being taken or could be taken by computer science educators to ensure that the needs of industry and an economy reliant on reliable information and advancing technology are met.

INTRODUCTION

The dawn of a new century was quickly dimmed for computer science academic programs as students decided that the effort involved in mastering computer science was not worth it. In the face of business failures, particularly those related to Internet services, the public formed the perception that careers related to a computer science degree were becoming scarce. Reports of job outsourcing did not improve this perception. However, as the number of local job opportunities fluctuates, the overall needs of industry for technological support and innovation continue to grow. In spite of some improvement in enrollments, recently, industry's needs continue to outstrip the number of graduates being produced by CS academic programs.

Given that women and certain minorities are not well-represented in computer science, it seems likely that a significant population of people capable of mastering computer science is available for recruitment if we can just establish environments and incentives that will attract them. A first step towards achieving this end is reversing

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

certain misconceptions that have arisen about careers available for computer science majors.

STUDENT PERCEPTIONS

According to Yardi and Bruckman, teenagers reportedly perceive computer science as "boring, antisocial, and irrelevant to their lives" [19]. They enjoy taking advantage of the games and social networks made possible by computer science but show no interest in computer-related careers. Their perception of these careers as being filled with lonely, endlessly detailed work may be combined with an assumption of low job availability to help stifle such interest. Reports of business failures, job reductions and job outsourcing seem more prevalent than reports of a more positive nature. In addition, surveys suggest that under-represented groups often felt excluded from the enthusiasm of the far more common white and Asian male students and felt that they were not welcome [6, 10].

Yardi and Bruckman found in interviews that teenagers rarely had a good idea of what computer science entailed but did have an impression of a lone coder developing, writing, debugging, and maintaining software in isolation [19]. Teenagers expressed a lack of interest in pursuing computer science because they were social and creative and, in some cases, simply assumed they didn't have the ability to handle the complex minutiae of computer programming.

CORRECTING PERCEPTIONS

Why Correct Misperceptions

According to the 2007-2008 report by the Computing Research Association [21], total enrollment in CS programs increased 6.2% between 2007 and 2008. This increase in enrollment, the first reported during the previous six years, is slowing the drop in graduating CS majors that has been seen since several high-profile business failures and reports of job outsourcing helped make the prospects for computer careers appear grim. In contrast, the Bureau of Labor Statistics in the U.S. Department of Labor continues to predict a large growth in computer-related job opportunities [2]. Between 2008 and 2018, software engineering jobs are predicted to increase by 34%, and IT management positions are predicted to increase by 17%. Indeed, the only computer career not predicted to increase by 2018 appears to be the one closest to the career imagined to be typical for computer science majors by social and creative individuals; programmer positions are predicted to decrease by 3%, although turn-over in the over 400,000 jobs should result in plenty of job opportunities for those who want to primarily sit in a cubicle and code. The current increase in enrollment, which must still translate into retained graduates, is insufficient to meet the predicted needs of industry, even if the current employees don't move into management or retire. If a modern economy depends on information technology and technological innovation [15] then insufficient enrollment in technology-related majors has a significance that goes well beyond the health of an academic department.

Industry, academic programs struggling to survive, and, perhaps, the future quality of life in the United States depend on a pro-active effort to increase interest in

technology-related programs. This paper makes the assumption that many students capable of enjoying a rewarding computer-related career are currently pursuing majors they perceive as safer or more rewarding and discusses methods for making those potential computer science majors aware of the possibilities and benefits of a CS education.

Recruiting Authority Figures

In 2009, the author and a co-worker of the university were awarded a grant by the Texas Workforce Commission. The grant included funding for scholarships, paid mentors, and travel stipends as an incentive to high school teachers and counselors to attend workshops at the university. The grant did not cover funding for summer camps for pre-college students, but it did support travel to high schools to make presentations. The workshops were designed to count as continuing education credits that would be recognized by school administrators or applied towards a professional counselor's license. The co-worker and I hoped that the workshops would convince teachers and counselors to help recruit students into technology-related majors, thus allowing us to potentially impact a larger number of youth than we could working alone. Bruckman et al. [1] describe a pipeline that extends back before high school; although our grant does not specifically deal with junior high students, we welcomed any junior high or elementary school teachers who were interested in attending the workshops.

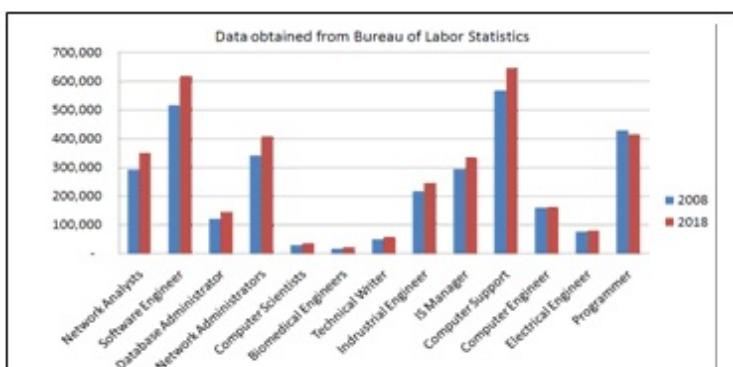


Figure 1 - Predicted Number of Jobs by 2018 [2]

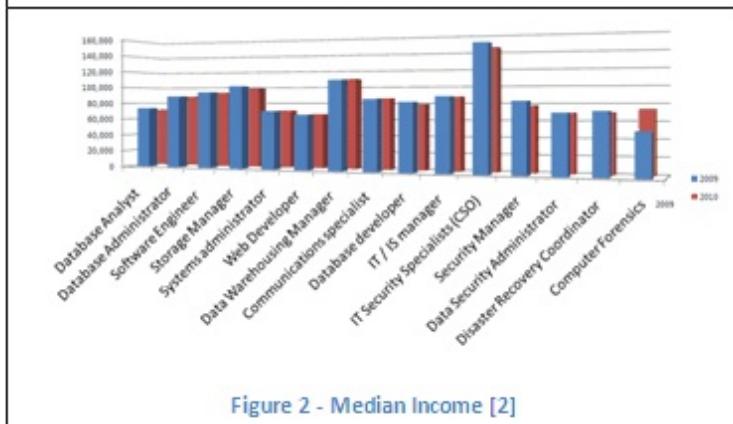


Figure 2 - Median Income [2]

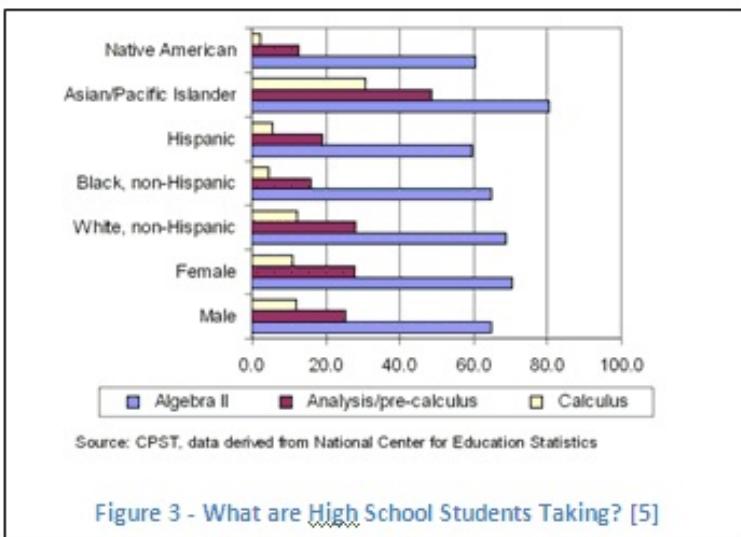
The teacher workshops lasted two days and the counselor workshops lasted one day. They were divided into sessions of one to one-and-a-half hours, although some tutorials were as long as three hours. The sessions covered topics like misconceptions about technology-related careers and a discussion of alternative careers that may appeal to a wider range of personalities. The teacher workshops typically included hands-on activities, like building a computer, using Web 2.0 applications in the classroom, and the use of interesting programming

environments like Scratch [11], Alice [3], and Robocode [13]. The counselor workshops included sessions that discussed the educational needs of students considering different technology-related majors, the qualities expected of software engineers, and a more in-depth coverage of alternative job opportunities.

Of the three workshops held to date, each one has contained at least one attendee who was surprised to learn that there are still job opportunities for students who study computer science. Those majors impacted by the effort to improve students' interest in science, technology, engineering, and mathematics (STEM) should make more of an effort to share information about existing and predicted job opportunities. Indeed, this information should be readily available to parents, not just teachers and counselors, because of the impact of parents' opinions on a student's selection of major. I once helped a student who was obtaining a degree in history with a computer science minor; she told me that her parents had forbidden her to waste her time on a computer science major. The parents need to know that the jobs have not disappeared (figure 1) and pay well (figure 2). The students need to know that these careers need not involve sitting alone staring at a computer all day.

The workshop attendees are shown that even entry-level programmers [7] need to be creative and interact well with members of their team and the intended users of the software. They also need to possess knowledge appropriate for the problems they are asked to solve, suggesting that students whose primary interest is not computers may still want to consider learning how to apply programming skills in their primary area of interest. We describe a typical software engineer as a creative problem solver who needs to be able to work closely with users whose expertise is not computer science in a dynamic and often exciting environment of problem solving. Additionally, the software engineer may need good management skills to work with members of a team, negotiating conflicts and bringing the group together to accomplish a task efficiently.

In addition to the attempt to make software engineering sound far more exciting than high school students and many of their authority figures imagine, a session is held to emphasize that not every CS major must work on a business's accounting software. The computer science department at Sam Houston State University now offers an information assurance track and a graduate degree in digital forensics, and one CS professor is working with a Biology professor on a bio-technology course, so these areas are emphasized among others as alternatives to the traditional CS career. The University of Washington has produced an interesting video that I like to use [16]. Some of their former students talk about their work on sensor networks that help firefighters, on resources to assist education in India, on the translation of images into a tactile format understandable by blind people, etc. The session typically ends with mention of the creativity needed to support the development of methods of human-computer interaction and a brief mention of the music genome project [12] and the Picasso project [8] as a way of emphasizing that an interest in the arts does not necessarily negate the usefulness of a CS degree.



Once a student is convinced to consider computer science or some other major that supports technology, the issue becomes whether the student has the background to succeed in that major. Counselors, in particular, are asked to encourage students to obtain a strong background in math and science if they have any aptitude in that area. At one workshop, a teacher

claimed that technology-related courses were being eliminated due to lack of interest; the reason given was that the students felt the courses were too difficult. Although the majority of high school students take algebra 2, only about one-fourth take pre-calculus and about half of those take calculus [5]. When broken down by ethnic groups, the numbers are significantly lower for under-represented groups (see figure 3). In so far as these numbers are not related to some lack of preparedness caused prior to high school, it should be possible for authority figures, given a good reason, to share with students the encouragement that they need to pursue a stronger mathematical background prior to college. For many programs, the lowest math course recognized by computer science and engineering majors is calculus; students should strengthen their problem solving skills early and save their tuition money for other courses if the necessary math background can be obtained in high school.

Other Forms of Outreach

In addition to trying to recruit high school teachers and counselors to help guide teenagers to a better understanding of what computer science is and what sort of careers are available, we also have been visiting local high schools, either to visit their computer classes or participate in a career day. Presentation materials similar to those used during the workshop, albeit more concise, help the visiting mentor or professor to convey something of the value and possibilities of a computer-related career.

Others have approached students through summer camps or, more directly, by proposing new curricula that incorporates computational thinking or encourages development for human-computer interaction.

Marcu et al. [9] held a four-week summer camp for junior high girls. Students took turns assuming different engineering roles in order to build a product out of items from PicoCricket. The students became more engaged in the work as they were required to present their work in a social setting. Creating an environment where young people can have fun with creative projects can make a significant difference in their attitudes towards technology-related careers.

Yardi et al. [20] proposes the use of HCI to teach computational thinking. Students can exercise their creativity designing a user interface. By having students find out what a fellow student wants in the system, the students can learn that software development is not something that is done alone in isolation. Currently, the idea has been incorporated into a summer program to encourage students to attend college.

FEEDBACK

Attendees of our teacher workshops were asked to evaluate the workshops. They were asked to rate each session on a scale of 1 to 4, where 1 indicates the session was of no benefit and 4 indicates that the session was of great benefit to the attendee. The high school teacher workshop covered the following sessions:

Table 1. Session Evaluations

Session	Min.	Average	Max.
Careers vs. misconceptions	3	3.428571	4
What students need to succeed in college	2	3.333333	4
But I'm interested in X	3	3.8	4
Building your own computer	1	3	4
Building your own network	2	3.333333	4
Using Web 2.0 in the classroom	3	3.571429	4
Using Flash	1	2.6	4

The sessions that addressed misconceptions of computer science careers and discussed recommended courses for students who might one day consider a technologically-related major were almost universally liked. This was especially true of the session on digital forensics, biotechnology, and technology in the arts.

The hands-on sessions were designed to introduce the teachers to things that might be used to build interest in technology. The co-worker organizes efforts to refurbish old computers for schools and charities that need computers. Similar community service activities might also help build students' interest in the inner workings of computers. Other sessions included the use of tools that can be used to encourage algorithmic thinking and teach rudimentary programming while creating entertaining animations. The Flash session, in particular, focused on the ways in which Flash might be used to discuss simple decision-making and the creation of interesting prototypes. At least two of the attendees expected the sessions to include detailed instructions on how to integrate these tools into their classes and were displeased that we simply showed enough of their use to allow the teachers to build curricula around them.

Regardless of whether the attendees appreciated the hands-on sessions, they appeared to all view a greater knowledge of the computer science field to be invaluable for guiding their students.

CONCLUSION

Many people in the local community do not know much about the nature of professors' jobs or the scientific research taking place at the university. When some state legislator starts talking about converting state colleges into extensions of high school, someone invariably comments that professors are not always good at advertising the nature of their discipline and the extent of their accomplishments outside their specific academic circles. The academic computer science community and the industries that depend on technological support and innovation need to likewise make an effort to inform youth and the authority figures in their lives of the nature of computer science, the careers to which it may lead, and the excitement possible in working to solve a problem and finally seeing the pieces come together into a working solution. The public needs to be informed that the job opportunities are growing, even when they temporarily seem to disappear in today's economy, and software engineers and similar careers are considered among the better job opportunities.

ACKNOWLEDGMENTS

I would like to thank the Texas Workforce Commission and its Texas Youth in Technology Grant for funding efforts of colleges to meet the future needs of our economy.

REFERENCES

- [1] Bruckman, A., Biggers, M., Ericson, B., McKlin, T., Dimond, J., DiSalvo, B., Hewner, M., Ni, L., Yardi, S. (2009). "Georgia Computes!" Improving the Computing Education Pipeline. *SIGCSE '09*, 3-7 March 2009, Chattanooga, TN.
- [2] Bureau of Labor Statistics, U.S. Department of Labor (2010). Computer Software Engineers and Computer Programmers. Occupational Outlook Handbook, 2010-11 Edition. Retrieved June 14, 2010, from <http://www.bls.gov/oco/ocos303.htm>.
- [3] Carnegie-Mellon University (2010). Alice.org. Retrieved on 9/10/2010 from <http://www.alice.org>.
- [4] Carruthers, S., Milford, T., Pelton, T., Stege, U. (2010). Moving K-7 education into the information age. *WCCCE '10: Proceedings of the 15th Western Canadian Conference on Computing Education*, Kelowna, BC, Canada.
- [5] Commission on Professionals in Science and Technology. Retrieved 4/30/2009 from <http://www.cpst.org>.
- [6] Fisher, A., Margolis, J. (2002). Unlocking the clubhouse: The Carnegie Mellon experience. *SIGCSE Bulletin* 34 (2).

- [7] Lee, C., Han, H. (2008). Analysis of skills requirement for entry-level programmer/analysts in Fortune 500 corporations. *Journal of Information Systems Education* 19 (1).
- [8] Mallin, E. (2010). On-line Picasso Project. Retrieved on 9/10/2010 from <http://picasso.shsu.edu>.
- [9] Marcu, G., Kaufman, S., Lee, J., Black, R., Dourish, P., Hayes, G., Richardson, D. (2010). Design and evaluation of a computer science and engineering course for middle school girls. *SIGCSE '10*, Milwaukee, WI, USA.
- [10] Margolis, J., Fisher, A. (2002). *Unlocking the Clubhouse: Women in Computing*. MIT Press, Cambridge, MA,
- [11] Massachusetts Institute of Technology (2010). Scratch: Imagine, program, share. Retrieved on 9/10/2010 from <http://scratch.mit.edu/>.
- [12] Pandora Media, Inc. (2010). Pandora Radio - Listen to free Internet radio, find new music. Retrieved on 9/10/2010 from <http://www.pandora.com>.
- [13] Robocode Home (2010). Retrieved on 9/10/2010 from <http://robocode.sourceforge.net>.
- [14] Schulte, C., Knobelsdorf, M. (2007). Attitudes towards computer science: Computing experiences as a starting point and barrier to computer science. *ICER '07: Proceedings of the Third International Workshop on Computing Education Research*, Atlanta, GA, USA.
- [15] Stephenson, P., Peckham, J., Hervé, J., Hutt, R., Encarnaçāo, L. (2006). Increasing student retention in computer science through research programs for undergraduates. *SIGGRAPH '06: ACM SIGGRAPH 2006 Educators Program*. Boston, MA.
- [16] University of Washington Computer Science and Engineering Department. (2007). "Pathways in Computer Science", Why Choose CSE?, retrieved on 1/25/2011 from <http://www.cs.washington.edu/WhyCSE>.
- [17] Wellman, B., Anderson, M., Vrbsky, S. (2009). PREOP as a tool to increase student retention in CS. *Journal of Computing Sciences in Colleges* 27 (2).
- [18] Wilde, D., Harris, E., Rogers, Y., Randell, C. (2003). The Periscope: supporting a computer enhanced field trip for children. *Personal Ubiquitous Computing* 7 (3-4).
- [19] Yardi, S., Bruckman, A. (2007). What is computing?: bridging the gap between teenagers' perceptions and graduate students' experiences. *ICER '07: Proceedings of the Third International Workshop on Computing Education Research*, Atlanta, GA, USA.
- [20] Yardi, S., Krolkowski, P., Marshall, T., Bruckman, A. (2008). An HCI Approach to Computing in the Real World. *J. Educ. Resour. Comput.* 8 (3).

- [21] Zweber, S. (2009). Upward trend in undergraduate CS enrollment; doctoral production continues at peak levels. *Computing Research News*, May 2009. Retrieved June 14, 2010, from <http://archive.cra.org/statistics/survey/0708.pdf>.

BUILDING A THRIVING CS PROGRAM AT A SMALL LIBERAL ARTS COLLEGE*

Timothy Urness and Eric Manley
Department of Mathematics and Computer Science
Drake University
Des Moines, IA 50311
515 271-2118
timothy.urness@drake.edu, eric.manley@drake.edu

ABSTRACT

In this paper we describe several techniques that have helped increase enrollment in the computer science program from 23 computer science majors in 2008 to 42 computer science majors in 2010 - an increase of 82.6%. We discuss issues related to curriculum, programming assignments, and professor-student interactions that have made the discipline more attractive and manageable to a variety of students within the setting of a small liberal arts college.

INTRODUCTION

Liberal arts colleges promote a diverse study of disciplines to emphasize breadth of education, analysis, and integrity in students' intellectual experiences. The goal is to provide a learning environment that prepares students for meaningful personal lives, responsible citizenship, and ultimately, professional accomplishments. At first glance, the liberal arts approach may seem contradictory to an effective method of teaching computer science - a field closely tied to real-world problems and engineering. However, a computer science program can be strongly supported by the liberal arts paradigm [14]. Liberal arts programs in computer science typically emphasize problem-solving, applications of theory, communication, and intellectual skills, not just current technology trends or operational details that may change rapidly. The material content of computer science requires analyzing, synthesizing, and organizing information. These life-long

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

skills are fundamental to a liberal arts education [8]. In this paper, we discuss several of the issues related to curriculum, programming, and professor-student interactions that have made the discipline more attractive and manageable to a variety of students.

CURRICULUM

The curriculum of a liberal arts college emphasizes breadth of study. As a result, students must take a wide variety of courses from outside their chosen area of study. Thus, only approximately 39 hours of classroom time in computer science or mathematics courses can reasonably be required for a major [8]. We acknowledge that each school has a unique environment and several of these suggestions may not be applicable or have the same effect on student enrollments or experiences. However, we have found that these recent changes in our curriculum have encouraged an influx of students to study computer science.

Use Introductory Course to Recruit

Entering students typically have misconceptions about computer science. In a study by Deborah Wiley, a teacher in North Canton Ohio, 66 percent of high school students had no idea of what the field of computer science is about. Only 7 percent believed that computer science involved programming and networking [5]. While strategies to inform and recruit high school students have shown to be effective [9], we have chosen to focus our efforts on students that have already selected and enrolled in the college. We feel that our limited resources can be best utilized by introducing students to computer science, stressing the potential it has to assist other areas of study and contribute to a liberal arts education. This approach has resulted in a number of students adding computer science as a second major or minor.

Using the approach of attracting current students, it is important to view the introductory course (CS1) as a recruitment tool. Ultimately, the introductory course should be an invitation to computing that highlights applications and usefulness of algorithmic thinking and processing. We have found success in not requiring any programming or computing experience for this course - only some basic mathematics (e.g. college algebra). The goal is for students to take the course and have their views of computer science enlightened by introducing algorithms, programming, basic fundamentals, and applications to other disciplines. In the process, we hope to debunk any previous and erroneous attitudes and beliefs the students may have [2, 7].

Simply providing an excellent introductory course will not necessarily entice students to enroll in the course. In our case, increasing the number of majors was likely a result from adding the introductory computer science courses to the list of courses that satisfy a liberal arts general education requirement. Since computer science is a versatile discipline, the case can be made for the introductory course to be placed in several different general education categories (e.g. information technology, critical thinking, quantitative analysis).

Hooking Nonmajors with a Broad CS0

It has been our experience that CS0 can also be effective in introducing and recruiting students to the discipline of computer science. In contrast to programming-centric introductory courses, introductory courses that provide a broad survey of computer science principles have attracted a much higher percentage of female students and had success in student retention [4, 13].

Simplify Curriculum

Once a student's interest is piqued by an introductory course, it is important to demonstrate that the curriculum is simple and manageable. As suggested in [8], courses with prerequisites should be minimal as to not deter a student that takes the introductory course during his or her sophomore year. Demonstrating that the major (or minor) can still be fulfilled in the senior year without having declared it as a freshman will allow for the inclusion of many more students.

Pay Attention to the Student Experience

Computer science can be an intimidating field of study, full of complicated jargon and ever-changing technology. This is particularly daunting for students in disadvantaged populations or students that have not yet formed an academic identity. As suggested in [5], an approach to increasing women in computing is to "pay ferocious attention to the quality of the student experience." We believe this applies to increasing enrollment numbers in general - increasing the *quality* of education experiences for all students.

In an effort to make computer science approachable, we eliminate "weed-out" courses that affirm only the brightest or most-committed students. We feel that the professor should be an advocate for each student, allowing grace for students that have a wide range of computing experiences. As professors, we have no influence on the students' background or prior availability of technology. We do, however, have control over how the student perceives computing in the classroom. By paying attention to the student experience, the unfortunate preconceptions of the discipline can be overcome provided every student is given the attention and opportunity to succeed.

Encourage CS Minors

The number of majors in a program is an often-used and potentially misleading metric for the success of a college program. Students with multiple majors, minors, or concentrations add to the health of a program, but may not be counted as a major. A low number of students enrolled in a course, however, will seemingly always get the attention of an administrator with the power to cancel the course. The far-reaching applications of computer science make it a natural complement to other disciplines as a minor. A program filled with computer science minors can lead to a vibrant, healthy, computer science department. Promotion of the computer science minor program can be done in a variety of ways: announcements in CS0 and CS1, department website, emails, etc.

Incorporate Current Technology Trends

A balance that must be found for each computer science professor is how much technology to teach vs. how much theory and concepts to introduce. Mobile application development can be an attractive method for attracting students' attention while still cultivating algorithmic and critical thinking within the discipline. Our experience is that offering an iPhone application development course in the context of software engineering has created a large amount of excitement and enthusiasm within the department.

PROGRAMMING AND PROGRAMMING ASSIGNMENTS

Students will (likely) spend more time working on assignments than in the classroom. The assignments also constitute a large percentage of the final grade. Thus, from the students' perspectives, the assignments are extremely important to the course and their perception of the discipline. To support student interest, it is critical that the assignments are relevant, manageable, not trivial, and highlight the concepts stressed in the classroom. The following "rules of thumb" can help professors develop meaningful, appropriate assignments that cultivate and support interest in the class and discipline.

Make Assignments Meaningful

Nick Parlante of Stanford University has coordinated the publishing of several "nifty" assignments at the annual SIGCSE conference [11]. These assignments were selected from criteria of being fun (nifty), useful for a broad audience (topical), scalable for easy and open-ended assignments, adoptable, inspirational and thought-provoking. Assignments that focus on highly technical aspects ("geeky") can result in the lasting impression that the real-world problems that computer science can help solve are shallow. Programming assignments that stress concepts and encourage students to utilize logic and problem-solving are likely to attract and retain a diverse collection of students [5]. It has been ours, and others, experiences that the assignments that involve "real world" problems or ask open-ended questions are the most likely to inspire creative, dedicated submissions and can encourage students to pursue the discipline further [6].

Facilitate Student Programming

A controlled lab environment allows a professor to make sure all of the compilers will work the same, the system (probably) won't crash, and everything will work as expected. Requiring students, however, to either purchase specific computer hardware or come into a lab to work on an assignment indicates that the technology they are interacting with is not the same as the ubiquitous computing they interact with on a daily basis. It is highly advantageous that students be able to work on their assignments wherever and whatever their home computer may be. Thus, choosing a development environment and technology that is platform independent, reliable, and flexible will make the discipline more accessible and attractive to students. Furthermore, creating thorough, readable, introductory descriptions can reduce the amount of frustration for a beginner.

Visual Programming Environments

A popular technique for getting students hooked on computer science is to use visual programming environments such as Scratch, Greenfoot, or Alice. These environments make programming accessible and immerse the programmer in a media-rich environment, which is appealing to larger audiences who might otherwise disregard computer science because of preconceived perceptions about programming.

PROFESSOR-STUDENT INTERACTIONS

"... building a relationship of respect between teacher and student for women and minority students is the first order of business - at all levels of school. No tactic of instruction, no matter how ingenious, can succeed without it." [12] As the previous quote illustrates, the professor-student relationship is extremely important. Small class sizes and small professor-to-student ratios are some of the reasons typically stated by students that attend small liberal arts colleges. The professor-student relationship can be foundational for a student deciding to pursue computer science. The following are a few suggestions to help develop and strengthen these important relationships.

Out-Of-Class Activities

We have had great success in opening up the ACM programming competition to any student that would like to participate. Many introductory students have chosen to take part in this all-day computing event which has been an effective recruiting tool. Students end up spending time with majors, making friends, and eventually adding computer science as a major or minor. The incentive for the students is a one-time Friday night practice "party" in which we hold a mock competition, followed by pizza, soda, and a Ms. Pac Man tournament. Since many of the students may be new to computer science, we don't expect them to be competitive. Instead, we encourage their effort and highlight the experience (e.g. new techniques learned, the fun of the Ms. Pac Man contest, the food, etc.). The result has been a high retention rate for the annual programming contest and an increase in majors and minors.

Communication and Approachability

A way to build a relationship of respect between professor and student is to treat every interaction with the utmost importance. Several ways we have found to be effective in our approachability are to hold extended office hours and respond to email or wiki questions as soon as possible. When students get prompt feedback, they are more likely to retain the information and the answer can be most effective. Similarly, when grading assignments or exams, a timely response is as important as a quality response. In order for students to learn from mistakes, it is most beneficial to point out these mistakes as close as possible to when the mistake was made. Delaying the feedback cycle, even if the feedback is extremely thorough, increases the chances that the student not learn from the mistakes made and the student may become frustrated with the process or professor.

WOMEN IN COMPUTING

According to Computing Research Association, women have earned 11.8% of the CS bachelor degrees in recent years [15]. Recruiting women to computer science is a topic of much research and discussion [1, 3, 10]. Our goal is to create an environment within our program that encourages the participation of students, regardless of gender. Thus, we attempt to teach computer science as a discipline that offers a great deal to interdisciplinary collaborations and practical applications, rather than the technological "geeky" stereotype that can sidetrack women from engaging in the discipline.

By focusing on creating a thriving program, our percentage of women in the program is currently 20%. We feel we have been successful in not deterring, and possibly encouraging, women to consider the major through the following actions: We do not require students to enter the program having any programming experience. It has been our experience that students that are "fresh" to computer science and have strong analytic skills have the potential to be excellent computer scientists. Therefore, whenever possible, and especially in the introductory courses, we stress the "experience is not a prerequisite" message that has been successful in recruiting women [5]. To do so, the curriculum must be simple enough to accommodate the novice student. Additionally, as previously mentioned, our beginning CS1 course starts from the ground floor - there are no prerequisites for our introductory courses. Any student, provided they have a strong work ethic, can take the introductory course and earn an A. Using CS0 and CS1 as recruitment tools has allowed for us to break stereotypes incoming freshman have regarding computer science [2]. Similarly, we have found that using a broad CS0 course can lay a solid foundation for studying computer science and attract women, as supported by reference [1]. In fact, over the last three semesters, women have made up just over 50% (46/91) of the students in CS0. Finally, our computer science program is complementary to several other majors (math, physics, and even biology). Having a high number of double majors reduces the "geek" image of CS majors on campus that can be beneficial to increasing the number of women in computer science [5].

CONCLUSION

Liberal arts colleges promote a diverse study of disciplines to emphasize breadth of education, analysis, and integrity in students' intellectual experiences. Computer science programs in small liberal arts colleges must effectively balance the ideals of a liberal arts education and current technology trends. We have highlighted several strategies involving curriculum, programming assignments, and professor-student interactions that, we feel, have made the discipline more attractive and manageable to a variety of students. The ultimate goal of education is to enrich students' lives. We feel that a computer science program has the potential to present a student with a unique perspective of analytical problem solving, application of theory, and analyzing, synthesizing, and organizing information. Under the best circumstances, this kind of program will thrive at a small liberal arts college.

REFERENCES

- [1] Alvarado, C., Dodds, Z., Women in CS: an evaluation of three promising practices, *Proceedings of Symposium on Computer Science Education (SIGCSE '10)*, 57-61, 2010.
- [2] Bennett, C., Urness, T., Using daily student presentations to address attitudes and communication skills in CS1, *SIGCSE Bull.*, 41, (1), 76-80. 2009.
- [3] Cohoon, J. M.. Must there be so few?: including women in CS. *Proceedings of the 25th international Conference on Software Engineering*, 668-674, 2003.
- [4] desJardins, M., Littman, M., Broadening student enthusiasm for computer science with a great insights course, *Proceedings of Symposium on Computer Science Education (SIGCSE '10)*, 157-161, 2010.
- [5] Fisher, A., Margolis, J., *Unlocking the Clubhouse: Women in Computing*, Cambridge, MA: MIT Press, 2002.
- [6] Layman, L., Williams, L., Slaten, K., Note to self: make assignments meaningful, *SIGCSE Bull.*, 39, (1), 459-463, 2007.
- [7] Lewis, C., Attitudes and beliefs about computer science among students and faculty, *SIGCSE Bull.* 39, (2), 37-41, 2007.
- [8] Liberal Arts Computer Science Consortium, A 2007 model curriculum for a liberal arts degree in computer science. *J. Educ. Resour. Comput.* 7, (2), 2007.
- [9] Morreale, P., Kurkovsky, S., Chang, G., Methodology for successful undergraduate recruiting in computer science at comprehensive public universities, *SIGCSE Bull.*, 41, (1), 91-95, 2009.
- [10] Othman, M. and Latih, R., Women in computer science: no shortage here!. *Commun. ACM*, 49, (3), 111-114, 2006.
- [11] Parlante, N., Nifty assignments, 2011, <http://nifty.stanford.edu/> retrieved January 11, 2011.
- [12] Steele, C., Race and the schooling of black Americans, *The Atlantic Monthly*, 4, 68-78, 1992.
- [13] Turner, E. H., Albert, E., Turner, R. M., Latour, L., Retaining majors through the introductory sequence, *SIGCSE Bull.*, 39, (1), 24-28, 2007.
- [14] Walker, H. M., Kelemen, C., Computer science and the liberal arts: a philosophical examination, *Trans. Comput. Educ.*, 10, (1), 1-10, 2010.
- [15] Zweben, S., Upward trend in undergraduate CS enrollment; doctoral production continues at peak levels. *Computing Research News*, 21, (3), 2009.

STUDENT ENGAGEMENT THROUGH APPLIED LEARNING: THE "LIVE" BUSINESS PARTNERSHIP MODEL *

Deborah Becker, Connie Hecker
Department of Computer Science, Mathematics and Physics
Missouri Western State University
Saint Joseph, MO 64507 USA
{dbecker, hecker}@missouriwestern.edu

ABSTRACT

This paper discusses applied learning models and the experiences of selecting a working model used at a regional university Missouri Western State University (MWSU). It will show how a blended Applied/Service Learning model followed by other universities, Iowa State University and Virginia Tech, was used to engage students in 'live' projects that strengthen problem solving and critical thinking skills while they analyze, design, and develop software applications from the clients' prospective. Project sponsors, corporate sponsors, and business officers bring industry's needs, expertise, and peer review to projects that blend legacy with new application development using cutting edge tools. Applied/Service learning (ASL) models are not new; inviting industry leaders into the classroom as an active component of classroom content is gaining momentum as educators seek to find new ways to engage a wide disparate student body spanning four or more generations. ASL models incorporate learning styles into course objectives and outcomes allowing students to bridge the generation gaps and come together as teams to produce prize winning web applications. Students are guided in year-long projects through detailed syllabi, rubrics, goals, and objectives while working through the entire software development life cycle (SDLC).

KEYWORDS: applied learning, community service, student engagement, experiential learning, project management

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1. APPLIED/SERVICE LEARNING MODELS

Building a new model in applied learning is not an easy task. Under a myriad of different concepts and names, ASL models have been the topics of educators seeking demonstrable outcomes to course goals and objectives. Forerunners like John Dewey (1938-1997), *Experience & Education*, Kurt Lewin (1890-1947), *Experiential Learning and Action Research*, and David Kolb (1939), who expanded on experiential learning with the introduction of *Learning Cycles and Learning Styles*, provide a foundation into the way people experience life and learning. Adding incites from Daniel Port and David Klappholz's work using Fred Brooks' "The Mythical Man-Month" (MMM) faculty create an atmosphere of discovery and invention. Today the small state university (college) faces the challenge of teaching student bodies comprised of several generations (Baby Boomers, Generation X, Millennials or Generation Y and the New Silent Generation or Generation Z) and is struggling to find teaching methods that reach and inspire.

In his article, "Experiential Learning Cycles, Overview of 9 Experiential Learning Cycle models", (Nell, 2004) John Dewey brings together models ranging from the one-stage model-experience, probably first used by Confucius (450 BC); "Tell me, and I will forget, show me, and I may remember, involve me, and I will understand", extending through models ranging in complexity from this simple one-stage model to very complicated models with six stages-(experience, induce, generalize, deduce, apply, evaluate). The intellectual philosophies of these models can be placed on a continuum (Nell, 2004):

Figure 1: Education intellectual philosophical continuum

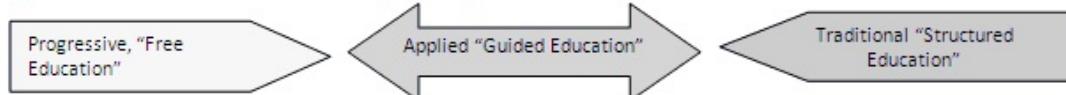


Figure 1-shows the relationship between progressive, applied, and structured learning environments. The progressive educational environment is learner-driven (students are allowed to set their own learning objectives), the applied environment is semi-structured or guided, and the traditional environment is tightly structured. Educational environments find their place along the continuum in modern higher education. In the Computer Science (CS) curriculum students are introduced first to a more structured environment and move towards the applied continuum during sophomore and junior courses and finally experience a more progressive environment during senior capstone or thesis courses.

After review of several ASL models, it was determined that Bert Juch's four-stage model-(doing, observing, reflecting, and planning) (Juch, 1983) aligns with the goals and objectives of MWSU's computer science discipline. Similar to David Kolb's four-stage model-(concrete experience, reflective observation, abstract conceptualization, and active experimentation) (Rimanoczy, 2004), Juch's model expands on active experimentation to interject planning into the model. Software development projects in the mid 1980's through the late 1990's failed at a rate exceeding 70% until educators and industry analysis recognized the importance of a detailed planning phase in the SDLC. Rimanoczy

(2004) states, "The importance of a Plan...connects what we want with what we can do." This is the heart of the task driven planning phase of every software development project.

Port and Klappholz (2006) use a learning module incorporating the reading of Fred Brooks' "The Mythical Man-Month" in a homework assignment and a reflection paper designed around a mathematical equation which purports to predict the cost and time savings of adding personnel to delayed projects. The outcomes of their studies report the illumination experienced by students as they discover "that models are based on assumptions that necessarily simplify the problem at hand" not meant as absolute but suggestive guidelines. During project planning each team build a Work Breakdown Schedule. Brooks' equation is designed to help the student incorporate a more complete overview of time constraints that includes time, effort, training, risk and modularity. They become personally aware that estimating and scheduling are arts that matures with experience and reflection. Project teams study past projects, comparing original estimates with actual projects outcomes striving to achieve better project plans and results.

History is replete with accounts of internships, apprenticeships, guilds and master-student associations under which the next generation learns and expands on the experience of past generations. Bringing 'Live' projects into the classroom approximates an internship experience for students. Consider this question asked by dedicated CS faculty: "How can we expect students to successfully plan, design, and implement a system correctly, when they have never experienced the SDLC before?". It is in the student's best interest to experience a real project with someone who can guide him/her through the process. Instructors become the facilitators as they lead student teams though the intricacies of a live project. This sounds good, but how do these types of projects get started?

2. THE PREPARATION

To increase the number of ASL opportunities for our majors, the CSMP Department created a list of perspective businesses and organizations in the region. Aided by the staff of the Department of Career Services at MWSU, the list of one hundred thirty-seven organizations including area insurance, manufacturing, IT, and other businesses and organizations that employed fifty or more employees was generated. The list also included targeted businesses in Kansas City and Omaha which might provide summer internship opportunities.

An *invitation letter* encouraging business to partner with the CS department in software and network development and an *Information Technology Survey* was generated and sent to all the organizations on the list. After a sixty day waiting period, the responses and surveys were compiled and the results tallied. From the results in Table 1, about 25% of the businesses responded. Of those responding, 61% had local IT departments and 39% were outsourcing IT services. Four (12%) of the businesses reported that they currently had a MWSU student working part-time in their IT department as help desk personnel, night time operators and software engineers. An additional eight (24%) indicated that they were interested in having a student intern and twenty respondents (61% of those responding) indicated they were interested in a site visit and would consider serving on an advisory board for our CS discipline.

Table 1: Computer Science Regional IT Survey Results

Overall: Regional IT Survey Results 137 surveys mailed 33 respondents	Business IT Usage & Needs	Future Plans	New Opts %
Order and Inventory Tracking	15	10	30%
Contact Management	11	18	55%
Accounting	22	0	0%
Point of Sale	7	0	0%
HR/Payroll	21	0	0%
Internet	7	26	79%
Corporate Website	8	18	55%
e-business / marketing	6	12	36%
IT Department Local	20	20	61%
IT Services Outsourced	13	13	39%
Site Visit	20	20	61%
Internship/opportunity	4	8	24%

Through email and site visits the ASL internship opportunities increased from the original four to over thirty in a three year period. Other, unexpected opportunities occurred from the survey; new software development projects and industry needed direction in curriculum updating and alignment. As a result of the survey's positive impact, the CS discipline has included in its goals to repeat it every 4-5 years.

For the past eight years, the CS faculty has been attending conferences on applied learning, service learning, and critical thinking, seeking a shared goal: student engagement and active learning. Using framework from Iowa State and Virginia Tech, as well as concepts from the Diploma program used in the United Kingdom (website Diploma, 2010; LSIS, 2010), CS faculty developed an intricate four-stage model.

3. THE MODEL

The model developed and used by the CS department applies Juch's (1983) four stages (doing, observing, reflecting, and planning) and the following five ASL standards:

- I. Problem Solving-students learn to take a *request for services* and break it down into the activities needed to complete a major project

- II. Communication Tools and Techniques-students keep journals, write documents, conduct meetings, perform interviews
- III. Information Tools and Techniques-Programming languages, modeling software, wiki and blogs are utilized simulating a professional IT environment
- IV. Learning and Self-management Tools and Techniques-students must organize their time and learn to use release time appropriately
- V. Teamwork Tools and Techniques for Working with Others-team creation and role assignments; each student has a leadership role in a major aspect of the project, mentoring, tutoring of other team member and the clients

This model is incorporated into course syllabi and rubrics to set outcome expectation for students in Systems Analysis and Design and Software Implementation courses. The two courses are designed to implement the full SDLS and integrate the ASL standards into a senior year experience for CS degree programs. Participating in 'live' software projects designed for area corporations and non-profit organizations students demonstrate mastery of context areas through applied-service learning. Experience is always the best teacher.

"We would never think of learning a physical skill simply by watching someone else do it; but somehow when it comes to learning information, we seem to think it's sufficient to merely read or hear the information once."

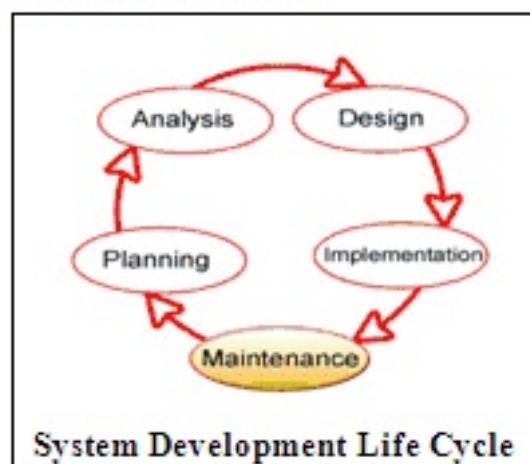
Author Unknown

4. THE PROCESS

Beginning in a junior level course, students in Web Application Development participate in development of a small project sponsored by an area company. All students enrolled in the spring semester class work on their version of the website. They compete in logo creation and web design; the winning entry becomes a live working website for some deserving small business.

The following fall semester several larger projects are started in the Systems Analysis and Design course. Students are introduced to the unified process, which is an object- oriented methodology that takes them through the system development life cycle (Figure 2). As students discover theoretically what needs to be done during the various phases of the SDLC, team assignments apply these theories to the live projects. **For example**, students are introduced to various methods of fact finding to determine the user's system requirements; these methods are then applied to a real project scenario to reinforce the context. Clients are invited into the classroom for interviews and project reviews. Before these interviews, teams create interview agendas focused on project objectives and

Figure 2: SDLC



System Development Life Cycle

requirements. Copies of the agendas are given to the instructor and sent via email to the client. During the interviews, the students lead the discussion; however, the instructor can intervene if specific questions need to be asked to help facilitate a successful project. Part of this learning process involves observation by faculty and peers from other student teams. Afterword, the instructor and peer reviewers debrief the project team members; a detailed interview report is generated verifying the major points of the discussion and emailed to the client, instructor, and project team.

5. THE OPPORTUNITIES

Our generation allows this teaching model to be successful for many reasons. First, service learning has gained immense popularity. According to the Corporation for National and Community Service, service learning is "a teaching and learning strategy that integrates meaningful community service with instruction and reflection to enrich the learning experience, teach civic responsibility, and strengthen communities."

Next, today's traditional student is technologically empowered, good at multitasking easily adapting to change. Here is a response from a typical Z generation student when asked what they want to do with their life, "I would do something where I was constantly growing and learning, and helping people... Primarily, I want to do something significant...and contribute to something greater than myself." Organizations in any community have a need for people with IT skills who want to learn and help others.

The current economic environment has displaced workers returning to colleges and universities to gain training. These adult learners have special needs for hands-on experiences. 'Live' projects allow these students to bring work experience to undergraduate teams which enhance project outcomes. Finally, there is a need for updated technology in almost every business and organization. Classroom projects allow for the integration of new technologies which is training for the next generation of employees. These combinations make this a perfect time for students to become involved in technology based service learning projects. Faculty led student projects for preferably non-profit organizations is a win-win opportunity for all parties involved.

6. REFLECTION AND REVIEW

There are many risks and challenges to having student teams produce viable software and hardware products. Because these projects are a "Team" effort, everyone must complete his/her individual tasks for the project to succeed. Students learn many technological aspects of an information system as they apply the concepts of the SDLC to their projects. However, the importance of the harder to teach soft skills usually become the best lessons learned. Students learn the significance of communication, accountability, responsibility, integrity, time management and organizational skills when involved in these projects. Faculty become masters at communication because expectations must be made very clear to students and to clients. Clients must understand that when working with students, projects usually take longer to complete and the outcomes are not always predictable. Faculty must encourage students to work as a team to organize, communicate and complete each aspect of the project. This scenario

replicates a real business project a student can expect after graduation. As in life, if the project is not managed well, excellent communication is not achieved, or team members do not get along well with one another, the project is in jeopardy.

Faculty also face many challenges when trying to incorporate "live" projects into the classroom. They must keep current with new and existing technologies. They must make sure students have the tools and knowledge needed to complete the requested project. Curriculum alignment is a constant challenge and communication between faculty, students and business clients is essential.

7. SUMMARY

Even with the risks and the challenges, the benefits are worth it. Students are able to site working experience on their resume's and speak to potential employers of the impact they had on their teams project. Faculty are given the opportunity to lead students to learn by guiding them in gaining valuable experience with life lessons. When faculty and students work with non-profit organizations on viable "Live" IT projects, it becomes a community service. Everyone is involved in something that can improve the community in which they live.

REFERENCES

- [1] Breese, J., Richmond, D. (2004). Applied Sociology and Service Learning: The Marriage of Two Movements. *Sociological Practice: A Journal of Clinical and Applied Sociology* (Springer Netherlands) Vol 4, Number 1 / March, 2002.
- [2] Diploma (2010) <http://www.qcda.gov.uk/47.aspx>, retrieved Sept 1, 2010.
- [3] Healey, M., Jenkins, A. (2000). Learning cycles and learning styles: Kolb's experiential learning theory and its application in geography in higher education. *Journal of Geography*, p. 99.
- [4] Juch, A. (1983) Personal Development: Theory and Practice in Management Training Shell International, Wiley, <http://reviewing.co.uk/research/learning.cycles.htm>, retrieved Aug 1, 2010.
- [5] Nell, J. (2004). Experiential Learning Cycles, Overview of 9 Experiential Learning Cycle Models. <http://wilderdom.com/experiential/elc/ExperientialLearningCycle.htm>, retrieved Aug 1, 2010 .
- [6] Port, D., Klappholz, D., (2006) "So You Want Brooks in Your Classroom?" *ACM Digital Library*. ACM, May 2006. ISBN:1-59593-375-1.
- [7] Rimanoczy, I. (2004) The learning cycle, Steps in the process of learning and change, Action Learning News, Sept, 2004. <http://www.limglobal.net/readings/articles/Excerpt%20IFAL%20Sept%202004.pdf>.

- [8] Service Learning Models, Iowa State University, (2010).
<http://www.celt.iastate.edu/ServiceLearning/s-lmodels.html>, retrieved Aug 18, 2010.
- [9] SDLC Figure 2, (2010)
http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle, retrieved Aug 18, 2010.

FRANCES-A: A TOOL FOR ARCHITECTURE LEVEL PROGRAM VISUALIZATION*

Tyler Sondag
Iowa State University
sondag@iastate.edu

Kian L. Pokorny
McKendree University
klpokorny@mckendree.edu

Hridesh Rajan
Iowa State University
hridesh@iastate.edu

ABSTRACT

Integral to computer science education are computer organization and architecture courses. We present Frances-A, an engaging investigative tool that provides an environment for studying real assembly languages and architectures. Frances-A includes several features that enhance its usefulness in the classroom such as graphical relationships between high-level code and machine code, illustrated step by step machine state transitions, color coding to make instruction behavior clear, and illustration of pointers. Frances-A uses a simple web interface requiring no setup and is easy to use, making it easy to adopt in a course.

INTRODUCTION

Computer organization and architecture courses are a crucial component in computer science education [1]. The study of a computer architecture and its behavior is a typical component to such a course. Such components often revolve around "toy" architectures. Although there are advantages to learning a "real" architecture [17], it is complex and difficult to compress this into a semester; therefore, it is desirable to use interactive tools which have been shown to be highly effective for education [8, 17].

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Many tools exist for simulating programs on different architectures [2-4, 7, 11, 13], however, they are typically time consuming to learn and difficult to use. As a result, adopting them in a course is challenging. At the same time, other difficult topics like machine language must be learned which may be significantly different from previous languages students have encountered.

To solve these problems, we present Frances-A, a tool for visualizing program execution on a realistic architecture. Frances-A includes features that enhance its usefulness in education. First, it provides a simple, easy to learn and use web interface requiring no setup. Second, it graphically shows how familiar high-level code maps to machine code, thus, not requiring thorough knowledge of a machine language to start using the tool. Third, it shows graphically how each machine instruction impacts the machine state. Fourth, it allows forwards and backwards stepping through the program allowing students to revisit complicated steps. Fifth, it color codes parts of the machine state to make the impact of each instruction clear. Finally, difficult concepts surrounding addresses (e.g. pointers and stack) are illustrated using color coded arrows.

By providing a simple web-based interface three of the four biggest factors hindering adoption of such visualization tools in educational settings¹ are avoided, namely time to learn the new tool, time to develop visualizations, and lack of effective development tools (other problems such as reliability and install issues are also eliminated) [17]. The interface displays the system state in a logical way and illustrates several important concepts. Further, backwards stepping is a rare feature that our tool includes. All of this together makes the Frances-A tool easy to adopt, use, and understand.

RELATEDWORK

Architectural Simulators

A large body of work exists for simulating architectures and teaching computer organization and architecture [2-4, 7, 11, 13]. There are also many simulators targeted toward advanced users that are typically very complex and difficult to learn. We now briefly discuss work in this area most similar to our introductory computer architecture pedagogical tool.

Null and Lobur developed MarieSIM [11] for use in teaching computer architecture and assembly language. MarieSIM uses a simple assembly language and has an accompanying textbook. Further, MarieSIM requires users to program simulations in the machine language whereas Frances-A gives students the option of entering simulations using high-level languages. Thus, the learning curve is low and students can visualize code they typically write. To help ease the adoption of Frances-A into existing courses, course materials have been developed around topics that complement existing courses utilizing a standard textbook.

¹We solve the fourth problem by making interesting examples (as lessons) available on Frances-A's website

Other related projects include Graham's "The Simple Computer" [7] which uses a command line interface and custom ISA, and GSPIM [3], a MIPS simulator which also shows control flow graphs, but does not maintain actual instruction ordering.

Program Visualization

A large body of work exists for software visualization [5,8,10,12,14,16-20]. Price et al. [12] make the distinction between algorithm visualization (abstract code) and program visualization (actual code). We focus on program visualization and consider algorithm visualization to be complementary to Frances-A. A major difference between this work and previous work is that we believe the Frances-A interface is much simpler than related projects. This addresses the primary factor limiting adoption of previous work [17]. This is able to be done because Frances-A's backend consists of several powerful program analysis techniques. Rather than requiring installation, Frances-A is deployed via a web interface, removing hurdles such as software and OS dependencies. Finally, Frances-A is developed using a realistic machine model and instruction set rather than toy models, avoiding a disconnect between the tool and "real" language [17].

Examples of program visualization techniques include debuggers (e.g. gdb [6]) and graphical debuggers (e.g. kdbg). These tools are often difficult to learn due to their power and expressiveness. Furthermore they do not visualize program structure. Also, debugging concepts such as breakpoints, different techniques for stepping through execution, etc. may be confusing at first. Frances-A's interface is only as expressive as necessary for introductory students. Finally, the interface has fixed abstractions that allow us to eliminate issues like breakpoints and different techniques for stepping through execution.

Most similar is HDPV [16], a runtime state visualization tool for C/C++ and Java programs. This work is complementary to our own in two ways. First, HDPV focuses on visualization of data structures, whereas our focus is on control flow, system state, and program behavior. Second, they deal with large programs whereas we focus on introductory courses. HDPV captures low level details like memory layout, however, they do not trace register values. For introductory students this can be quite confusing. For example, loop counters often never go beyond registers. This limitation is addressed in Frances-A by modeling registers, stack, and heap separately. Finally, Frances-A allows the user to step backward in the code. HDPV does not. Also similar is IBM's Jinsight tool [20] which focuses on more advanced topics.

GOALS FOR FRANCES-A

Specific goals motivated the design of Frances-A. Chief among these goals is an easy to use tool for introductory students learning low-level computer concepts. This includes being easy to learn, requiring no setup, and not requiring thorough knowledge of a machine language. Next, the tool needed to be effective.

Several steps were taken to make the tool as easy to use as possible. First, the tool requires no setup, avoiding issues such as software dependencies. This helps avoid adoption hurdles regarding reliability and eases tool investigation by potential educators.

Next, the simple interface design has a small learning curve. This is necessary to ensure that tools are feasible to adopt in a single semester without distracting students. To facilitate this simple interface, graphical features showing complex properties such as pointers, changes in state and accesses to memory locations are utilized. This includes a logical layout of the system state. Another highly important goal is to not require students to have a thorough knowledge of machine language to start using the tool. Since architecture courses are often the first exposure a student has to machine language, the tool needed to ease this process as much as possible.

A main goal of the Frances-A system to set it apart from others is ease of use, however, effectiveness is critical. It has been shown that the way students interact with visualization tools is more important than the visualization itself [8]. Thus, another goal is to have a hands-on tool that improves the learning process with the following properties:

1. *Support visualization on a real architecture and instruction set.* This makes the knowledge gained by using the tool applicable to standard learning materials and in the real world.
2. *Allow students to enter simulation code in a familiar high-level language.* This helps students visualize how the machine will handle familiar source code. This also allows students to more rapidly perform their experimentation instead of coding visualization code in an unfamiliar machine language.
3. *Allow students to step both forward and backward during program visualization.* This is a feature which is rare amongst such visualization tools. This feature is crucial to allow students to revisit complex instructions and sequences of instructions without re-running the entire simulation.
4. *A graphical and logically organized layout.* We desired a graphical layout that was logically organized, color coded to show accesses and modifications to the machine state, and illustrated pointers.

FRANCES-A

Now the major features of Frances-A are described. To start using the tool readers may point their WWW browser to the URL <http://cs.iastate.edu/~sapha/frances>.

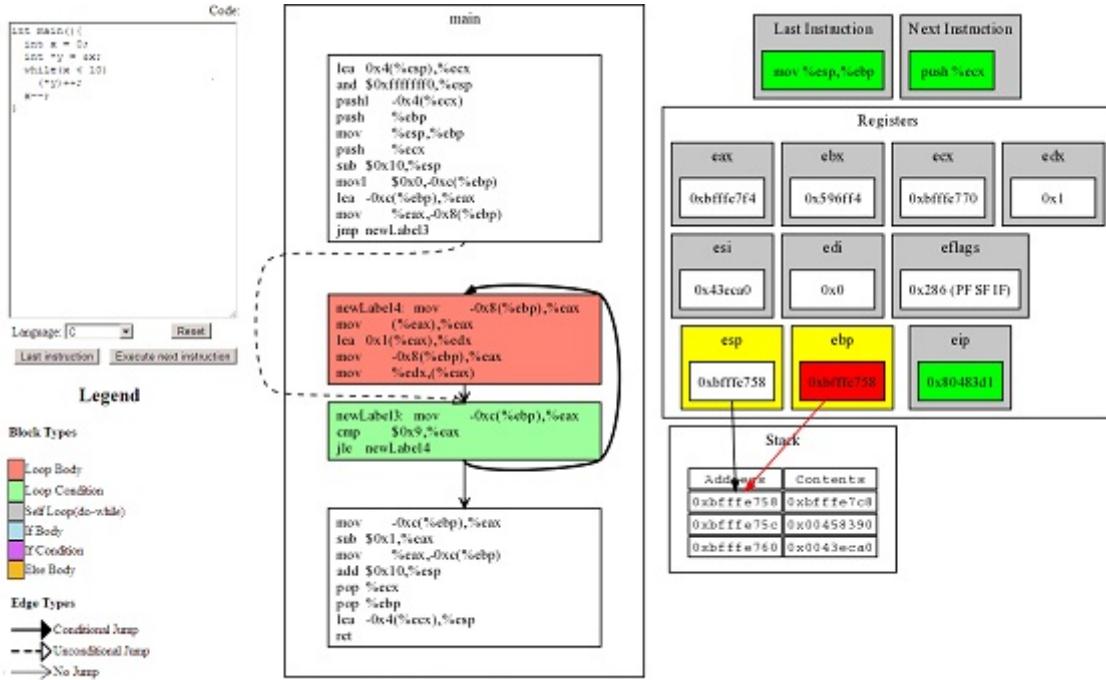


Figure 1: Simple while loop running through Frances-A.

Interface

Key to Frances-A is a simple easy to use web-interface that requires no setup. An example of this interface is shown in Figure 1. For now, let us ignore the detailed aspects of the figure and focus on its major components. There are three main components.

High-level code Entry First, on the far left, is a high-level code input box. Currently, code may be written in C, C++, and FORTRAN. Support may easily be added for any language that can compile down to native machine code. Initially, this box is editable so that users may enter their simulation code. After editing code, the user clicks the "Compile" button. At this point, the code entry box becomes read-only and the "Compile" button is replaced with buttons for stepping backwards and forwards in the assembly code. For example, in Figure 1 the "Compile" button has been pressed and several steps have been executed. At this point the simple while loop code is no longer modifiable unless the "Reset" button is pressed.

Low-level code In the middle is the machine code. This representation comes from previous work on the Frances tool [15]. This component is discussed in more detail in the next section.

Machine state Finally, the far right is the machine state. Currently, this portion of the tool supports the x86 architecture. This portion of the interface is described in detail later.

High-level to machine language relation

The middle section of the interface shows a graphical representation of the machine code (from previous work on the Frances tool targeting teaching code generation [15]). This representation allows users to easily see how high-level code maps to machine code using graphical features such as color coding and different edge drawing techniques. In Figure 1 the middle portion shows the simple while loop. The legend in the bottom left describes edge types and color codes, allowing students to easily identify the code constructs.

The purpose of this portion of the interface is to ease the burden when learning machine language. Students enter code in familiar high-level language, then see how their code is represented in machine code and how that code modifies the system state. Thus students write in familiar high level language and Frances-A provides the connection between the high level, assembly and machine states. Finally, it also helps the user visualize how different program structures behave at the machine level. Currently this portion of the tool supports AT&T and Intel x86 syntax as well as MIPS. Interested readers should refer to the original paper on the tool for a thorough discussion [15].

Graphical layout of machine state

The major extension to the previous work [15] that sets Frances-A apart is the graphical representation of the machine state. In this section each aspect of the graphical representation of the machine state is discussed. The first part consists of the blocks marked "Last Instruction" and "Next Instruction". As the labels suggest, these denote the previously executed instruction that gave the current state and the next instruction to be executed. For example, in Figure 1, a *mov* instruction was just executed and *push %ecx* is next. This allows the user to find the current location in program execution. Then, they can consider the changes caused by the last instruction. For example, the user can see that the last instruction placed the value in *%esp* into *%ebp*. By inspecting the current state, the user can see that these two registers contain the same value. Next, they can try to determine the effects of the next instruction before it executes.

Note that the "Next Instruction" box contains the actual next instruction, not just the next sequential instruction. That is, if the "Last Instruction" was a conditional jump and the branch is set to be taken, the "Next Instruction" is the target of the branch not the fall-through case.

Next, the system's registers are separated from the rest of the state. Within this group there are logical separations. In Figure 2, notice that the first row of registers are the general purpose registers *%eax*-*%edx*. In the next row, the two registers *%esi* and *%edi* are placed together since these are typically used for storing addresses for memory reads and writes. Also in this row is the *eflags* register which contains the results of compare instructions as well as other secondary results of operations. In this figure, the *PF*, *SF*, and *IF* flags are set (all others are unset). The user is also shown the actual hexadecimal value of this register. In the final row, there are three pointer registers. The first two are stack pointers, *%esp* and *%ebp*. Looking at the values contained in the figure, one can determine that these addresses are located on the stack. The final register in this row is *%eip*, the instruction pointer.

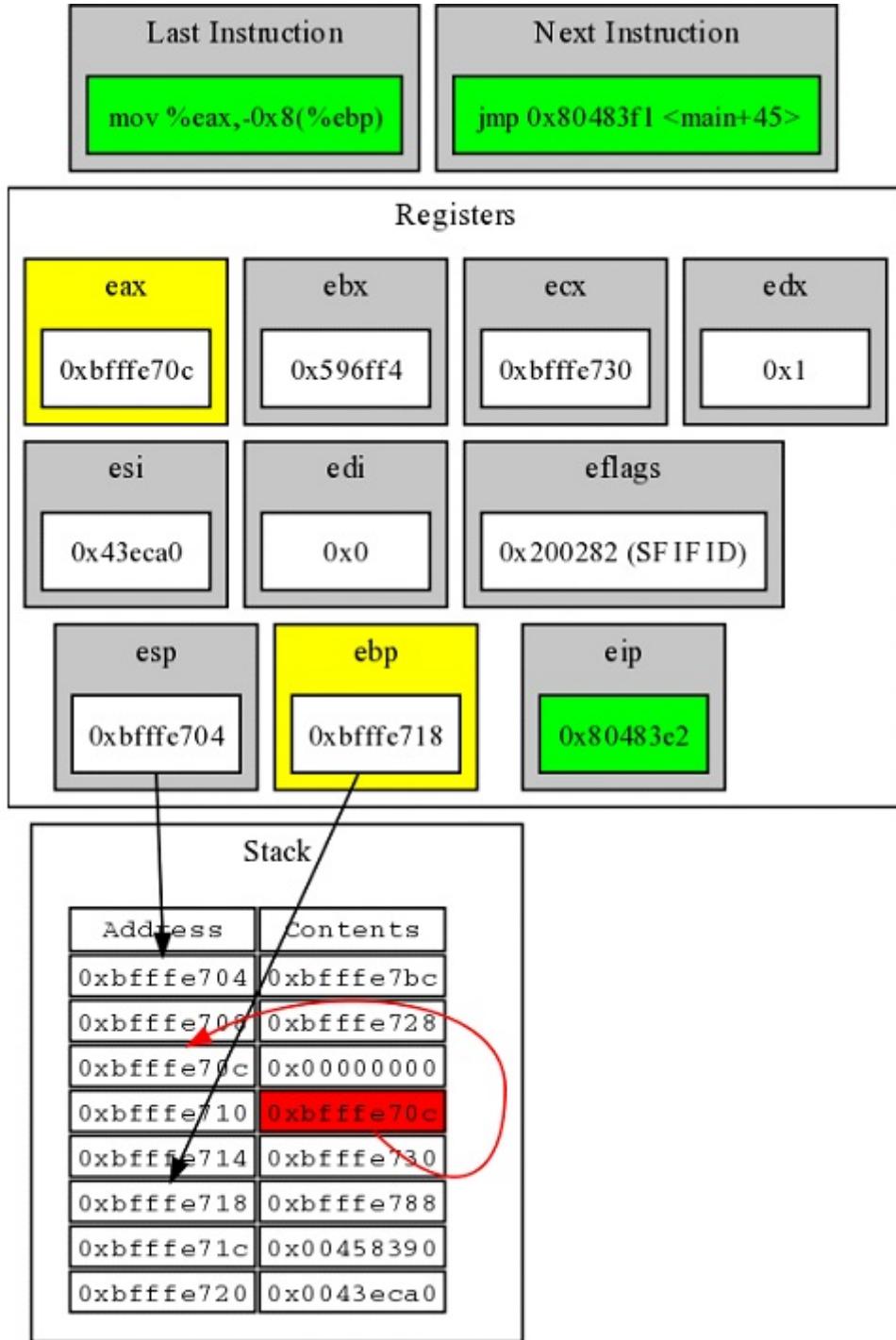
Next, consider the representation of the stack. Figure 2, shows a stack of 8 elements. Each element has its own row with columns specifying the address and contents of that location. The stack is important since it contains most local variables (though some never leave the registers) as well as other temporary values. For example, in Figure 2, the last instruction moved the value in `%eax` to the fourth stack location. This corresponds to the assignment of the address of variable `x` to variable `y` from the code in Figure 1. As a result, there is now an edge from the stack location containing this value to the stack location corresponding to variable `x` (third stack location). This illustrates the behavior of pointers in the machine. The addresses are included in the representation so that users can see how contents of registers correspond to locations on the stack (e.g. stack pointers `%esp` and `%ebp`).

In Figure 2 there are two edges from registers to the stack, for `%ebp` and `%esp`. When stepping through the simulation, it is easy to see that the `%ebp` register points to the location on the stack before the 4 locations are added by the `sub $0x10,%esp` instruction. From the figure it is clear that `%esp` points to the top of the stack. This helps illustrate the purposes of the `%esp` and `%ebp` registers.

Color coding in Frances-A helps illustrate the purposes of the instructions and their impact on the machine state. Green boxes denote portions of the state that always change and are not referenced directly (previous and next instructions as well as instruction pointer). Yellow boxes around the register contents signify that the last instruction accessed these registers. For example, in Figure 2, the `%eax` register has just been read (it was the first operand in the last instruction). Additionally, the `%ebp` register has been accessed when calculating the stack location for the target of the operation. Red highlighting indicates that the contents of the register or stack has been changed by the last instruction. In Figure 1 the `%ebp` register has been modified to contain the value from `%esp` and is thus colored red. In Figure 2, the value in `%eax` was assigned to the stack location corresponding to variable `y`. Thus, the corresponding stack location is red. This avoids the need for the user to try to determine the offset of the stack location manually. Similarly, consider the edge color coding. Notice that the two stack pointers are drawn in black whereas the pointer in the stack corresponding to variable `y` is drawn in red. Like the register and stack coloring, edges in red were changed in the last step.

Reverse stepping

Another important feature of Frances-A which is rare among similar tools, is the ability to step backwards through execution. This is a necessary feature that allows students to revisit complicated steps and groups of steps in the simulation. Note that reversing can also be simulated in a tool by rerunning the simulation; however, students may lose the context of simulation if too many steps are required for revisiting the previous instruction. Often students are surprised by the results of an instruction or group of instructions. The ability to step back in the instruction sequence provides immediate comparative results.

Figure 2: Frances-A after running the tenth instruction (`int *y = &x;`) from Figure 1.

Backend

First, the original Frances tool [15] is used to display the machine code portion of the interface. Next, we use the GDB debugger [6] to perform much of the simulation.

Also several newly developed programs to detect changes in state, detect pointer targets, and arrange the state appropriately are utilized. Finally, GraphViz DOT [9] is used to create the visualization.

CONCLUSION AND FUTUREWORK

Knowledge of computer organization and architecture is a critical component in computer science education. To ease the process of learning real architectures and their behavior we introduce Frances-A. A key benefit of this tool is that it is easy to learn, easy to use, and requires no setup. Further, several steps are taken to enhance its effectiveness. This includes logical separation of components of the machine state (including register types), edge drawing to show pointer targets and stack behavior, color coding to show accesses and writes to illustrate each instruction, and the ability to step both backwards and forwards through execution.

Future work involves support for additional architectures such as MIPS. Due to the existing design of Frances-A, this may quickly be done if users desire. Finally, a thorough evaluation (currently in progress) to ensure the usability and effectiveness of the tool will be completed. Initial results are promising.

Acknowledgments Sondag and Rajan were supported in part by US NSF under grants 06-27354, 07-09217, and 08-46059.

REFERENCES

- [1] Computing curricula 2008: An interim revision of cs 2001.
<http://www.acm.org/education/curricula/ComputerScience2008.pdf>.
- [2] B. Nikolic et al. A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization. *IEEE Transactions on Education*, 52:449 - 458, 2009.
- [3] P. Borunda, C. Brewer, and C. Erten. GSPIM: graphical visualization tool for MIPS assembly programming and simulation. *SIGCSE Bull.*, 38(1):244-248, 2006.
- [4] G. Braught and D. Reed. The knob & switch computer: A computer architecture simulator for introductory computer science. *J. Educ. Resour. Comput.*, 1(4):31-45, 2001.
- [5] M. J. Conway and R. Pausch. Alice: easy to learn interactive 3D graphics. *SIGGRAPH Comput. Graph.*, 31(3):58-59, 1997.
- [6] Free Software Foundation. GDB: The GNU Project Debugger.
<http://www.gnu.org/software/gdb/>
- [7] N. Graham. Introduction to computer science (3rd ed.). West Publishing Co., 1985.

- [8] C. D. Hundhausen. Integrating algorithm visualization technology into an undergraduate algorithms course: ethnographic studies of a social constructivist approach. *Comput. Educ.*, 39(3):237-260, 2002.
- [9] J. Ellson et al. Graphviz - open source graph drawing tools. *Graph Drawing*, 2001.
- [10] M. McNally et al. Supporting the rapid development of pedagogically effective algorithm visualizations. *Journal of Computing Sciences in Colleges*, 23(1):80-90, 10/2007.
- [11] L. Null and J. Lobur. MarieSim: The MARIE computer simulator. *J. Educ. Resour. Comput.*, 3(2):1, 2003.
- [12] B. A. Price, I. S. Small, and R. M. Baecker. A taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4:211-266, 1992.
- [13] P.S. Coe et al. An integrated learning support environment for computer architecture. In *WCAE-3 '97*.
- [14] D. Sanders and B. Dorn. Jeroo: a tool for introducing object-oriented programming. In *SIGCSE*, 2003.
- [15] T. Sondag, K. L. Pokorny, and H. Rajan. Frances: A tool for understanding code generation. In *SIGCSE*, 2010.
- [16] J. Sundararaman and G. Back. HDPV: interactive, faithful, in-vivo runtime state visualization for C/C++ and Java. In *Symposium on Software visualization*, 2008.
- [17] T.L. Naps et al. Exploring the role of visualization and engagement in computer science education. In *ITiCSE-WGR*, pages 131-152, 2002.
- [18] U. Wolz et al. 'scratch' your way to introductory cs. In *SIGCSE*, 2008.
- [19] J. Urquiza-Fuentes and J. Á. Velázquez-Iturbide. A survey of successful evaluations of program visualization and algorithm animation systems. *Trans. Comput. Educ.*, 9(2):1-21, 06/2009.
- [20] W.D. Pauw et al. Visualizing the execution of java programs. In *Revised Lectures on Software Visualization*, International Seminar, pages 151-162, 2002.

TEACHING AGILE METHODOLOGY IN A SOFTWARE ENGINEERING CAPSTONE COURSE*

Baochuan Lu, Tim DeClue

Department of Computer and Information Sciences

Southwest Baptist University

Bolivar, MO

417-328-1676

blu@sbuniv.edu, tdeclue@sbuniv.edu

ABSTRACT

Agile methodology as a relatively new approach to software engineering is becoming more popular in both industry and academia. Learning agile software development methodologies will unquestionably increase the marketability of our students as entry-level software engineers. But how agile methods should be taught at the undergraduate level in addition to traditional approaches is still being debated. The authors taught agile methods in their software engineering/senior project course for the first time in the fall of 2010. Students seemed stimulated by fresh perspectives and the lightweight processes offered by agile, but implementing agile methodology in an academic environment posed unique challenges. In this paper, the authors document their increased understanding of agile methodology through literature reviews, the challenges learned by teaching agile methods, and some potential areas for improvement.

INTRODUCTION

A quick literature survey shows that teaching the agile methodology in the undergraduate software engineering curriculum has attracted attention from many educators and the results of their experiments are mostly positive [1][2][3][4][5][6]. This paper presents a unique and significant perspective due to the course design and the techniques used. The next section outlines the structure of the course, in which agile methods are taught. The uniqueness of agile methodology are then summarized in the

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

methodology section, which is followed by the experience and discussion sections that discuss the challenges encountered and lessons learned through this experience.

The Course

The course under study is a two-semester course sequence that encapsulates a software engineering (SE) course and a capstone project. This course includes two parallel components. The theoretical part consists of weekly lectures/seminars, reading assignments, and quizzes, which cover core SE theories and best practices. The development component involves building a software product for a real customer. Students are required to work at least 10 hours each week on their projects. The goal of this course is to introduce fundamental software engineering concepts and provide an opportunity for students to apply their knowledge in real software development and project management.

The projects are real in that the requirements are from real customers for solving real world problems. Project ideas are first solicited from industry partners, such as software companies, non-profit organizations, schools, and individuals who have sponsored projects before. Proposals are evaluated based on their relevance to the school's mission statement and the project scope. Each selected proposal is converted into a project overview statement to be handed to the students. All projects are team-based (2 to 5 students). Each team is formed based on the members' SE strength and personalities. After a team is assigned a project, they become responsible for making all the decisions in all life-cycle activities, including scheduling meetings with stakeholders. A faculty member is also assigned to each team as a mentor, who meets with the team regularly to keep the students accountable by viewing their time logs, checking project progress, and asking questions.

Students start a project by meeting with their clients to gather requirements and create a project management plan. Students are required to define the project scope, prepare a requirement report, propose solutions, estimate the time and cost involved, outline business values, and identify stakeholders to whom they are responsible. During this process, students have to deal with real customers, changing requirements, and other real world challenges, such as communication breakdowns. Students also choose their software development methodologies, such as the computer language and development environment, unless the customer has specific requirements. Then, students determine the functional scope of the system and document it using either use cases or user stories. Then, a work breakdown structure or product backlog is used to store tasks and prioritize them. The outcome of a project includes project deliverables, such as running software system, source code, test cases, and code level documentation, technical reports, and presentations to the customers and faculty.

Agile Methodology

Software engineering (SE) is defined as a systematic, disciplined, and quantifiable approach to development, operation, and maintenance of software. This discipline encompasses a large body of knowledge [7] concerning software development techniques

and project management practices. Software development is difficult not only because the end product (software) is complex with a huge state space, but also because the process is complex involving many aspects, including human, organizational, and technical [3]. Essential SE principles and practices must be taught to prepare students as future software developers. Classical SE textbooks cover established software processes, which have been developed over time to address software development challenges. Agile methods present an interesting approach to these problems because they explore new ways to achieve the same difficult SE goal, i.e. timely delivery of high quality software within a budget. Whether agile should be taught at the undergraduate level, however, has been debated. A recent paper presented ten convincing reasons why software engineering programs should teach agile software development [8]. The reasons include increasing wide adoption in industry, educating for teamwork, dealing with human aspects, and providing a single teachable framework.

Agile methodology was coined in 2001 to categorize a number of new lightweight software development methodologies developed in the mid 1990's, such as DSDM (Dynamic Systems Development Method), Crystal, XP [9] and Scrum [10]. As opposed to heavyweight plan-driven processes, these new methodologies are lightweight, short cycled, less wasteful, and focus more on the human aspect in software development. Agile practices are guided by the Agile Manifesto [11]: "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, responding to change over following a plan".

One characteristic of traditional approaches, such as the unified process, is plan-driven, which means such processes rely heavily on documented (codified) life cycle processes to ensure software quality and address risks which arise from uncertainties. For example, an up-front software design considers both the current and foreseeable requirements, which makes the process heavyweight. Experience tells us that getting the design right up-front is almost impossible unless the requirements are knowable early in the development and the customer knows exactly what the customer wants. The agile approach, on the other hand, uses evolutionary design to acknowledge the fact that change is often inevitable. Instead of striving for a stable design, the "deliver quickly, change quickly, and change often" agile practices get running code in production as early as possible and use customer feedback to drive further improvements and changes. Such agility is achieved by reducing the cost of change, which depends on enabling practices that increase the reversibility of the software product [12]. The enabling practices include automated testing, continuous integration, and design refactoring. Agile methodology is unique also because it is more people-(team-)oriented. It invests heavily in individual skill-building rather than organizational rule sets [13]. Therefore, cultivating a set of critical skills in developers and maintaining close customer collaboration is essential for implementing agile methodology.

EXPERIENCE

Each software development methodology has its strengths and weaknesses. The relative merits of the agile approach and the traditional approach have been debated for

over a decade [13]. The traditional approach suffers from its inflexibility, which makes it less adaptable to changing requirements. Needless to say, creating a good initial design is very difficult. On the other hand, the traditional approach is more established (easier to teach) and easier to grade due to the extensive documentation required. The success of agile practices has been documented indicating that the agile approach is particularly suitable for projects with a dynamic environment and emerging requirements. Even though the two approaches have seemingly contradictory rules, they are not completely incompatible. To strike a balance between agility and discipline, a combined approach might be more feasible. For example, OpenUP (Open Unified Process) [14] is designed as a lightweight, adaptable unified process implementation that works well in an agile environment to address its lack of planning.

The authors' course takes a similar combined approach. Both the traditional approach and the agile approach are covered. The textbook used covers the unified process at length and has a short section on agile process. The authors supplement the agile coverage using mostly online resources. Because the enabling practices of agile methodology rely heavily on tools, various tools are introduced and demonstrated in class as detailed below.

The authors setup subversion repositories and require students to use them for version control of their source files. One team chose to use Sourceforge, which provides free subversion repositories for opensource projects. Each team is assigned a Windows virtual server accessible both on-campus and off-campus. Students setup their desired development environment on those virtual servers and use the servers also as their test platforms.

Various automated testing tools are introduced in the course. Fitnesse [15] is an acceptance-testing framework that allows non-technical users to specify and run acceptance tests for software systems. The interface consists of wiki pages, through which users can setup and run acceptance tests. Example test cases were studied in class. The xUnit framework is introduced for conducting unit testing for many programming languages, such as JUnit for Java, CppUnit for C++, and NUnit for .Net. An extensive example on how to unit test Java programs was discussed in class. In addition to automated GUI testing using JUnit, another GUI testing tool, Selenium [16], is introduced for automated web app testing across different platforms.

For collaboration, students use CentralDesktop [17], a web-based collaboration tool, for sharing documents, such as a project management plan, meeting minutes, and time sheets, and conducting web-meetings with off-campus stakeholders. Because CentralDesktop offers limited versioning functionality, students are encouraged to convert their design documents into plain text files and check them into their subversion repositories instead. Students' experience with subversion has been positive. Basecamp [18] is also introduced as another alternative web-based collaboration tool. The free plan for Basecamp allows each user to create one project, invite other stakeholders, track todo lists and milestones, and create shared writeboards.

Reading tasks assigned in the course can be categorized into three groups. The first group consists of excerpts from seminal works such as, the Mythical Manmonth, Waltzing with Bears, Peopleware, and Joel on Software. The second group consists of contemporary writings on agile methodology, such as Essential Skills for Agile

Development [19] and Scrum and XP from the Trenches [20]. The third group has one book, the Getting Real book [23], which teaches not only software development methodologies but also philosophies on business, design and marketing. A self-published book by the creator of the open source web application framework Ruby on Rails belongs to its own category. Most ideas presented in the book are unconventional, radical, and most of the time contradictory to the classical SE principles students read in their textbooks. Such reading assignments are intended to stimulate the students' thinking and broaden their views. It also reminds them to stay open-minded, as there is no one-size-fit-all solution; new and fresh ideas are continuously being added to the SE body of knowledge.

DISCUSSION

Software engineering disciplines, including agile practices, are not completely new to students at the authors' school because classical SE processes are taught in our system analysis and design courses and various agile practices, such as pair programming and user stories are taught and practiced in other prerequisite courses as well. More than 50% of the students in this class have had an internship or significant industrial experience. Therefore, the authors did not experience student opposition as documented in other research [21].

The main challenges students face are in communication and planning both stemming from a lack of experience and training. To achieve agility, agile methodology relies on the tacit interpersonal knowledge embedded in the team rather than knowledge written in plans [22]. Therefore, close team collaboration and interaction is essential, though it is well known that effective teamwork is difficult for students. No student works on a single course fulltime. Given their busy schedules, scheduling activities that involve all stakeholders is difficult. To alleviate the problem, students have to rely on asynchronous off-campus means of communications, such as the online collaboration tools, which require different communication skills and result in longer response times.

Students are required to set their own milestones and deadlines. However, self-imposed deadlines do not usually work for students as they will procrastinate as long as possible and scramble to meet the due dates by resorting to their college survival skills. Even though the agile approach is not plan-driven, developers must constantly practice mental planning deciding what should be in the next version. Failure to do so poses a risk to the success of the whole project. As Fowler indicates, the lack of will to design can be a reason why evolutionary design can fail [12]. For students, another reason can be the lack of design experience, which can be addressed by more training. The danger of implementing agile methodology is that when evolutionary design fails, agile methodology degenerates into poor-quality-oriented hacking. Students should document and submit milestones as they go, so that they are forced to think about design and reflect on what they have done well and what they have done poorly.

In agile development, leadership is distributed. All developers are involved in design, coding and testing activities. Shared file ownership encourages students to share their solutions because they are less attached to their code. The side effect of this practice

could make time tracking and assessment more difficult. To assess individual contribution to group projects, the students are required to submit confidential peer evaluations.

Teaching agile methodology also poses challenges to instructors. Agile is not a silver bullet. The quality of code highly depends on the quality of its author. Implementing agile practices means investing in the skills of individual developers, which requires more training/coaching and stepwise introduction. For example, students will not use a tool unless it is required and some training on how to use it is provided. Similar to the "green-field development" proposed in [5], students' engagement with a real project can be postponed. Some teaching-projects can be studied first. Those example software projects can be used to illustrate abstract concepts, such as layered design, and demonstrate practices, such as unit testing. Because the students do not write the code from the teaching projects, they should feel less attached to the code when they are asked to refactor the design, complete test cases, or add extra functionality. The goal is to bridge the knowledge gap and help students embrace agile practices.

SUMMARY

Teaching agile methodology is challenging, but it is worthwhile as it enriches students' software development experience. With better preparation and training, the authors hope students will appreciate and embrace agile practices, such as test-driven-development, simple design, refactoring, and time-boxing.

REFERENCES

- [1] Reichlmayr, T., "An Agile Approach to an Undergraduate Software Engineering Course Project", 33rd ASEE/IEEE Frontiers in Education Conference, 2003.
- [2] Cleland, S., "Agility in the Classroom: Using Agile Development Methods to Foster Team Work and Adaptability Amongst Undergraduate Programmers", 16th Annual NACCQ, 2003.
- [3] Hazzan, O., Dubinsky, Y., "A HOT --- Human, Organizational and Technological --- Framework for a Software Engineering Course", ICSE, 2010.
- [4] Rico, D., Sayani, H., "Use of Agile Methods in Software Engineering Education", Agile Conference, 2009.
- [5] Melnik, G., Maurer, F., "Introducing Agile Methods in Learning Environments: Lessons Learnt", XP/Agile Universe, 2003.
- [6] Alfonso, M., Botia, Antonio. "An Iterative and Agile Process Model for Teaching Software Engineering", 18th CSEET, 2005.
- [7] Guide to the Software Engineering Body of Knowledge (SWEBOk), <http://www.computer.org/portal/web/swebok/htmlformat>, retrieved December 1, 2010.
- [8] Hazzan, O., Dubinsky, Y., "Why Software Engineering Programs Should Teach Agile Software Development", SIGSOFT Software Engineering Notes, 2007.

- [9] Extreme Programming: <http://www.xprogramming.com/xpmag/whatisxp.htm>, retrieved December 1, 2010.
- [10] What is Scrum: <http://www.mountaingoatsoftware.com/scrum>, retrieved December 1, 2010.
- [11] Agile Manifesto: <http://agilemanifesto.org/>, retrieved December 1, 2010.
- [12] Fowler, M., "Is Design Dead?",
<http://www.martinfowler.com/articles/designDead.html>, retrieved December 1, 2010.
- [13] DeMarco, T., Boehm, B., "The Agile Methods Fray", Computer, 2002.
- [14] OpenUP, <http://www.eclipse.org/epf/>, retrieved December 1, 2010.
- [15] Fitnesse, <http://fitnesse.org/>, retrieved December 1, 2010.
- [16] Selenium, <http://seleniumhq.org/>, retrieved December 1, 2010.
- [17] CentralDesktop, <http://www.centraldesktop.com/>, retrieved December 1, 2010.
- [18] Basecamp, <http://basecamphq.com/>, retrieved December 1, 2010.
- [19] Back, E., Essential Skills for Agile Development,
<http://elliottback.com/wp/wp-content/essential-skills-for-agile-development.pdf>, retrieved December 1, 2010.
- [20] Kniberg, H., Scrum and XP from the Trenches,
<http://www.infoq.com/resource/minibooks/scrum-xp-from-the-trenches/en/pdf/ScrumAndXpFromTheTrenchesonline07-31.pdf>, retrieved December 1, 2010.
- [21] Sanders, D., "Student Perceptions of the Suitability of Extreme and Pair Programming", Extreme Programming Perspectives, Addison Wesley, 2002.
- [22] Boehm, B., "Get ready for agile methods, with care", Computer, 2002.
- [23] 37signals, Getting Real, <http://gettingreal.37signals.com/>, retrieved December 1, 2010.

INDEX OF AUTHORS

- Anisetty, P. 45
Arnold, T. 122
Atla, A. 96
Baber, S. 113, 137
Bagsby, P. 225
Becker, D. 166, 275
Brandon, D. 61
Brown, M. 44
Buerck, J. 225
Burch, C. 38, 150
Cain, J. 233, 252
Chu, V. 34
Cigas, J. 196, 233
Clark, J. 235
Cordova, J. 16
Cunningham, B. 113
Daly, T. 23
DeClue, T. 252, 293
Defoe, D. 141
Donaldson, S. 72
Eaton, V. 16
Ferrer, G. 139
Foust, G. 144
Goadrich, M. 36, 146, 148
Graetz, E. 141
Grant, M. 62
Hare, B. 233
Hartness, K. 259
Headington, M. 181
Hecker, C. 166, 275
Heeler, L. 244
Heeler, P. 244
Henehan, B. 137
Hoelzeman, D. 113
Kimball, J. 252
Kurkovsky, S. 141
Lindoo, E. 218
Liu, Y. 116
Lu, B. 252, 293
Manley, E. 268
Massengale, R. 113
McCormick, J. 163, 249
McCount, F. 60
Mei, C. 181
Meinke, J. x
Mertz, A. 215
Middleton, D. 31, 152
Mirielli, E. 213
Mooshegian, S. 225
Morell, L. 2, 130
Naugler, D. 5
Pearce, J. 205
Periyasamy, K. 181
Phelps, G. 116
Pokorny, K. 283
Qualls, J. 62
Rajan, H. 283
Reed, D. 197, 216
Renwick, J. 40, 137
Riggs, K. 53
Rogers, M. 165, 174, 235
Schmidt, C. 189
Sheng, V. 96
Sherrell, L. 62
Siever, W. 244
Singireddy, N. 96
Sondag, T. 283
Spradling, C. 235
Tada, R. 96
Talbert, J. 88
Tarnoff, D. 80
Taylor, K. 16
Thigpen, B. 137
Thomas, S. 162
Tran, Q. 104
Urness, T. 268
Van Cleave, N. 215
Willoughby, L. 225
Wilson, J. 218
Wright, D. 42
Xing, C. 6, 154
Yan, B. 166
Yang, T. 122
Young, P. 45

- Yousef, M. 158
Zhang, X. 181
Zhao, W. 181
Zhou, Y. 88

JCSC

Volume 26 Number 5

May 2011

Muhlenberg College

2400 Chew Street

Allentown, PA 18104-5586