The Journal of Computing Sciences in Colleges

Papers of the Fifteenth Annual CCSC Northeastern Conference

April 16-17, 2010
University of Hartford
West Hartford, CT

John Meinke, Editor
UMUC — Europe

George Benjamin, Associate Editor
Muhlenberg College

Susan T. Dean, Associate Editor
UMUC — Europe

Michael Gousie, Contributing Editor
Wheaton College

Volume 25, Number 6
June 2010
# TABLE OF CONTENTS

THE CONSORTIUM FOR COMPUTING SCIENCES IN COLLEGES BOARD OF DIRECTORS ................................. x

CCSC NATIONAL PARTNERS ........................................ xi

FOREWORD ........................................................ xi

John Meinke, UMUC Europe

CCSCNE 2010 CHAIRS’ WELCOME ................................. 1

Ingrid Russell, Hisham Alnajjar, University of Hartford

BOARD MEMBERS — CCSC NORTHEASTERN REGION ............... 2

2010 CCSC NORTHEASTERN REGIONAL CONFERENCE COMMITTEE .... 2

REVIEWERS – 2010 CCSC NORTHEASTERN CONFERENCE ............ 3

IS COMPUTING AT A TIPPING POINT? A PERSONAL PERSPECTIVE — FRIDAY PLENARY SESSION ............... 5

Fran Allen, IBM T. J. Watson Research Center

COMPUTING AND SUSTAINABILITY: AN ICT PROJECT IN THE HIGH ANDES — SATURDAY PLENARY SESSION ............... 6

Alison Young, Christchurch Polytechnic Institute of Technology

MOBILE PHONE APPS IN CS0 USING APP INVENTOR FOR ANDROID — PRE-CONFERENCE WORKSHOP ............... 8

Hal Abelson, Massachusetts Institute of Technology, Mark Chang, Franklin W. Olin College of Engineering, Eni Mustafaraj, Franklyn Turbak, Wellesley College

VIDEO SCENARIOS: LISTENING, DISCUSSING, EXPLORING, SOLVING – A PARTICIPATORY APPROACH FOR THE COMPUTING CLASSROOM AND BEYOND — PRE-CONFERENCE WORKSHOP ............... 11

Madalene Spezialetti, Trinity College
DEVELOPING COMPETITIVE PROJECT PROPOSALS FOR NSF — PRE-
CONFERENCE WORKSHOP ........................................... 14
Scott Grissom, National Science Foundation

HOW RUBRICS THAT MEASURE OUTCOMES CAN COMPLETE THE
ASSESSMENT LOOP ..................................................... 15
Frances Bailie, Iona College, Bill Marion, Valparaiso University, Deborah
Whitfield, Slippery Rock University

ASSESSING COLLABORATIVE AND EXPERIENTIAL LEARNING .......... 26
Mihaela Sabin, University of New Hampshire

LESSONS AND TOOLS FROM TEACHING A BLIND STUDENT .......... 34
Richard Connelly, Providence College

DESIGNING VALUE SENSITIVE SOCIAL NETWORKS FOR THE FUTURE . 40
Jami Cotler, Janelle Rizzo, Siena College

EXPANDING CS1: APPLICATIONS ACROSS THE LIBERAL ARTS .......... 47
Bridget Baird, Christine Chung, Connecticut College

INTEGRATING RESEARCH PROJECTS IN CS1 ............................... 55
Deborah Sturm, Sarah Zelikovitz, College of Staten Island (CUNY)

AN INTRODUCTORY COMPUTATIONAL COURSE FOR SCIENCE STUDENTS .............................................................. 60
Sohie Lee and Ellen Hildreth, Wellesley College

TEACHING COMPUTER SCIENTISTS TO PLAY WELL WITH OTHERS —
PANEL DISCUSSION ...................................................... 66
Charles Welty, University of Southern Maine, Jesse M. Heines, University of
Massachusetts Lowell, Margaret Menzin, Simmons College

TEST-FIRST DESIGN PEDAGOGY AND SUPPORT FOR INTRODUCTORY OO
PROGRAMMING — TUTORIAL PRESENTATION ............................. 70
Viera K. Proulx, Northeastern University

FLIPPING COINS: AN ACTIVITY FOR INTRODUCING COMPUTATIONAL
THINKING TO NON-COMPUTER SCIENCE MAJORS —
DEMONSTRATION ....................................................... 73
David Toth, Merrimack College

ENSEMBLE - THE ONLINE COMMUNITY CENTER FOR COMPUTING
EDUCATORS — DEMONSTRATION ........................................ 74
Gregory W. Hislop, Drexel University, Lillian N. Cassel, Villanova University,
Lois M. L. Delcambre, Portland State University, Edward A. Fox, Virginia
Tech, Richard Furuta, Texas A&M University, Peter Brusilovsky,
University of Pittsburgh, Daniel D. Garcia, University of California, Berkley

PSUzzle: WEB-BASED INTRODUCTION TO PROGRAMMING PRACTICE — DEMONSTRATION ................................................................. 76
Delbert Hart, SUNY Plattsburgh

ROBOTRAN: EVOLUTION OF AN UNDERGRADUATE SOFTWARE PROJECT .............................................................. 78
Mark Meyer, Kevin Mastropaolo, Canisius College

COMPUTER SCIENCE MEETS INDUSTRIAL ROBOTICS: A VISUAL SERVOING PROJECT FOR A COMPUTER VISION COURSE .......... 85
Sebastian van Delden, University of South Carolina Upstate

ARCHITECTURE OF A JAVA FRAMEWORK FOR DEVELOPING GENETIC ALGORITHMS IN AI CLASS ................................. 93
T M Rao, Sandeep Mitra, The College at Brockport, State University of New York

EXPANDING STUDENT ENTHUSIASM FOR, AND UNDERSTANDING OF, INTRODUCTORY CS — PANEL DISCUSSION .......................... 100
Bridget Baird, Connecticut College, Martin Allen, CTW Consortium, Christine Chung, Connecticut College, Norman Danner, Wesleyan University, Madalene Spezialetti, Trinity College

UNDERSTANDING NSF FUNDING OPPORTUNITIES — TUTORIAL PRESENTATION ................................................................. 102
Scott Grissom, National Science Foundation

EXPERIENCES BUILDING A COLLEGE VIDEO GAME DESIGN COURSE ................................................................. 104
Paul E. Dickson, Hampshire College

MOVING CS50 INTO THE CLOUD ................................................................. 111
David J. Malan, Harvard University

LEARNING COMPUTER SCIENCE CONCEPTS USING IPHONE APPLICATIONS ................................................................. 121
Judith O’Rourke, Ian MacDonald, David Goldschmidt, The College of Saint Rose

EVOLUTION OF CAPSTONE-COURSES IN SOFTWARE ENGINEERING A FINISHING SCHOOL ................................................................. 129
Timoth C. Lederman, Siena College
SUPPORTING AGILE PROCESSES IN SOFTWARE ENGINEERING COURSES ............................................................. 136
Delbert Hart, SUNY Plattsburgh

TEACHING AN ITERATIVE APPROACH WITH ROTATING GROUPS IN AN UNDERGRADUATE SOFTWARE ENGINEERING COURSE ........ 144
Karen Anewalt, Jennifer A. Polack-Wahl, University of Mary Washington

INSTRUCTIONAL ASPECTS OF STUDENT PARTICIPATION IN HUMANITARIAN FREE AND OPEN SOURCE SOFTWARE — PANEL DISCUSSION ............................................................. 152
Heidi J. C. Ellis, Western New England College, Ralph Morelli, Trinity College, Gregory W. Hislop, Drexel University, Norman Danner, Wesleyan University

THREADS EARLY — TUTORIAL PRESENTATION ................. 155
John Beidler, Yaodong Bi, University of Scranton

BANTAM JAVA COMPILER PROJECT: EXPERIENCES AND EXTENSIONS ............................................................. 159
Marc L. Corliss, Joshua Davis, Hobart and William Smith Colleges, David Furcy, University of Wisconsin Oshkosh, Lori Pietraszek, Lockheed Martin

SEMANTIC HOOKS USING A MINI-LANGUAGE FOR A COMPILER PROJECT ............................................................. 167
John S. Mallozzi, Iona College

LEVERAGING HARDWARE DESCRIPTION LANGUAGES AND SPIRAL LEARNING IN AN INTRODUCTORY COMPUTER ARCHITECTURE COURSE ............................................................. 174
John Robinson, Ganesh R. Baliga, Rowan University

WIDENING THE K-12 PIPELINE AT A CRITICAL JUNCTURE WITH FLASH™ ............................................................. 181
Michael Sands, James Evans, Glenn David Blank, Lehigh University

CONNECTING UNDERGRADUATE PROGRAMS TO HIGH SCHOOL STUDENTS: TEACHER WORKSHOPS ON COMPUTATIONAL THINKING AND COMPUTER SCIENCE ............................................................. 191
Patricia Morrealle, David Joiner, George Chang, Kean University

RECRUITMENT OF CS MAJORS THROUGH A NON-PROGRAMMER’S PROGRAMMING CONTEST ............................................................. 198
Mary Anne L. Egan, Siena College
WRITING INTENSIVE AND WRITING EXTENSIVE: A CONTINUUM FOR ADVANCING WRITING IN COMPUTER SCIENCE EDUCATION — PANEL DISCUSSION .......................................................... 205
Yana Kortsarts (Moderator), Widener University, Timothy Dansdill, Quinnipiac University, Adam Fischbach, Widener University, Mark E. Hoffman, Quinnipiac University, Janine Utell, Widener University,

TEACHING VIRTUAL REALITY TO UNDERGRADUATES — TUTORIAL PRESENTATION .................................................... 210
Daniel Cliburn, The University of the Pacific

ACCESSIBLE ROBOTICS AND INTELLIGENT SYSTEMS FOR SOCIAL SCIENCE UNDERGRADUATES — POSTER SESSION .............. 211
Tom Armstrong, Wheaton College

ROLE OF THE PROGRAMMING ASSIGNMENT SUMMARY AS A QUALITATIVE ASSESSMENT OF INDIVIDUAL STUDENT PERFORMANCE — POSTER SESSION ........................................... 214
Karina V. Assiter, Wentworth Institute of Technology

TEACHSCHEME, REACHJAVA: INTRODUCING OBJECT-ORIENTED PROGRAMMING WITHOUT DROWNING IN SYNTAX — POSTER SESSION ................................................................. 218
Stephen Bloch, Adelphi University

THE COMPUTING ONTOLOGY AND ITS IMPLICATIONS IN COMPUTING EDUCATION — POSTER SESSION ................................. 221
Lillian N Cassel, Siva Kumar Inguva, Villanova University

ENSEMBLE: ENRICHING COMMUNITIES AND COLLECTIONS TO SUPPORT EDUCATION IN COMPUTING — POSTER SESSION .......................... 224
Lillian (Boots) Cassel, Villanova University, Ed Fox, Virginia Tech, Weiguo Fax, Virginia Tech, Richard Furuta, Frank Shipman, TAMU, Dan Garcia, Univ of Cal, Berkley, Lois Delcambre, Portland State Univ, Peter Brusilovsky, University of Pittsburg, Greg Hislop, Drexel University, Sridhara Potluri, Villanova University

CSTA SOURCE: A WEB REPOSITORY OF K-12 COMPUTER SCIENCE TEACHING AND LEARNING MATERIALS — POSTER SESSION .... 227
Lillian (Boots) Cassel, Sowmya Moturi, Villanova University

A WEB DESIGN & MULTIMEDIA DEGREE WITH A TWIST — POSTER SESSION ................................................................. 230
Denise S. Leete, Pat Coulter, Pennsylvania College of Technology
COMPUTER GRAPHICS TAUGHT BY BUILDING A RENDERING ENGINE — POSTER SESSION ............................................. 233
Paul E. Dickson, Hampshire College

THE BENEFITS OF USING SCRATCH TO INTRODUCE BASIC PROGRAMMING CONCEPTS IN THE ELEMENTARY CLASSROOM — POSTER SESSION ............................................. 235
Patrice Gans, The Fraser-Woods School

CREATING THE TECHNOLOGICALLY SAVVY K-12 TEACHER — POSTER SESSION ..................................................... 237
Susan P. Imberman, Roberta Klibaner, College of Staten Island

INTEGRATING MERKLE-HELLMAN KNAPSACK CRYPTOSYSTEM INTO UNDERGRADUATE COMPUTER SCIENCE CURRICULUM — POSTER SESSION ..................................................... 239
Yana Kortsarts, Widener University, Yulia Kempner, Holon Institute of Technology

medPing: USING DATA STRUCTURES WITH EMBEDDED MEDICAL DEVICES — POSTER SESSION ............................................. 241
Mark D. LeBlanc, Wheaton College

LEARNING DATA STRUCTURES THROUGH TEXT ADVENTURE GAMES — POSTER SESSION ............................................. 243
Lisa N. Michaud, Saint Anselm College

RUNNING A COMPUTER SECURITY COURSE: CHALLENGES, TOOLS, AND PROJECTS — POSTER SESSION ............................................. 245
Vladimir V. Riabov, Bryan J. Higgs, Rivier College

USING GROUPS ON FACEBOOK TO IMPROVE COMMUNICATION WITH STUDENTS — POSTER SESSION ............................................. 248
James Ryder, SUNY Oneonta

A GENERAL EDUCATION COURSE - “INTRODUCTION TO ALGORITHMIC THINKING” - USING VISUAL LOGIC© — POSTER SESSION ........ 250
Namita Sarawagi, Rhode Island College

A DIJKSTRA’S ALGORITHM SHORTEST PATH ASSIGNMENT USING THE GOOGLE MAPS API — POSTER SESSION ............................................. 253
James D. Teresco, Mount Holyoke College

OUR EXPERIENCES INCORPORATING ROBOTICS INTO OUR SERVICE COURSE — POSTER SESSION ............................................. 256
David Toth, Merrimack College
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:

**Myles McNally**, President (2010), 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.

**Bob Neufeld**, Vice President (2010), Professor Emeritus of Computer Science, McPherson College, P. O. Box 421, North Newton, KS 67117, neufeld@mcpherson.edu.

**Jim Aman**, Membership Chair (2010), Assoc. Professor, Computer Science, Saint Xavier University, Chicago, IL 60655, (773) 298-3106 (H), (630) 728-2949 (cell), aman@sxu.edu.

**Bill Myers**, Treasurer (2011), Dept. of Computer Studies, Belmont Abbey College, Belmont, NC 28012-1802, (704) 461-6823, (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), 011 - 49-6221- 31 58 71 (fax), meinkej@acm.org.

**Kim P. Kihlstrom**, Southwestern Representative (2011), Associate Professor of Computer Science, Westmont College, 955 La Paz Road, Santa Barbara, CA 93108, kimkihls@westmont.edu.

**Elizabeth S. Adams**, Eastern Representative (2011), James Madison University - Mail Stop 4103, CISAT - Department of Computer Science, Harrisonburg VA 22807, 540-568-2745(fax), e-mail: 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, ssigman@drury.edu.

**Ernest Carey**, Rocky Mountain Representative (2010), Dean, College of Technology and Computing, #249, Utah Valley University, Orem, UT 84058-5999, (801) 863-8237 (O), (801) 318-6439 (cell), ECarey@uvu.edu.

**Lawrence D’Antonio**, Northeastern Representative (2010), Ramapo College of New Jersey, Computer Science Dept., Mahwah, NJ 07430, (201) 684-7714, ldant@ramapo.edu.

**David R. Naugler**, Midsouth Representative (2010), Computer Science, Southeast Missouri State University, One University Plaza, Cape Girardeau, MO 63701, (573) 651-2787, dnaugler@semo.edu.

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

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

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

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

**Susan Dean**, Associate Editor, Collegiate Professor, UMUC – Europe, US Post: CMR 420, Box 3669, APO AE 09063; (H) Werderstr 8, D-68723 Oftersheim, Germany. 011-49-6202-5 77 82 14, 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 Program Coordinator, 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, (office), 318-342-1101 (fax), wiedemeier@ulm.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
Cengage Learning
Shodor
Turings Craft

FOREWORD

This is the final issue in volume 25 of the Journal of Computing Sciences in Colleges. It’s very hard to believe that this concludes a quarter century of this publication, a major portion of which I have been involved with. When I look back at those first issues that were offset printed and stapled on regular 8½” x 11” paper. We’ve come a long way. Inclusion in the ACM Digital Library was a real hallmark.

The format of the Journal evolved to its present form relatively early, and we have retained this form now for over two decades. (The one change over those years was the increase in font size when I realized that my own eyes were getting older and we needed to increase the font size for readability.)

As I look back at the last quarter century, thanks are due to many. First of all thanks are due to the Consortium for their support over that period. Without their support there would not be any conferences and there would not be a Journal. My sincere thanks to the Consortium Board for all they have done to ensure that these regional conferences continue and that we have a quality Journal that records the reviewed papers of the presenters. Yes, all papers are peer reviewed using a double blind reviewing process to insure that quality is maintained, and each issue contains a welcome statement from the individual conference which indicates the acceptance rate.

Thanks are also due to the regional boards and conference committees for the hard work that goes into preparing a regional conference. (I served as conference chair three times, and can vouch for the amount of work that is involved.) All of these people are essential in pulling off a successful conference. Needless to say my contact with the regional conferences at this point is primarily with the contributing editors, and every one of them (Mike Gousie for this conference) has been wonderful to work with.

Then, there’s the final production. Susan Dean and George Benjamin have been wonderful to work with over the decades. Susan is very involved with final editing of manuscript, which sometimes is a real challenge. Her contribution has become very
essential as we went from one conference per year 25 years ago to now having ten regional conferences. George handles the coordination with our printer, Montrose Publishing, and that has been most helpful to me being located in Germany. George does the final checking of the blue line and coordinates getting the final printing to the individual conferences. Without these two people and their contributions it would be impossible to deliver the quality that is in the *Journal*. And then there’s Montrose Publishing which is our printer. They have been wonderful to work with for over two decades, and have assumed more within the last couple years, now that they are taking care of mailing the *Journal* to the membership. My sincere thanks to all the folks there.

As I express thanks, I would be remiss in not thanking our National Partners listed above. They help to supply the income that assists in keeping conference attendance costs down, and the Consortium has always endeavored to make the conferences accessible to those of us with minimal travel budgets. Please express your thanks to them for allowing the continuation of the *Journal* and the conferences at minimal cost.

I would be remiss in not expressing thanks to Upsilon Pi Epsilon, the National Honor Society for Computer Science. They have been contributing for many years allowing the conferences to have activities for students. A primary example is the student programming contest which is a part of almost every conference. We appreciate very much their support of our students.

Of course, we cannot leave out ACM SIGCSE. They provide the “in cooperation” support that is so very helpful. ACM also has been very supportive in posting the *Journal* to the ACM Digital Library.

As we approach the end of another academic year, the end of a quarter century of the *Journal*, and the beginning of the next quarter century, I would encourage you, the reader to, become more involved in the local conferences. Everything is accomplished on a volunteer basis, so your help, no matter how small it may seem to you, is always needed and appreciated. A critical item is reviewing, and you are not confined to only your regional conference. We have a number of reviewers who review for multiple conferences. Become one of them. Contact information for the regional conferences is located on the Consortium web page, [www.cESC.org](http://www.cESC.org).

Attend a regional conference (or two or three) as well as help execute the conference of your choice. Check out the web site, the list of regions and the upcoming conferences, as well as how to contact the region and indicate how you would like to help out. The CCSC Board can also use your help. Feel free to contact any Board member. We can always use the help.

*John Meinke*
UMUC Europe
CCSC Publications Chair

xii
Welcome to West Hartford and the 15th Annual Conference of the Northeast region of the Consortium for Computing Sciences in Colleges at the University of Hartford. The conference is held in cooperation with ACM Special Interest Group on Computer Science Education.

We are pleased to present the proceedings of the conference. It includes papers, panels, demos, tutorials, and posters that reflect the broad array of interests of computer science educators. We accepted 22 out of 52 submitted papers, an acceptance rate of 42%. Each paper received at least three reviews. Non-paper submissions received careful review as well. The faculty poster session, in its second year at the conference, saw a significant increase this year.

The program features two distinguished invited speakers. Dr. Fran Allen, of IBM T.J. Watson Research Center, speaks on "Is Computing at a Tipping Point? A Personal Perspective". Professor Alison Young, of Christchurch Polytechnic Institute of Technology in New Zealand, speaks on "Computing and Sustainability: An ICT Project in the High Andes".

In addition to the technical sessions, the conference features three excellent pre-conference workshops. As you enjoy the presentations and discussions, we also hope that you will take advantage of what the area has to offer. The conference venue is a few minutes from Hartford. As the state capitol, Hartford offers the vitality of a city in a smaller more intimate atmosphere with arts, entertainment, dining, and cultural opportunities.

We are fortunate to have worked with a wonderful group of people. The program and the proceedings would not have been possible without the collaboration and hard work of committed people whose efforts resulted in this excellent program. We are grateful to the commitment and efforts of members of our conference committee: David Hemmendinger, Aaron Cass, Darren Lim, Gary Parker, Susan Imberman, Yana Kortsarts, Mark Hoffman, Adrian Ionescu, Michael Anderson, Sandeep Mitra, John Gray, Ralph Morelli, Jonathan Hill, Lonnie Fairfield, Carolyn Rosiene, Stoney Jackson, Paul Tymann, Del Hart, Marc Corliss, Frank Ford, and Stan Kurkovsky.

We thank all our reviewers whose prompt and thorough reviews of the submissions contributed greatly to the quality of these proceedings. Ultimately, it is the participation of those who attend that make the conference what it is. Many thanks to all of the authors for making the conference what it is today. Finally, we wish to extend our appreciation for the support we received from our home institution.

We hope you enjoy the conference and find these proceedings to be a valuable resource.

Ingrid Russell, Hisham Alnajjar, Conference Co-Chairs
University of Hartford
BOARD MEMBERS — CCSC NORTHEASTERN REGION

Lawrence D'Antonio, Chair, 2007-2010 ....... Ramapo College of New Jersey, NJ
Richard Wyatt, Secretary, 2007-2010 ........... West Chester University, PA
Adrian Ionescu, Treasurer 2008-2011 .............. Wagner College, NY
Michael Gousie, Editor, 2007-2010 ............... Wheaton College, MA
David Hemmendinger, Membership Chair, 2005-2010 ...... Union College, NY
Timothy Fossum, Webmaster, 2008-2011 ........ SUNY Potsdam, NY
Frank Ford, Registrar, 2005-2008 ................. Providence College, RI
Lonnie Fairchild ................................ SUNY Plattsburgh, NY
Amruth Kumar ..................................... Ramapo College of New Jersey, NJ
Scott McElfresh .................................. Carnegie Mellon University, PA
Hemant Pendharkar ................................ Worcester State College, MA
Viera Proulx ..................................... Northeastern University, MA
Ingrid Russell ...................................... University of Hartford, CT
James Teresco .................................... Mount Holyoke College, MA
Karl R. Wurst ...................................... Worcester State College, MA

2010 CCSC NORTHEASTERN REGIONAL CONFERENCE

COMMITTEE

Ingrid Russell, Conference Co-Chair ............... University of Hartford, CT
Hisham Alnajjar, Conference Co-Chair .............. University of Hartford, CT
David Hemmendinger, Papers Co-Chair ............... Union College, NY
Aaron Cass, Papers Co-Chair ........................ Union College, NY
Darren Lim, Papers Co-Chair ........................ Siena College, NY
Gary Parker, Panels Co-Chair ........................ Connecticut College, CT
Susan B. Imberman, Panels Co-Chair ............... College of Staten Island, NY
Yana Kortsarts, Demos Co-Chair ...................... Widener College, PA
Mark Hoffman, Demos Co-Chair ........................ Quinnipiac University, CT
Adrian Ionescu, Speakers Co-Chair .................. Wagner College, NY
Michael Anderson, Speakers Co-Chair ............... University of Hartford, CT
Sandeep Mitra, Posters Co-Chair .............................. SUNY Brockport, NY
John Gray, Posters Co-Chair ................................. University of Hartford, CT
Ralph Morelli, Tutorials and Workshops Co-Chair ...... Trinity College, CT
Jonathan Hill, Tutorials and Workshops Co-Chair ...... University of Hartford, CT
Lonnie Fairchild, Registration Chair ........................ SUNY Plattsburgh, NY

REVIEWERS – 2010 CCSC NORTHEASTERN CONFERENCE

Elizabeth Adams .............................. James Madison University, VA
Sven Anderson .................................. Bard College, NY
Tom Armstrong .................................. Wheaton College, MA
Mark Bailey .................................... Hamilton College, NY
Reneta Barneva ................................ SUNY Fredonia, NY
Valerie Barr ..................................... Union College, NY
Chris Bennett ................................. University of Maine Farmington, ME
Stephen Bloch ................................. Adelphi University, NY
Barbara Bracken ................................. Wilkes University, PA
Warren Carithers ......................... Rochester Institute of Technology, NY
Ping-Tsai Chung .............................. Long Island University, NY
Porter Coggins ................................. University of Great Falls, MT
Marc Corliss ................................ Hobart and William Smith Colleges, NY
Jami Cotler .................................... Siena College, NY
Lawrence D’Antonio ....................... Ramapo College of New Jersey, NJ
Nicholas DeLillo ............................... Manhattan College, NY
Adrienne Decker ................................ University at Buffalo, SUNY, NY
Michael Eckmann ................................. Skidmore College, NY
Emanuel Emanouilidis ........................ Kean University, NJ
Lonnie Fairchild ................................. SUNY Plattsburgh, NY
Timothy Fossum ................................ SUNY Potsdam, NY
Alessio Gaspar ................................. University of South Florida Polytechnic, FL
Ernie Giangrande Jr. ............................... Wingate University, NC
John Giordano ................................ US Army
Delbert Hart ................................... SUNY Plattsburgh, NY
Matthew Hertz ................................ Canisius College, NY
Jonathan Hill .................................... University of Hartford, CT
Mark Hoffman ................................. Quinnipiac University, CT
Donald Hsu ...................................... Dominican College, NY
Susan Imberman ................................. The College of Staten Island, NY
Bo Kim ........................................ Daniel Webster College, MA
Bradley Kjell ................................ Central Connecticut State University, CT
David Klappholz ..................................... Stevens Institute, NJ
Roberta Klibaner ......................................... College of Staten Island, NY
Mark LeBlanc .............................................. Wheaton College, MA
Christopher League ...................................... Long Island University, NY
Joan Lucas .................................................. SUNY Brockport, NY
Bonnie MacKellar ........................................ St John's University, NY
Aparna Mahadev ........................................ Worcester State College, MA
Jeff Martens ............................................. University of Maryland, Baltimore County, MD
Fred Martin .................................................. UMass Lowell, MA
Chris Merlo .................................................. Nassau Community College, NY
Jose Metrolho ............................................ IPCB, Portugal
Mark Meyer .................................................. Canisius College, NY
Sandeep Mitra ............................................ SUNY Brockport, NY
Takunari Miyazaki .......................................... Trinity College, CT
Patricia Morreale ......................................... Kean University, NJ
Ralph Morelli ............................................... Trinity College, CT
Robert Noonan ............................................ College of William and Mary, VA
Greta Pangborn ........................................... Saint Michael's College, VT
David Parker .................................................. Salisbury University, MD
Eileen Peluso .............................................. Lycoming College, PA
Sofya Poger .................................................. Felician College, NJ
Rathika Rajaravivarma .................................... NY City College of Technology, NY
Veeramuthu Rajaravivarma ................................ SUNY, Farmingdale State College, NY
David Rilett .................................................. Wentworth Institute of Technology, MA
Stefan Robila ................................................ Montclair State University, NJ
Ingrid Russell ............................................... University of Hartford, CT
Mihaela Sabin ............................................... University of New Hampshire, NH
Richard Sharp ............................................... St. Lawrence University, NY
Cherie Sherman ............................................... Ramapo College of NJ, NJ
Michael Siff ............................................... Sarah Lawrence College, NY
John Slimick .............................................. University of Pittsburgh at Bradford, PA
Hal Smith .................................................... Penn State New Kensington, PA
Daniel Spiegel ............................................. Kutztown University of PA, PA
Deborah Sturm ............................................... College of Staten Island (CUNY), NY
Joo Tan ....................................................... Kutztown University of Pennsylvania, PA
Virginia Teller ............................................... Hunter College (CUNY), NY
Jim Teresco .................................................. Mount Holyoke College, MA
Rebecca Thomas ............................................ Bard College, NY
John Vaughn .............................................. Hobart and William Smith Colleges, NY
David Voorhees ........................................... Le Moyne College, NY
Linda Wilkens .............................................. Providence College, RI
Robert Yoder ............................................... Siena College, NJ
Peter Yoon ................................................... Trinity College, CT
IS COMPUTING AT A TIPPING POINT?

A PERSONAL PERSPECTIVE

FRIDAY PLENARY SESSION

Fran Allen, IBM Fellow Emerita
IBM T. J. Watson Research Center
Recipient of ACM’s 2006 Turing Award

Over the last 60+ years computing, communications, and information, originally emerging from disparate disciplines, have evolved into an unimagined set of capabilities influencing, directly or indirectly, nearly every person, every institution, and every endeavor in much of the world. These capabilities form a global infrastructure for a world where time and place are often irrelevant, access to information is instantaneous, and knowledge widely shared.

The talk will address the following question: Will the field continue to evolve from where we are or has it reached a tipping point? I will draw on my experience with languages, compilers, and high performance computing to suggest a few answers.

SHORT BIOGRAPHY

Fran Allen’s specialty is compilers and program optimization for high performance computers. Soon after joining IBM Research as a programmer in 1957 with a University of Michigan masters degree in mathematics, Fran found the technical goal that would drive her career: Enable both programmer productivity and program performance in the development of computer applications. One result of this goal was that Fran was the recipient of ACM’s 2006 Turing Award: “For pioneering contributions to the theory and practice of optimizing compiler techniques that laid the foundation for modern optimizing compilers and automatic parallel execution.”

Fran is a member of the American Philosophical Society and the National Academy of Engineers, and a Fellow of the American Academy of Arts and Sciences, ACM, IEEE, and the Computer History Museum. Fran has several honorary doctorate degrees and has served on numerous national technology boards including CISE at the National Science Foundation and CSTB for the National Research Council. Fran is an active mentor, advocate for technical women in computing, an environmentalist and explorer.
This address will describe an ICT research project that is context specific and achieved economic and social turnarounds where other ICT projects have failed. The message for computer science educators and professionals is that desired impact has less to do with science and technology and more to do with understanding context and culture. Evaluating implementation options to advance educational and social needs is applying intelligence to technology. Technology without context is a chasm.

The ancient Incan culture, through the Quechuan people of Antabamba Peru, a remote indigenous society high in the Andean Mountains has over 700 years of proven social, environmental and economically sustainable practice. Until only 10 years ago Antabamba was a time capsule which was isolated from the world by several days walk from the nearest road. When the road was built in 1995 the multinational products, television, marketing and western philosophies of business practice soon followed. Within 10 years the population of Antabamba was worse off than in anytime in the previous 700 years and risked losing what the developed world is in search of: sustainable practice.

Starting in 2003 the Unitec project spent a year learning what had underpinned this ancient culture. Yesterdays wireless technologies, internet, web design, No. 8 wire, aluminum foil satellite dishes and some basic tools were grounded in the traditional Incan methodologies of sharing, learning and understanding. Unparalleled results were achieved. Together with the local communities, the Unitec project developed a methodology called "Community Centric Empowerment" (CCE) which has been attributed by OSIPTEL, the Telecommunications Authority in Peru and the Latin American telecommunication council representative as the deciding factor that has separated this project from other "telecenter" projects in Latin America.

Additional studies focusing on the ability of ICT to reduce poverty and exploitation in third world countries by FITEL, the Rural development wing of OSIPTEL in Peru, support the notion of the importance of how, rather than what, when it comes to ICT use.
for poverty reduction (Bossio 2005) (Newman 2006). These studies showed the usage patterns and impact of the Unitec project to be quite distinctive compared with any other poverty alleviation project using ICT.

In keeping with the phenomenological methodology of the initial study, this address will describe the story of the Peruvian project to demonstrate to CS educators and professionals that how we implement ICT is as important as what we implement, when social and economic sustainability are our objectives. It lays down a challenge to CS educators and professionals to reconsider the priorities in our teachings and philosophies.

SHORT BIOGRAPHY

Alison Young is a Chair of Department of Computing at Christchurch Polytechnic Institute of Technology, Christchurch, New Zealand. She was previously Chair of Department of Computing at Unitec Institute of Technology in Auckland, New Zealand. Alison has an academic and professional career that has involved academic leadership in research, scholarship, teaching and curriculum development, nationally and internationally. She is an invited international keynote speaker, Vice Chair of the ACM Special Interest Group on Computer Science Education (SIGCSE), member of the international ACM Educational Council and a Fellow of the New Zealand Computer Society and National Advisory Committee on Computing Qualifications. Alison has a strong background in teaching, research and curriculum development and implementation of computing and information technology qualifications from certificate to doctorate level. Alison’s research interests include Women in Computing, Computing Education, Oral histories, ICT4D and the development and implementation of e-learning.
ABSTRACT

App Inventor for Android (AIA) is a new visual programming environment developed by Google to enable nonprogrammers to create simple mobile phone applications on Android phones. AIA programs are constructed by dragging and dropping components into an interface designer, and then describing the behavior of the components in a Scratch-like block-based programming environment. Programs are developed on a computer, and then downloaded and tested on a phone. In a Fall 2009 pilot program, AIA was used in introductory courses at eleven colleges and universities and at a high school.

This workshop is targeted at faculty teaching introductory computer science courses in college or high school. We will lead hands-on activities in which participants will create applications and test them on phones that we will provide. We will also describe
our experiences with using it in our courses, demonstrate applications created by students, and discuss relevant teaching techniques and practical issues.

**TENTATIVE WORKSHOP AGENDA (3 HOURS)**

1. Motivation and design philosophy (5 minutes).
2. Hands-on walk-through of two simple applications (30 minutes).
3. Demonstration of student apps from our courses (10 minutes).
4. Brainstorming about applications participants can write during the session (10 minutes).
5. Break (5 minutes).
6. Independent work by participants, with support from workshop leaders (60 minutes).
7. Break (5 minutes).
8. Q&A about AIA system (10 minutes).
9. Presentation of pilot course experiences, including sample assignments and projects, particularly from Wellesley and Olin (25 minutes).
10. Discussion of whether and how to incorporate mobile phone programming and AIA into courses taught by participants (20 minutes).

**WORKSHOP LEADERS**

Hal Abelson is a Professor of Electrical Engineering and Computer Science at MIT. He won the 1995 Taylor L. Booth Education Award given by IEEE Computer Society for “continued contributions to the pedagogy and teaching of introductory computer science, exemplified by the Logo and Scheme languages and their associated methodologies.” While a visiting faculty member at Google, he was one of the founders of AIA.

Mark L. Chang is an Assistant Professor of Electrical and Computer Engineering at the Franklin W. Olin College of Engineering where he led a prototype AIA course in the Fall of 2009. Mark’s current research interests include engineering education, reconfigurable computing, and the intersection of ubiquitous/mobile computing and social media.

Eni Mustafaraj is a Visiting Scientist at the Computer Science Department, Wellesley College. In Fall 2009, she co-taught the experimental course "Technologies for Communication" aimed at non-majors, and was responsible for creating teaching materials for AIA. She collaborated with members of the Google team to create a specific AIA component (Voting) to use in the classroom in a discussion about Internet Voting. In her research, she is interested in the problem of finding qualitative information on the Web and how social media content is incorporated in search results.

Franklyn Turbak is an Associate Professor of Computer Science at Wellesley College. In Fall 2009, he mentored students in the introductory CS course that used AIA. His research focuses on the design, analysis, and implementation of expressive programming languages.
PARTICIPANT REQUIREMENTS

Participants must bring laptops with wireless networking. The laptops can run any operating system, as long as they have a supported browser (currently Firefox 3 or higher) and version 1.6 or higher of the Java Runtime Environment. The AIA environment is accessed via the Web, so no other software need be installed.

Participants also need to have Gmail accounts, since these accounts are necessary to access AIA.

We will supply the Android phones that will be used by the participants during the workshop.

ENROLLMENT

Enrollment should be limited to 30 (space permitting) to ensure that we can offer adequate support to participants.
VIDEO SCENARIOS: LISTENING, DISCUSSING, EXPLORING, SOLVING – A PARTICIPATORY APPROACH FOR THE COMPUTING CLASSROOM AND BEYOND*

PRE-CONFERENCE WORKSHOP

Madalene Spezialetti
Department of Computer Science
Trinity College
300 Summit Street
Hartford, CT 06106
Phone: 860-297-2519 860-836-9828
Fax: 860-297-3531
Madalene.spezialetti@trincoll.edu

ABSTRACT

This highly participatory workshop introduces attendees to the use of video scenarios as a tool for teaching computing in a problem-centric manner that emphasizes active listening, discussion and exploratory problem solving. The video scenarios present problems in the context of life-like settings and emphasize the role of computing as part of a process for solving problems that are diverse and frequently ill-defined. Attendees will participate in a variety of small and large group exercises to experience the many ways video scenarios can be used to energize classroom, laboratory and programming experiences, while developing computational, entrepreneurial and critical thinking skills.

INTENDED AUDIENCE

College and secondary school CS educators whose classes may range from general education through programming and upper-level computing courses.

* Copyright is held by the author/owner.
PRESENTER BIOGRAPHY

Madalene Spezialletti is an Associate Professor of Computer Science at Trinity College. Her research focuses on the development and use of video and animation in Computer Science education. She received a SIGCSE Special Projects Grant to support her work in the production of video scenarios, which are short films designed to develop computational thinking skills by presenting computing-based problems in the context of extemporaneous, real-life settings. She gave a presentation on the results of her funded research at SIGCE 2009 in a session for recipients of SIGCSE Special Projects Grants. She also presented a poster at SIGCSE 2009 on the video scenario approach and two of her works were accepted to the Video Channel at SIGCSE 2009. A poster and two videos have also been accepted for SIGCSE 2010. She has used video scenarios in teaching CS1, CS2 and general education classes.

MATERIALS PROVIDED

Each participant will receive both a paper and an electronic summary of the presentation portions of the workshop and will receive an electronic summary of the discussions in the feedback portions of the workshop. The videos will be available on the web (a selection of videos can currently be viewed at www.Virt-U.org).

AGENDA

Overview of the Paradigm of Video Scenario-based Learning

Since the novelty and many possibilities of using the approach can be best appreciated by participation, the introduction will be brief, so that attendees can quickly become engaged in the experience of using video scenarios. (10 minutes)

Participatory Exercises

Participants engage in a series of exercises allowing them to experience and explore the variety of ways in which video scenarios can be utilized.

- Clients, Consultants, Critical Listening and Computational Thinking. This role-playing exercise utilizes video scenarios in which clients (members of fictitious companies) talk about their enterprises in a conversational manner as if addressing web design consultants (the audience). The clients are not specific as to what they want on their site, rather the “consultants” must listen critically to the conversation, and identify the client expectations and requirements. In addition to identifying an overall concept and feature set for the web site, the consultants must also conceive of ways in which computing technologies can be utilized to enhance the client’s services, sales or web presence. This is an excellent exercise in any computer science class, as it is a very accessible way to introduce students to verbally expressing themselves to team members (as consultants), to presenting ideas to a critical audience (consultants to clients) and to questioning and analyzing the ideas of others (as clients). For this exercise, attendees are divided into small groups. Two video scenarios are used, so that every group has the experience of being both
consultants and clients. After viewing the videos, the consultants derive their “pitches” while clients discuss their expectations (based on the characters in the video). Consultant and client groups are then paired for a “pitch” meeting in which consultants describe their ideas to the clients and clients respond. (30 minutes for exercises involving two videos, followed by 10 minutes of discussion and feedback and hints on other uses for the scenarios viewed).

- **Ethics and Practices: What to Do or How to Do It.** These exercises demonstrate using video scenarios as a catalyst for exploring questions of ethics and approaches to effectively interacting with team members or clients. Attendees explore the use of the videos for class and group discussions as well as writing exercises. (30 minutes, including feedback)

- **Problem Solving Scramble.** Videos scenarios with a more computational or algorithmic flavor will be used for these exercises. In the scramble technique, small groups are given a short period of time to propose a solution to a video-scenario problem. This time-limited technique is an ideal way of incorporating the use of videos into any setting where time is at a premium. (30 minutes, including feedback)

- **All the Way to Implementation.** The video scenarios are by nature ambiguous, designed to invite a variety of interpretations and problem-solving approaches. These exercises allow attendees to explore ways to handle such ambiguity when using the videos as a basis for implementation assignments or labs. (30 minutes, including feedback)

**Wrap-up: Ideas and Tips**

Input will be sought from attendees for alternative uses of video scenarios and for ideas of possible future directions for types or topics of videos. Do-It-Yourself tips for developing scenario-based scripts for discussion or programming project descriptions (which, for those interested in creating their own videos, could also serve as a basis for filmed projects) will be presented. (25 minutes)
DEVELOPING COMPETITIVE PROJECT PROPOSALS FOR

NSF*

PRE-CONFERENCE WORKSHOP

Scott Grissom
Program Director
National Science Foundation
sgrissom@nsf

SUMMARY

This interactive workshop enables participants to work in small teams to explore a systematic process for converting an idea into a competitive proposal. We will identify the most common strengths and weaknesses cited by reviewers of NSF proposals. Led by a Program Director of the NSF Division of Undergraduate Education, the workshop will use a series of interactive exercises in which participants first reflect on their own current understanding of the issue (Think), then share these ideas in small groups (Share) and then again with the entire group (Report), and finally hear an expert's opinion (Learn). Particular attention will be given to the following elements of NSF proposals.

- Goals and Expected Outcomes
- Implementation Strategy
- Evaluation Plan
- Broader Impacts
- Dissemination Strategy

Through this process, participants should develop a better understanding of the attributes of a competitive proposal and the role that reviewers play in the proposal decision process; the factors that reviewers consider in processing proposals; and approaches for anticipating and responding to these factors. This will enable them both to prepare more competitive proposals for NSF's education programs and provide them with a broader perspective on writing successful proposals. Q&A opportunities will be encouraged throughout the workshop.
HOW RUBRICS THAT MEASURE OUTCOMES CAN COMPLETE THE ASSESSMENT LOOP

Frances Bailie  
Department of Computer Science  
Iona College  
New Rochelle, NY  
fbailie@iona.edu

Bill Marion  
Department of Mathematics and Computer Science  
Valparaiso University  
Valparaiso, IN 219-464-5422  
Bill.Marion@valpo.edu

Deborah Whitfield  
Department of Computer Science  
Slippery Rock University  
Slippery Rock, PA  
724-738-2935  
deborah.whitfield@sru.edu

ABSTRACT

Program assessment of student learning includes the following steps: 1) involving all constituents to establish program goals, 2) developing measurable student learning outcomes for each of the goals, 3) developing measurable outcomes for each course that map to the student learning outcomes, 4) determining appropriate assessment methods in the courses, 5) creating assessment instruments (or rubrics) for each of the methods, 6) establishing benchmarks, 7) analyzing the data, and 8) using the results to improve student learning. This paper focuses on the last four steps by beginning with a generalized assessment plan for an undergraduate computer science program. A generalized rubric for computer programs is presented that measures selected student learning outcomes. This paper demonstrates how to apply the generalized rubric to two specific computer programming assignments. Benchmarks associated with the rubrics are suggested. Sample results are analyzed to identify problems and propose solutions—“closing the loop.”

* Copyright © 2010 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.
INTRODUCTION

A major reason for developing an assessment plan is to determine how well a program is doing in meeting the goals faculty set for student success. The driving force in determining program goals should be the competencies the faculty want their students to exhibit and the skills they want their students to possess after completing the program. Since reasons for developing an assessment plan vary, e.g., to satisfy regional accrediting bodies or to obtain discipline-specific certification, it is not possible to adopt a single blueprint that will be appropriate for all. However, we can identify a few basic components of effective assessment plans [3, 5, 8, 15].

ABET has adopted specific definitions for the following terms: Program Educational Objectives and Program Outcomes [6]. Educational objectives are broad statements that “describe the career and professional accomplishments that the program is preparing graduates to achieve” whereas outcomes are measurable statements that “describe what students are expected to know and be able to do by the time of graduation.” Differences between the two were exemplified in 2009 [13]. This paper adopts the model: program goals and student learning outcomes both articulate that which we want our students to achieve by the time they graduate.

Once the goals and student learning outcomes have been articulated, measurable course outcomes must be developed that map to the student learning outcomes to ensure that all courses in the program of study are addressing the overall student learning outcomes. This process also verifies that all student learning outcomes are addressed in at least one course. (In the remainder of this paper, the term outcomes will refer to student learning outcomes.) The next step would be to determine one or more assessment methods to measure the outcomes. A combination of both indirect and direct methods of assessment is desirable. Indirect methods include: student or faculty surveys, reflection papers, focus groups, and exit interviews where students are asked to discuss what they have learned and its impact on their learning. Direct assessment methods include programming assignments, projects, in-class tests, portfolios, oral presentations and the ETS’s Major Field Test [4, 11, 12, 16]. These methods require students to demonstrate what they have learned rather than to discuss what they have learned. In addition, for each of the methods deployed, scoring guides (rubrics) based on a set of performance criteria must be developed to help evaluate student work [9, 10]. At this point, faculty will want to establish thresholds (benchmarks) and develop processes for analyzing and reporting the data to the stakeholders. Data that is collected must be analyzed to identify problems, solutions must be proposed and implemented, and the process must then begin again—“closing the loop.” Problems must be tracked until they are resolved [7].
GENERAL CS GOALS & OUTCOMES

A common starting point for creating goals and outcomes is to draw upon the institution’s mission statement to construct a set of program goals that articulate what the program hopes to accomplish. These should be high level statements that are then broken down into more specific student learning outcomes (SLOs). It is essential that these outcomes be explicit and measurable and that they target a specific skill level, such as those articulated in Bloom’s Taxonomy [14]. Here we present a set of program goals and student learning outcomes developed by the authors and other colleagues by combining and refining the goals and outcomes at their respective institutions.

**Goal I: Critical Thinking and Problem Solving:** Students will develop problem-solving and critical thinking skills and use these skills to solve abstract and complex computing problems.

**Student Learning Outcomes**

Students will:

A. develop abstract models and design a solution to a computing problem  
B. design an algorithmic solution using decomposition and stepwise refinement  
C. develop and design software solutions using different design methodologies, data structures, and programming languages

**Goal II: Theoretical Foundations:** Students will acquire a working knowledge of the theoretical foundations of computer science.

**Student Learning Outcomes**

Students will:

A. use mathematical underpinnings of the discipline of computer science  
B. examine the correctness and efficiency of the design of a software system  
C. analyze the complexity and computability of algorithmic solutions

**Goal III: Ethical Responsibilities:** Students will become informed and educated citizens in terms of their professional responsibility to address the social and ethical implications of the use of technology.

**Student Learning Outcomes**

Students will:

A. recognize the ethical, legal, and social implications of computing  
B. analyze the impact computing has on the global society  
C. ensure the security, privacy, and integrity of data

**Goal IV: Communication and Interpersonal Skills:** Students will acquire communication and interpersonal skills necessary to perform effectively in a technical environment.

**Student Learning Outcomes**

Students will:

A. use oral and written communication skills to convey technical information effectively and accurately
B. use their interpersonal skills when working in a team environment  
C. use interpersonal skills when working with those outside of computing

**Goal V: Professional Responsibilities:** Students will be provided with a foundation for continuing education and growth in the field of computing.

**Student Learning Outcomes**

Students will:

A. recognize the need for continued professional and educational development  
B. be prepared for graduate study or a professional career in computing.

Skill levels for the outcomes would vary amongst programs thus reflecting the individuality of the institution. The learning outcomes articulated above can be mapped to ABET’s Program Outcomes [2]. The mapping in Table 1 relates ABET’s new criteria for 2009 [6] to our SLOs. ABET’s outcomes are separated into outcomes for all of computing (a through i) and additional outcomes for Computer Science (j and k).

**Table 1: ABET Mapping**

<table>
<thead>
<tr>
<th>ABET’s Program Outcomes for CS</th>
<th>Our SLOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) An ability to apply knowledge of computing and mathematics appropriate to the discipline</td>
<td>II.A</td>
</tr>
<tr>
<td>(b) An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution</td>
<td>I.B</td>
</tr>
<tr>
<td>(c) An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs</td>
<td>I.A, I.C, II.C</td>
</tr>
<tr>
<td>(d) An ability to function effectively on teams to accomplish a common goal</td>
<td>IV.B</td>
</tr>
<tr>
<td>(e) An understanding of professional, ethical, legal, security and social issues and responsibilities</td>
<td>III.A, III.C</td>
</tr>
<tr>
<td>(f) An ability to communicate effectively with a range of audiences</td>
<td>IV.A, IV.B, IV.C</td>
</tr>
<tr>
<td>(g) An ability to analyze the local and global impact of computing on individuals, organizations, and society</td>
<td>III.B</td>
</tr>
<tr>
<td>(h) Recognition of the need for and an ability to engage in continuing professional development</td>
<td>V.A</td>
</tr>
<tr>
<td>(i) An ability to use current techniques, skills, and tools necessary for computing practice</td>
<td>V.B</td>
</tr>
<tr>
<td>(j) An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices [CS]</td>
<td>II.B, II.C</td>
</tr>
<tr>
<td>(k) An ability to apply design and development principles in the construction of software systems of varying complexity [CS]</td>
<td>I.C</td>
</tr>
</tbody>
</table>
PROGRAMMING RUBRICS

Since computer programming is a central part of the computing discipline, computer programming assignments are often used to measure whether one or more SLOs are being met. Course outcomes in programming courses would typically map to SLOs associated with Goal I, Critical Thinking and Problem Solving and some or all of the SLOs associated with Goal IV, Communication and Interpersonal Skills. For instance, written communication skills could be assessed in the readability of the written code and in the interaction with the user. Interpersonal skills with those outside of computing could be assessed if software is written for general use. Interpersonal skills in a team environment could be assessed for team programming projects. More specifically, SLOs associated with Goal II, Theoretical Foundations, could be assessed for a given program that simulates finite automata, or compares memory management schemes or develops a heuristic for a Hamiltonian circuit (Outcomes A, B, and C, respectively) and SLOs associated with Goal III, Ethical Responsibilities, could be measured by a programming assignment which implements encryption algorithms.

The first step in developing a generalized programming rubric would be to identify the specific SLOs that are to be measured. The selections here are SLO I.B (algorithmic solution), I.C (program design), IV.A (user interface and code readability). The next step would be to decide on how many levels of performance is appropriate. Too few levels can make it difficult to fit a specific situation into a performance level and too many levels can make it difficult to develop meaningful differences among performance levels. The generalized rubric uses four levels: Unacceptable, Poor, Good, and Excellent. The last step is to describe performance at each level. One strategy might be to begin by describing the highest level of performance for the outcome and then back off slightly for each lower level. Scoring a rubric involves assigning a value or range of values to each of the levels of performance. For example, Unacceptable: 1, Poor: 2, Good: 3 and Excellent: 4. For a finer granularity, one could use a scale such as: Unacceptable: 0-3, Poor: 4-6, Good: 7-8, and Excellent: 9-10. Table 2 contains the generalized programming rubric.

<table>
<thead>
<tr>
<th>Table 2. General Programming Rubric</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Solution I.B</strong></td>
</tr>
<tr>
<td>An incomplete solution is implemented on the required platform. It does not compile and/or run.</td>
</tr>
<tr>
<td>Few of the selected structures are appropriate. Program elements are not well designed.</td>
</tr>
<tr>
<td><strong>Program Design I.C</strong></td>
</tr>
<tr>
<td><strong>User Interface IV. A</strong></td>
</tr>
</tbody>
</table>
### Rubric Application in CS1

The first step in developing a rubric for a specific programming assignment would be to relate each element of the general rubric to the specific details of the programming assignment at hand. The next step would be to determine the extent to which the general description should be modified to reflect student performance on the program. It may also be helpful to suggest specific situations that would merit each of the performance levels whenever possible.

The general rubric shown in Table 2 may be adapted to a Java assignment where the student is to design a program that utilizes inheritance to display a snowman in the base class and then make the snowman melt in the derived class. The program statement includes these pertinent program requirements: a two part snowman, coal eyes and buttons, a hat, a conversation bubble, a sun, and each time the user clicks on the sun the snowman is to shrink by some factor and the original conversation bubble that said “Hello” should display “I’m shrinking.” The specific programming rubric for this assignment is displayed in Table 2a.

<table>
<thead>
<tr>
<th>Code Readability IV.A</th>
<th>Insufficient program documentation, incorrect indentation, and/or poor identifier selection.</th>
<th>Program is minimally documented, some identifiers are inappropriate or inconsistent indentation.</th>
<th>Some required documentation is missing, or identifiers are inappropriate, or statements are not indented correctly.</th>
<th>All required documentation is present, the program is correctly indented, and appropriate identifiers are selected.</th>
</tr>
</thead>
</table>

#### Table 2a. Specific Programming Rubric for CS1

<table>
<thead>
<tr>
<th>Solution I.B</th>
<th>Unacceptable</th>
<th>Poor</th>
<th>Good</th>
<th>Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td>An incomplete solution is implemented on the required platform. It does not compile and/or run.</td>
<td>A completed solution is implemented on the required platform and uses the compiler specified. It runs, but the shrinking factor is inappropriate. Apply Poor if the program does not permit multiple clicks or the amount of melting is too drastic for a single click.</td>
<td>A completed solution is tested and runs without errors, but the shrinking is not applied correctly. Apply Good if a student’s algorithm does not consider both size and location of all pieces of the snowman.</td>
<td>A completed solution runs without errors. It meets the specifications and works for all user mouse clicks.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parent and Child Design I.C</th>
<th>Unacceptable</th>
<th>Poor</th>
<th>Good</th>
<th>Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td>The parent is not designed appropriately to facilitate inheritance.</td>
<td>Some methods or attributes are not appropriately placed and/or extraneous items are placed in a class.</td>
<td>The parent and child classes contain appropriate methods and attributes, however access to attributes and/or communication are not well designed.</td>
<td>The parent and child classes are efficient, reliable and use appropriate methods, communicatio and objects</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User Interface IV.A</th>
<th>Unacceptable</th>
<th>Poor</th>
<th>Good</th>
<th>Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td>The graphic is not correctly displayed. Apply this when student’s parent code does not display base shape correctly.</td>
<td>The snowman does not melt. Use this category when parent displays, but child’s code does not display the melting snowman.</td>
<td>The snowman melts, but does not look natural. This may be applied if the student “shrinks” the hat or the coal eyes.</td>
<td>The snowman is correctly formed and shrinking looks natural.</td>
<td></td>
</tr>
</tbody>
</table>
Rubric Application in Compiler Design

As another example, the generalized rubric can be adapted to assess the third phase, intermediate code generation, of a semester long group project in a senior level compiler design course. Each group is provided with a lengthy assignment statement of the quads (three operands and one opcode) they are to generate from their top-down parser that they wrote for the second phase. The grammar includes complex expressions, strings, integers, floats, arrays and sub-programs. For this assignment the rubric is used to assess Outcome I.A, designing a solution from an abstract model, Outcome IV.A, program readability and Outcome IV.B, teamwork. The specific rubric for this assignment is displayed in Table 2b. Since program readability was covered in the previous example, it will not be displayed in the table.

Table 2b. Specific Programming Rubric for Compilers

<table>
<thead>
<tr>
<th>Solution Modeling I.A</th>
<th>Unacceptable</th>
<th>Poor</th>
<th>Good</th>
<th>Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>An incomplete solution is implemented on the required platform, and uses the specified language. It does not compile and/or run.</td>
<td>A completed solution is implemented on the required platform, and uses the specified language. It runs, but the intermediate code generation does not use the top down parsing model appropriately for any non-trivial constructs. This would be applied if the project only implements basics such as the symbol table, simple assignments and expressions.</td>
<td>A completed solution is implemented on the required platform, and uses the specified language. It produces intermediate code for complex expressions with precedence, and control structures including ifs and loops.</td>
<td>A completed solution is implemented on the required platform, and uses the specified language. It produces intermediate code for subroutines and arrays.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teamwork IV. B</th>
<th>Unacceptable</th>
<th>Poor</th>
<th>Good</th>
<th>Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td>The student did not complete the tasks assigned</td>
<td>The student completed tasks assigned, but not in a timely manner and thus had a negative impact on the team.</td>
<td>The student completed the tasks assigned in a timely manner, but was not helpful to the morale of the group</td>
<td>The student completed the tasks assigned in a timely manner and was an asset to group morale.</td>
<td></td>
</tr>
</tbody>
</table>

A generalized rubric is a template for assessing typical programming assignments that provides the faculty with a starting point and format for assessment. By using a rubric, the faculty can ensure a more objective analysis of the learning outcomes. In addition, the analysis process will be facilitated since the key program traits have already been mapped to course assignments.
Using Rubrics

Reporting the results from the data collected by the rubrics can be done in a variety of ways. For example, the report could consist of the number and percentage of students who scored in each level on each of the traits or the mean of each outcome, as depicted in Table 3. This data could also be reported graphically. Wherever possible, it is important to be consistent with how data is reported from other assessment methods (e.g., do not use percentages for surveys and means for assignments) and with the scales that are used (e.g., if excellent for assignments is a 5, then excellent for portfolios should also be a 5). The means and percentages could also be weighted if that is deemed appropriate. It is expected that students will be assessed multiple times for each of the outcomes using different assessments. For example, Outcome IV.A that addresses oral and written communication skills is measured in the programming assessment but could also be measured in any oral or written assessments in other courses. In addition, measurements for each outcome could also be taken at the beginning, midpoint and end of the CS program. For example, Outcome I.C, program design, could be measured in 2 programs in CS I and 2 programs in CS II at the beginning of the CS program Scores for each outcome would then be aggregated over all of the assessments that measure the outcome as shown in Table 4.

<table>
<thead>
<tr>
<th>Assignment</th>
<th>N</th>
<th>Unacceptable(1)</th>
<th>Poor(2)</th>
<th>Good(3)</th>
<th>Excellent(4)</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Freq</td>
<td>Pct</td>
<td>Freq</td>
<td>Pct</td>
<td>Freq</td>
</tr>
<tr>
<td>Solution I.B</td>
<td>25</td>
<td>3</td>
<td>12%</td>
<td>5</td>
<td>20%</td>
<td>12</td>
</tr>
<tr>
<td>Program Design I.C</td>
<td>25</td>
<td>4</td>
<td>16%</td>
<td>7</td>
<td>28%</td>
<td>9</td>
</tr>
<tr>
<td>User Interface IV.A</td>
<td>25</td>
<td>2</td>
<td>8%</td>
<td>4</td>
<td>16%</td>
<td>12</td>
</tr>
<tr>
<td>Code Readability IV.A</td>
<td>25</td>
<td>1</td>
<td>4%</td>
<td>3</td>
<td>12%</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outcome I.C at Beginning of CS Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Program 1: CS I</td>
</tr>
<tr>
<td>Program 2: CS I</td>
</tr>
<tr>
<td>Program 1: CS II</td>
</tr>
<tr>
<td>Program 2: CS II</td>
</tr>
<tr>
<td>Overall</td>
</tr>
</tbody>
</table>

To determine if students are making satisfactory progress in reaching each outcome by the end of the CS program, these scores are then measured against predetermined benchmarks that faculty have set as reasonable expectations for their students. For example, for Outcome I.C that addresses program design, faculty might expect 70%, 80% and 90% of the students to achieve a level of “Good” or better at the beginning, midpoint and end of the CS program, respectively. Using means to measure performance,
benchmarks might be set at 2.8, 3.0 and 3.2 or better. It is important to note that there are no hard and fast rules when developing rubrics and benchmarks; they must adhere to the purpose and expectations of the individual CS program. We have presented here some examples that could be modified to meet faculty established requirements for assessment.

What happens if students fail to meet the benchmarks for an outcome? Suppose, as in our example in Table 4, only 56% of the students at the beginning of the program achieve a score of “Good” or better for Outcome I.C on program design with a mean of 2.6. First, it is important not to overreact to data and make drastic changes based on a single poor result. However, if the entire department or a department subcommittee on assessment concludes that these results demand action, then they must determine what changes should be made to improve student performance. Initiatives such as 1) more detailed feedback to students on their program design, 2) more examples of good program design, 3) student-led discussions on evaluating program design of sample programs, 4) closed labs targeted at program design could be suggested for implementation. Once changes have been made, the results need to be tracked for at least one assessment cycle to determine if student performance has improved. If not, then other initiatives might be tried or it may be determined that the benchmark was unrealistic or the rubric should be modified. This process, often referred to as “closing the loop” is at the heart of a successful assessment plan.

CONCLUSION

The focus of this paper has been on the process of creating rubrics and benchmarks to measure student learning outcomes. Specifically, we presented a generalized programming rubric and demonstrated how it could be adapted to two specific programming assignments. We then discussed the critical and often neglected process of what to do with the collected data, since assessment for the sake of creating reams of data is a useless endeavor. The results must be carefully analyzed in light of the established benchmarks and if weaknesses are recognized as true problems, then solutions must be devised and implemented. Results must then be re-examined in the next assessment cycle to determine if the changes improved student performance. If not, the process must begin again and tracked until a viable solution has been found.

The examples presented in this paper should be taken as guides rather than mandates. Assessment plans must be tailored to the needs of an individual department and designed to meet the goals and outcomes that derive from the mission of the department and the institution. It is important to keep in mind that data can accumulate very quickly so particular care should be taken to decide in advance what data you wish to collect and how you will use it [1]. If you stick to the plan and stay focused on the goal of program improvement to enhance student learning, then assessment can be a most valuable enterprise.
REFERENCES


ASSESSING COLLABORATIVE AND EXPERIENTIAL LEARNING

Mihaela Sabin
Computer Information Systems Program
Division of Science and Technology
University of New Hampshire
(603) 641 4144
mihaela.sabin@unh.edu

ABSTRACT
Collaborative and experiential learning has many proven merits. Team projects with real clients motivate students to put in the time for successfully completing demanding projects. However, assessing student performance where individual student contributions are separated from the collective contribution of the team as a whole is not a straightforward, simple task. Assessment data from multiple sources, including students as assessors of their own work and peers’ work, is critical to measuring certain student learning outcomes, such as responsible team work and timely communication. In this paper we present our experience with assessing collaborative and experiential learning in five Computer Information Systems courses. The courses were scheduled over three semesters and enrolled 57 students. Student performance and student feedback data were used to evaluate and refine our assessment methodology. We argue that assessment data analysis improved our understanding of (1) the assessment measures that support more closely targeted learning outcomes and (2) how those measures should be implemented.

1. INTRODUCTION
Student direct experience with real-world team projects and discovery and inquiry-based pedagogical methods are well researched approaches to effective learning. Computing education research extensively supports collaborative and experiential
learning, as demonstrated by studies presented at the SIGCSE, SIGITE, and the Consortium for Computing Sciences in Colleges’ conferences. Team projects further improve student learning if they are developed with the participation of industry partners [1, 3, 4, 5] or non-profit organizations [7, 8]. Integrating projects with external clients into course curriculum, however, has its challenges. One challenge in particular, assessment, is the focus of this paper.

When team projects are a major course requirement, assessing student performance raises specific problems: (1) individual student credit should be derived from credit that is earned by the entire team and is based on the overall quality of project deliverables; (2) there are other assessors, besides the instructor, who are directly involved with the project development process and, consequently, assessment data should be collected from those sources, too. Assessment models that address these problems have been proposed in the computing education community [2, 6, 9].

In this paper we describe our experience with the evaluation of an assessment methodology that we designed for three courses in our Computer Information Systems program. The courses, Database Design and Development, System Analysis and Design, and System Implementation with DBMS, implement a collaborative and experiential learning model in which students work on teams to develop real-world projects for community partners. The learning model implementation has progressed in two stages based on our evaluation of student feedback and assessment data of student performance. In the rest of the paper we present the proposed assessment methodology; discuss findings from analyzing the assessment data we collected; introduce some assessment revisions; and show a preliminary evaluation of the redesigned assessment methodology.

2. ASSESSMENT METHODOLOGY

2.1 Curricular Framework

In spring 2008, two Computer Information Systems (CIS) upper level courses at the University of New Hampshire (UNH) pioneered a collaborative and experiential learning model that involved three non-profit local organizations. The courses enrolled 23 students, who formed seven teams. Student teams conducted site visits and interacted with the users of the proposed projects. Students assumed different roles pertaining to the team tasks, presented team work products in class, offered feedback to the other teams, and made public presentations and demonstrations, including participation in the poster session of the UNH Undergraduate Research conference.

Team project requirements had three areas of interest: 1) the product, what teams delivered; (2) the process, how teams worked; and (3) presentations and demonstrations, what and how teams told their clients, peers, and outside world about their work. Students learned in the course by working on project assignments: project releases, public poster preparation and presentation, project report and demonstration; and by participating in team work, such as interaction and communication within the team and with the client. Students were responsible for holding weekly team meetings and making three client site visits.
2.2 Assessment Instruments

In an effective instructional model, course requirements are mapped to learning outcomes through adequate measures of student achievement of those outcomes [10]. The courses in our study shared the goals of preparing students to: identify and analyze user needs; design and implement a computer-based system that meets those needs; use and apply concepts and practices in core information technologies; function effectively on teams to accomplish a common goal; communicate effectively with a range of audiences; and allocate and manage effectively time on task. The team project requirements were reflective of these goals, and student achievement was measured through a variety of assessment instruments used by three different categories of assessors: students, the instructor, and external evaluators, such as clients, other CIS faculty, or IT professionals involved with our program. Assessment provided by students took two forms: self-assessment and assessment of their peers. Student grade in the course was entirely based on project-related work.

Table 1 summarizes the mapping of course requirements to assessment measures and their corresponding assessors. Points awarded to team members on an individual basis had two sources, the instructor (I) and self/peer (S/P) evaluations. Note that only 8 points (last entry in Table 1) of the final grade were based on student work that was done individually. The rest of the 32 points awarded to each student individually were based on student participation on the team as assessed by him/herself and peers. Based on methodologies proposed by [2, 9], we developed a self/peer evaluation form with a total of 20 criteria in five areas: communication, interaction, process, contribution, and responsibility. Students were asked to (1) score themselves and their peers using a 1 to 5 scale for each criterion; (2) comment on their individual responsibilities within the group and on team members’ performance; and (3) quantify the relative contribution of each team member (using percentage values totaling 100%). The formula proposed by Clark et al. [2] was used to calculate how student quantification of members’ relative contributions (including self) translates into actual self/peer awarded points.

Table 1  Course requirements mapping to assessment measures and corresponding assessors

<table>
<thead>
<tr>
<th>Requirement</th>
<th># Points Awarded and by Whom</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>To all team members</td>
</tr>
<tr>
<td>Four project releases</td>
<td>36 by instructor (I)</td>
</tr>
<tr>
<td>Final project report</td>
<td>4 by peers (P)</td>
</tr>
<tr>
<td>Project poster presentation and project demo</td>
<td>8 by peers and external evaluators (P/E)</td>
</tr>
<tr>
<td>Client site visits</td>
<td>6 by external evaluators (E)</td>
</tr>
<tr>
<td>Project weekly reports</td>
<td>6 by instructor (I)</td>
</tr>
</tbody>
</table>
2.3 Assessment Evaluation Results and Lessons Learned

2.3.1 Time on Task

Our collaborative and experiential learning model proved successful in making students spend the required time outside class. In a student survey administered at the end of the semester, students reported that they spent almost twice as much time outside class (5.52 hours/week on average) than in any other course with student team requirements (2.86 hours/week average). Even the individual portion of that time was 21% higher than the typical outside class time for any other course. Our college is a commuting school and 99% of the students work. The job time reported by students in our study averaged 29.3 hours/week, which was almost identical to 29.5 hours/week that students spent in and outside class for a CIS course.

A total time demand of almost 60 hours/week on average poses a very serious challenge. Student time outside a CIS class was divided among team meetings, client site visits, online collaboration to prepare project assignments, and presentation at public events. The student survey overwhelmingly indicated that finding time for team work outside class was the most critical problem they encountered in the course. To address this problem, we adjusted the implementation of our model by (1) scheduling student-client interactions during class time; (2) consolidating some of the project deliverables; and (3) having students include team work process reporting in the project release artifacts. We have also eliminated the formal client evaluation of team work processes. They were invariably a source of maximum scores and were confirming the merit of the partnership overall. Clients have very limited time for engaging in a more rigorous evaluation of student professional behavior.

2.3.2 Team Work and Communication

Student perceptions of team work showed the highest and strongest agreement on the essentiality of team work to the success of the course project (average score of 4.85 and standard deviation of 0.47). The highest and strongest disagreement was on the team members’ equal contribution to the project completion (lowest average score of 3.64 and largest standard deviation of 1.33). Survey questions asked students to rate their own contribution as well as their team members’ with respect to effective communication, significance of contributed work, and level of responsibility and dependability. In general, students scored higher their own contribution and participation than those observed at their peers. Second to the lowest score was the team’s ability to adequately mitigate internal conflicts without the instructor’s intervention. The largest gap between how students perceived their own participation versus their team members’ was noted on the question about effective and timely communication.
2.3.3 Student Performance

In Figure 1 we show how the distribution of final grades compares with the distribution of performance results obtained from individually made contributions (assessed by the instructor solely) versus contributions made collectively through team work on project deliverables and process (with self, peers, instructor, and external evaluators as assessors).

We notice that team performance (counting up to maximum 92 points of the final grade) dictated the student final grades, which ended up the B to A bracket. Individual performance, on the other hand, with the exception of A grades, lagged behind team performance and final grades in the C- to B bracket. The very small weight of 8 points awarded for individually submitted work and the lower grades students received for this type of work explain the discrepancy between individual performance and student final grades. To address the problem of slackers who were getting free rides, we redesigned the assessment scheme to include considerably higher weights for student individual performance. In the assessment redesign we present next, we introduced exams and homework assignments students were asked to do on their own.

ASSESSMENT REDESIGN AND PRELIMINARY FINDINGS

The revised assessment scheme was applied to three courses in Fall 2008 and Spring 2009. Four additional community partners sponsored the course real-world projects and 34 CIS majors in ten student teams carried out those projects.

Two main changes have been made to the assessment methods. First, we complemented the course project with an individual component that weighs half of the final grade and is primarily assessed by the instructor (Table 2). A small portion of the homework grade was obtained through student self-assessment of the drafts of their homework assignments.
Table 2: Assessment measures mapped to course requirements that include an individual student component in addition to the course project.

<table>
<thead>
<tr>
<th>Individual Student Component</th>
<th>Pts to all team members</th>
<th>Pts. assigned individually</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exams</td>
<td>N/A</td>
<td>30 (I)</td>
</tr>
<tr>
<td>Homework Assignments</td>
<td>N/A</td>
<td>15 (I), 5 (S)</td>
</tr>
<tr>
<td>Project Component</td>
<td>Pts. to all team members</td>
<td></td>
</tr>
<tr>
<td>All team project deliverables</td>
<td>25 (I), 15 (P/E)</td>
<td>10 (S/P)</td>
</tr>
<tr>
<td><strong>Total: 70 (I) + 30 (others)</strong></td>
<td><strong>40 (I/P/E)</strong></td>
<td><strong>60 (S/P/I)</strong></td>
</tr>
</tbody>
</table>

Second, we simplified considerably the self/peer evaluation rubric, which now uses three criteria only: on time completion, effective and timely communication, and effective teamwork. The quantifier of relative contribution of each team member has been simplified, too, and lets students impose deductions on the project deliverable grade based on questionable or substandard team work observed of themselves and peers.

Complementing team projects with exams and homework assignments that assess student learning on an individual basis (60% of the final grade with 45% graded by the instructor) has the desirable effect of aligning individual performance with final grade and team performance pattern (Figure 2). Again, the highest concentration of A’s is provided by team performance. However, the individual performance, ultimately, dictates the final grade.

Figure 2: Distribution of grades (final grades and individual and team performance results contributing to the student final grade). Maximum 60 points of the final grade counted towards individual performance.

We conclude this paper with comparative results of the analysis of student feedback on team work and team communication for the two phases of our evaluation.
These results point to an even higher and stronger disagreement on “everybody contributed equally” question (average of 3.52 and standard deviation of 1.45). We also notice a larger gap between how students perceived their own level of responsibility and communication effectiveness versus what they observed about their team members. Further investigation of the assessment methodology is needed to understand how we can improve team work and communication and student satisfaction with a collaborative and experiential learning model.

**Table 3** Student feedback on team work and communication

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Spring 2008</th>
<th>Fall 2008 and Spring 2009</th>
</tr>
</thead>
<tbody>
<tr>
<td>Everybody contributed equally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>My team mates communicated effectively</td>
<td></td>
<td></td>
</tr>
<tr>
<td>My team mates showed responsibility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>My team mates contributed significantly</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team mitigated all teamwork issues</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teamwork was important</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I communicated effectively</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I showed responsibility</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I contributed significantly</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


LESSONS AND TOOLS FROM TEACHING A BLIND STUDENT*

Dr. Richard Connelly
Department of Mathematics and Computer Science
Providence College
Providence, RI, 02918
(401) 865-2132
rconnell@providence.edu

ABSTRACT
This paper relates the experience of teaching a blind computer science major in three courses and part of a fourth. For programming, a header file (pcspeak.h) was created that connects the standard C++ iostreams to eSpeak, an open source text-to-speech system. Braille handouts were created using a Braille printer that embosses Word documents.

1. INTRODUCTION
It is demanding to teach computer science to the blind in today’s graphical world [3, 13] and the challenges are student dependent [3, 12]. Like Califf, et al. [3], the intent is to relate the experience to aid others. In the command line era, computer science was considered a possible career for the blind [1, 13].

The student is a computer major who is now blind, but had partial vision into high school. He was a pleasure to teach, because he is academically strong, reliable, flexible, and has a sense of humor. He is very proficient in Braille and similarly skilled with the JAWS® [5] screen reader and the PAC Mate® [5] note taker / reader.

Three courses and part of a fourth were taught across one academic year. The first course was scheduled, but the others were done on short notice due to faculty illness. The first course also differed, because it was “off” term and one-to-one.
2. TOOLS

The first course (Computer Science II) is the second of the core programming courses. It covers object design, linear structures, and an introduction to recursion. The author and a mathematics colleague needed “tactile line” tools. A search yielded dimensional paint (e.g., [11]), raised line drawing kits (plastic paper with a rubberized pad) (e.g., [10]), and Wikki Stix [17] (paraffin coated yarn). The latter worked well, but paint lines tended to flatten and plastic paper lines were not very tactile.

The PAC Mate® reader converts text files (txt or doc) to Braille and displays them line-by-line using pop-up pins to create Braille characters. The author teaches with handouts, so the intent was to use text handouts with separate tactile diagrams. An option was to have parts of a textbook commercially converted to Braille, but it was unclear if diagrams and special layouts would be converted.

The blind student and another were scheduled to take the course. The plan was to have them work as partners with considerable teacher and tutor support as had been done in Computer Science I. When the other student transferred, the author used Braille handouts in class and converted graphical labs and projects to text equivalents.

The first handout attempt used a Braille labeler and Wikki Stix. It worked, but was too time consuming. The mathematics colleague noted that the student owns a Tiger Braille Printer [16] which was readily lent to the faculty. The printer can emboss 20 dots per inch which far exceeds Braille requirements. It embosses Word [7] documents by converting text to Braille and converting tables, diagrams, arrows, etc. to high density dots. It uses a 29 point font which required considerable layout planning. Even in landscape mode, many of the handouts had foldouts that required cutting and taping. Considerable time was needed to plan and emboss a document, but it made possible what was otherwise impossible.

Without a partner, the student had no peer to explain I/O and graphics. Computer Science II uses C++, so the author built an audio header file (pcspeak.h) for the standard iostreams. The pcout, pcin, and pcgetline operators replaced cout, cin, and getline respectively. Each invoked the underlying operator and then sent the text to an open source text-to-speech system (eSpeak [4]). eSpeak is synchronous, so a program halted until the speech completed. The student heard prompts, input, results, and used pcout and pcin to debug. Here is a code sample:

```c
#include "pcspeak.h"

string s;
int n;
pcout << "Enter a string: " << pcendl;   //pcout replaces cout; pcendl replaces endl
pcgetline(s);   //pcgetline(s) replaces getline(cin, s)
pcout << "Enter an integer: " << pcendl;  //pcout replaces cout; pcendl replaces endl
pcin >> n;     //pcin replaces cin
```

The graphical project based on L-Systems [9] was replaced by a text version of the mouse-in-the-maze. The Braille practice mazes were very effective. The graphical Klondike Solitaire project was replaced by a text version of Clock Solitaire. The postfix
project only required substitute characters for the plus and minus operators and the space, because eSpeak can speak punctuation. The pcspeak system made further changes unnecessary.

3. COMPUTER SCIENCE II

Since the course was one-to-one, it was easy to innovate as needed. As examples, Braille dominoes were used to demonstrate cyclic buffers, dominoes and Wikki Stix to demonstrate linked lists, and Braille call stacks to demonstrate recursion. Class time was often extended so that the student could practice and completely read the Braille.

The student programmed in text using his PAC Mate®. Someone else transferred the file to C++ and ran it in Visual Studio [7]. The student usually used audio (JAWS®) to read code. By observation, this meant programs had to be modest sized. It was essential that programs be built in parts.

A computer science tutor taught separate tutoring sessions. Worked solutions were provided and she used them, the pcspeak system, and visualization (e.g., a bank queue) to teach. She required a thorough analysis before coding and step-by-step coding. When typing for the student, she did not edit. She was very effective, because the author did no additional tutoring.

The tests were open book with a practice test and a review session. The author administered the tests himself. Initially, it was unclear how to restate and clarify material. It was agreed that the student should “say everything at each step”. This allowed restating without coaching. Dominoes and Wikki Stix were used to answer cyclic buffer and linked list questions.

4. OTHER COURSES

4.1. Background

On short notice, the author taught the blind student in Assembler Language and Database Management Systems and completed Operating Systems. The latter only required creating the final and agreeing on an administration method. For the others, there was insufficient time to commercially convert parts of texts to Braille. Further, Assembler Language does not use a text and database texts have many diagrams that might not be converted.

Normally, the author teaches with handouts only, although students can borrow textbooks. The handouts contain concepts, definitions, facts, and one or more worked examples. The latter are presented on a board and/or computer. Students do additional practice problems either individually or in groups using paper and/or a computer. The practice prepares them for the tests which are open book. The handouts often have textboxes with arrows that step through algorithms, practice, etc. From course surveys, sighted students like the “handout paradigm” and expect it.

The number of diagrams, special layouts, etc. precluded using text handouts with the PAC Mate®, so the author created both text and Braille handouts. Because of the short notice, the Braille handouts were not available before class. This made it difficult
for the blind student, because he cannot read Braille as quickly as sighted students can read text. Further, Braille versions of textboxes and other side bar items usually required foldouts which were difficult to navigate. Since the blind student cannot do practice problems at the same rate as sighted students, some handouts contained answers. This changed the paradigm, especially for sighted students.

The author administered the Braille tests himself. Before each test, he and the blind student reviewed the practice test and agreed on an administration method which was a variation of the “say everything at each step” method from Computer Science II.

4.2. Assembler Language

The course uses the SPIM [14] assembler which displays all its registers on the screen. The language is usually learned by stepping through programs and watching the registers. This method was not workable for the blind student. A text-to-speech system tailored to the SPIM screen would have helped.

Teaching from the front using a graphical screen does not help the blind student. Instead, the author had the students work in groups and he circulated among them. The blind student’s partner was his tutor from Computer Science II. She was both his partner and tutor. They often started concurrently (one read Braille, the other did set up) and then did an analysis before coding. They also developed techniques to explain her SPIM screen and to transmit data to and from his PAC Mate®. She was very effective, because the author did no additional tutoring.

Assembler language was very difficult for the blind student. The handouts contained programs filled with special symbols and unusual layouts. For example, the number symbol (#) starts a comment in SPIM, but starts a number in Braille (# as a symbol is several characters in Braille). Without a text-to-speech system, he had to learn by listening to his partner step through programs.

4.3. Database Management Systems

The course has a wide scope and includes labs (Access [7] and aspx [7]), concepts (Normal Forms, SQL, etc.), and many diagrams (ER, UML, etc.). Rapid comprehension of the handouts is integral to covering the material.

The wide scope meant there was a range of challenges. In Access, the sighted students created SQL statements graphically while the blind student created them as command lines. The aspx syntax required complex Braille foldouts and the corresponding labs had only graphical interfaces. Fortunately, aspx was the last topic and by then students could comprehend it abstractly. The other handouts had many diagrams which often overwhelmed the blind student, because they could not be completely read in class. The author tried more verbal presentation techniques, but the extra time jeopardized completing the course. Each typed page produced three to five Braille pages; in hindsight, there should have been a “helper” to manage the handouts. (The student revealed later that helpers were used in high school and that each school tended to reinvent the wheel.)
No tutor who knew the material was available. Initially, the blind student worked informally with several classmates, but, eventually, he worked with a classmate who became one of his tutors. He scheduled tutoring with the author which allowed him to absorb the material in his own style. He was very aware of his strengths and weaknesses and knew how to prepare for a test.

5. COMMENTS AND SUGGESTIONS

Tutors are essential as shown by teaching a course without one. Teaching or tutoring a blind student requires innovation and dedication. It can be exhausting and frustrating, but is very rewarding. The author would do it again and at least one tutor has expressed the same.

Teaching a blind student should not be assigned on short notice. It is unfair to the instructor and student alike. The former needs time to prepare handouts so the latter can to read them before class. If Braille handouts can be read beforehand, the handout paradigm would be more satisfactory. Even with proper notice, some subjects may not be feasible. Computer architecture was not because of the engineering diagrams and “bread board” kits.

The pcspeak system is one of the few tools for blind programmers. Others include a scripting language [13] for designing Visual Basic [7] forms, scripts [15] that enable navigation of Visual Studio using JAWS®, and PLUMB EXTRA [2]. The latter, in development, uses speech and sound effects as the user traces and/or creates a graphical representation of a data structure. There is a desperate need for more tools.

W3C has web page accessibility guidelines [18]. Since they are intended for various disabilities, it may be better to tailor them for the blind [12]. Web pages designed for screen readers (e.g., JAWS®) would be very helpful. Such pages might reduce the need for Braille pages. Friere, et al. [6] created a course that uses the guidelines and screen readers to educate web designers about the needs of the blind.

6. ACKNOWLEDGEMENT

The author sincerely thanks Barbara Leasher, Ph.D. who suggested dimensional paint, Wikki Stix, and the Tiger Braille Printer. Dr. Leasher taught the student calculus with these tools plus Model Magic [8] which was used to create tactile conic sections.

7. REFERENCES


DESIGNING VALUE SENSITIVE SOCIAL NETWORKS FOR THE FUTURE

Jami Cotler and Janelle Rizzo
Computer Science Department
Siena College, 515 Loudon Road, Loudonville, NY 12211
jcotler@siena.edu, je08rizz@siena.edu

ABSTRACT

With the dramatic increase in popularity and growth of social networks it has never been more important to teach our students studying computer science to be mindful of the values they impart into the design of social networking applications. By approaching the design of a social network using a value sensitive framework a deeper understanding of all stakeholders and the potential implications and benefits to them may be discovered. This study uses a value sensitive design approach combined with an analysis of how social networks are currently being used. Based on the findings, general social network design recommendations are offered.

INTRODUCTION

Social networking has become an integral part of all of our lives. Throughout our day we will see social networks mentioned in advertisements, news reports, and magazines across disciplines. It is hard to go through a day without hearing the words Facebook, Twitter, LinkedIn, MySpace or Flickr. This emergent and pervasive technology offers a wealth of opportunities for students studying computer science. Learning how to navigate a social network has become an important skill for success in the 21st century. Understanding at a deep level how social networks operate is quickly becoming a critical skill for computer science students interested in working as part of a social networking development team.

Batya Friedman and colleagues developed a tripartite methodology for integrating human values into the design process. [5] Value Sensitive Design (hereafter, VSD)
includes a conceptual, technical and empirical analysis of the problem space (in this case social networks). In the conceptual analysis, the design is evaluated by identifying both direct and indirect stakeholders. [5] Once acknowledged, the benefits and potential implications of the design to each stakeholder are considered. [4] This approach provides a deep understanding of the stakeholders and often reveals value conflicts that ought to be addressed in the design. This study focuses on the findings of a conceptual value sensitive design analysis of the social network Facebook. Design recommendations and ideas for future study are offered.

BACKGROUND

Problem Space/Context

With over 300 million registered users Facebook has quickly emerged as the virtual place to be. [6] [8] People from around the world are flocking to this social networking website from multiple demographics. With this attention comes a social compromise many of us aren’t aware we are making, while other less obvious stakeholders benefit. As the 6th most trafficked website in the US [6] this famous website is attracting attention from other groups, such as potential employers and cyber criminals.

Value(s) implicated

Three central human values are implicated in the system design of Facebook. The value of privacy often conflicts with the pursuit of visibility. [2] As our social networks and connections have moved on-line our privacy has been compromised, often without our direct awareness or informed consent.

Facebook offers the user the ability to select networks. Networks emulate users real-world connections, such as university affiliation. As a default, Facebook profiles are open to everyone in the users designated network. While we “voluntarily” post our profile, pictures, and personal information on Facebook, many users do not understand the social significance and potential implications of having this information easily available to the potentially thousands of users in their designated network.

A person with knowledge of web design and a few extra minutes can easily compromise the privacy of Facebook users. [3] Often when Facebook users make their profiles private they will maintain public friend lists in order to promote their visibility. While maintaining public friend lists offers visibility it also opens vulnerabilities to the privacy afforded to the user. If a “friend” on the friend list has an open profile it can be used as a direct link to that particular user. This can be accomplished with basic knowledge of the Facebook URL structures. Even without web design knowledge one can find any postings or images tagged by the person of interest in the open profile of a friend. Postings are messages left in a public space in a Facebook users profile. Pictures are “tagged” identifying Facebook users in the pictures.
Direct Stakeholders Identified

The direct stakeholders are the Facebook users. The Facebook users may use the social network to keep in touch with friends and family. Facebook has become the social “meeting place” for people of all ages. [1] [8] Businesses are quickly becoming another direct stakeholder as they use Facebook for marketing and customer outreach. Other businesses are creating applications that can be used within Facebook.

Indirect Stakeholders Identified

In a study conducted by Rosen and Kluemper they demonstrated that employers, while not actively using the social network in the same way as the direct stakeholders, are lurking and often use Facebook to research potential employees. [7] They found that many employers don’t look at actual Facebook profiles but find a much more revealing disclosure of information from the job candidate’s friends. [7] Potential employers will analyze the comments and tagged pictures posted by the candidate’s friends. According to their study, some employers also relate the number of Facebook friends to the popularity and extraversion of the candidate. Other employers look for revealing signs of speaking ill of former employers, evidence of excessive drinking, or revealing too much confidential information. [9] The opportunity for this type of disclosure of information increases with the number of friends, as the potential for a friend to have an open profile increases.

The Internet in general and Facebook specifically can be rapture for a stalker. Where else can one find a list of hundreds of friends of a potential victim? The amount of information that can be gathered in minutes would have taken days, if not weeks without Facebook. With an alarmingly high rate of users displaying full friend lists, the potential for a stalker to quickly and easily find this information is quite real.

Value and stakeholder conflict

<table>
<thead>
<tr>
<th>Value</th>
<th>Facebook Users/Marketing Businesses/Application Developers</th>
<th>Employers/Cyber criminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Privacy</td>
<td>Privacy settings are available but implications of not using them are not clear.</td>
<td>These indirect stakeholders appreciate profiles using default or open privacy settings as it allows them to view and learn more about the person in question.</td>
</tr>
<tr>
<td>Visibility</td>
<td>Facebook users often want visibility and achieve this through large friend lists and multiple tagged photos. This is in direct conflict with and often compromises privacy.</td>
<td>This indirect stakeholder appreciates highly visible profiles as it compromises security settings and allows them increased access the profile in question.</td>
</tr>
<tr>
<td>Informed consent</td>
<td>Consent for openness is the default and presents a social compromise not often comprehended.</td>
<td>The ethical issue can be raised that a job candidate generally does not provide informed consent to a potential employer to look at their Facebook profile or related Internet postings. Employers who view this information often do so without the knowledge and/or consent of the candidate they are interested in hiring. Some potential employees never get a chance to be interviewed or hired because of information found in Facebook about them. Future employers can also learn of gender, age, sexual orientation, and social habits which often are not legal to ask on a job application or interview. This information is all obtained without (informed) consent of the future employee. Without a clear example of how the profile may be viewed, a potential employee often is not aware of how vulnerable their (perceived private) information is.</td>
</tr>
</tbody>
</table>

**Project Goals**

The goal of our initial study was to determine if and how the identified values (privacy, visibility and informed consent) are being addressed by the design of Facebook. Profiles of a variety of groups of students were studied to determine what privacy settings were selected as well as other information such as number of friends.

Ultimately, the study was designed with the following two outcomes in mind. By better understanding how our students are using social networking and privacy settings we can use this information to educate our students to become more informed users of social networks. Secondly, we can apply this deeper understanding and incorporate it into our pedagogy when teaching our computer science students how to design social networks. Design suggestions are made to address the shortcomings found.

**METHODOLOGY**

The study was comprised of 264 undergraduate student Facebook profiles. The researchers viewed the public profiles of each of the students to determine the openness, privacy settings and number of friends of each student. This study was followed up with brief interviews with six students who had open profiles.
RESULTS

In this study the primary network of the students studied (Siena College) contained 6,064 users. Different majors were studied with the most interesting being business majors. Our focus was on this group of students. The sample size studied was 169 student profiles. In this group 28% had profiles that were open to the entire primary network, in this case 6,064 Facebook users. This group had an average number of 378.61 friends. This directly translates into creating privacy vulnerabilities for all friends of the user with the open profile. We also investigate the number of profiles displaying friend lists. We found that 71% of the student profiles viewed showed a full list of friends. The average number of friends for this group was 469.48.

In the pursuit of visibility; namely having a profile open to the primary network and displaying how many friends a users has, privacy is substantially compromised. We suspected this is happening without the knowledge or informed consent of the Facebook user. The research was followed up with interviews where questions about privacy and profile settings were discussed. Overwhelmingly the students interviewed were shocked, surprised and very concerned to find out that so many could see information they thought was private. None of the students interviewed stated they wanted their profile open to their primary network to promote visibility.

RECOMMENDATIONS/PROPOSED TECHNICAL SOLUTION TO THE VALUE CONFLICTS IDENTIFIED

It was clear that the users studied were not aware nor did they consent to have so many view their profiles. In the privacy settings, Facebook offers a way to view profiles as a friend. We propose that options to view the profile as someone who is not a friend (either in or out of your network) is also offered. A prototype of what this could look like is displayed below. This option would provide a clear way of seeing how the “world” sees your digital Facebook footprint. By adding this type of functionality to the design of a social network, the concept of informed consent is addressed. This idea can be extended to the design of other social networks.

Figure 1
Social networks have amazing potential to change and enhance the way we connect and socialize as humans. While embracing the capabilities, it’s important to maintain awareness and preempt potential VSD compromises. Awareness is the key to understanding how to best protect yourself while navigating the social networking highway. This information needs to be disseminated as a part of fundamental education.

INCORPORATING THE FINDINGS INTO THE CS CLASSROOM

A course being taken this year by one of the researchers has allowed direct application of the knowledge obtained through this study to be applied to an assigned project. The course is a senior level full year Software Engineering course in which students work with real clients to develop a variety of applications. The project assigned to the researcher was to develop a new social networking site, but a site only available to members of the Siena College community. This project coincides greatly with the research being done on social networks for this paper and has allowed greater insight and design for the new site. The researcher was named team leader for the task of developing a new social networking site and because of this research, privacy settings were one of the top issues being considered in the design of the course project.

Privacy settings are very important and the users of social networking sites need to be aware of how much of their information is available to the public for viewing. Since learning more about Facebook and the privacy features of that site and how open and available most information is there, the team developing the site for the Software Engineering course made the decision that by default, all personal information for users will be made private. Enough information, such as name, picture and an option to become a “friend” remains visible to facilitate networking and promote the growth of the social network. If after the initial setup of information for users, the individual user decides they want their information to be more visible; users will have the option of making information public in an informed way. It is ultimately the users choice whether or not they want their information to be displayed and available to others, and there is nothing wrong with having that information viewable to the public, as long as consent is given by that individual user and the user is made aware of exactly what is being displayed.

The software engineering project strikes a good balance between providing privacy in an informed way, while maintaining visibility to promote the growth of the social network as a direct result of our research results.

FUTURE WORK

In the spring 2010, we plan to extend our study with more in depth analysis of open surveys with follow-up face-to-face interviews with students and potential employers.

In our initial study we found the business students with the greatest occurrence of having an open profile were marketing majors. We would like to study this in greater detail to determine potential contributing factors.

We also noted that gender seemed to make a difference if a profile was open. The sample we selected had approximately a 50% male/female split. When we looked at only
the open profiles we found that 37% of the users with open Facebook profiles were women. We plan to explore this in greater detail.

CONCLUSION

Based on our initial findings, it is clear that while social networks such as Facebook are designed to inspire and promote visibility they often fail to provide privacy in an informed way. Many Facebook users are making a social compromise they are not aware they are making. By designing social networks of the future with more methods to determine how an account will look when it is viewed by the world would provide a social network more sensitive to the values of the primary stakeholders.

WORKS CITED


EXPANDING CS1: APPLICATIONS ACROSS THE LIBERAL ARTS

Bridget Baird and Christine Chung
Department of Computer Science
Connecticut College
New London, CT 06320
860 439-2008
bbbai@conncoll.edu

ABSTRACT
This paper will describe how applications in a variety of disciplines can enhance the teaching of the CS1 course. Examples will be given from a range of disciplines, including mathematics, economics, linguistics, history, biology, art and music. The applications will be linked to the computer science concepts being discussed. Such an approach broadens the appeal of the introductory course and also teaches students valuable problem solving skills.

INTRODUCTION
There have been a variety of approaches for teaching introductory computer science that have been used to entice students into the discipline. Some of these include the use of animation [10], film [7], or web design [2]. Other approaches have integrated the natural sciences into the computer science curriculum [1, 5]. At the same time that computer science is broadening its reach into other disciplines, there has also been an increased emphasis on problem solving techniques in the CS1 course. Some of these techniques have been quite extensive and complex [4, 9]. Others stem from the research by Margolis and Fisher [8] that shows that women are more likely to major in computer science if they can see the tangible applications and ways in which computer science can be used. Problem-solving activities also seem to increase the self-confidence of women taking CS1 [6]. Since the last Taulbee survey [3] shows that 11.8% of undergraduate degree recipients were women and nearly two-thirds of the undergraduate majors were white and non-Hispanic, it is imperative that efforts continue to be made to broaden the...
appeal of CS1 to underrepresented groups. Thus, using problem solving across a wide range of disciplines is a natural approach.

At smaller colleges the CS1 course has multiple audiences: serving as a base for those desiring to major or minor in CS but also serving as an introduction to the discipline for students who might only take one or two courses in the subject. Thus one of the first goals of the course is to produce a challenging but accessible and interesting course for all audiences. Another goal for this course, obviously, is to introduce basic programming concepts in computer science. Discussions at the college also focus on two additional goals. The first is to emphasize the importance of computer science in solving a variety of interesting, practical problems; students should become active problem solvers themselves. And then the final (implicit) goal is to interest a broader group of students in the joys of computer science; in the best case they are enticed to become majors and active practitioners and in any case all students finish the course with a greater appreciation for the discipline and a confidence in their abilities to apply these concepts to their own fields.

The language that is used to teach the introductory course is Python. This language seems particularly well suited to meeting the goals of this course: it is relatively easy to teach and learn, it does not have a lot of jargon, the programming environment is free and thus promotes continued use beyond the end of the course, and there is a wide range of modules that provide accessibility to a host of applications.

**CONCEPTS AND APPLICATIONS**

As the basic concepts of computer science are taught and as students develop proficiency with the language, an effort is made to show these concepts in use, linking them to practical applications. The choice of applications reflects teaching in a liberal arts college, where students from all disciplines take the introductory course. The institution has a long history of supporting interdisciplinary collaborations and includes a number of centers, including one in arts and technology (which requires this CS course). Thus the students not only come from a variety of backgrounds but also have a certain expectation about combining different fields in their studies. Table 1 shows the broad links among concepts and applications that are used in this course.

<table>
<thead>
<tr>
<th>Application</th>
<th>Programming Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linguistics, History, Political Science, English</td>
<td>String/file processing, conditionals, loops, arrays (or Python lists/data collections), searching and sorting algorithms</td>
</tr>
<tr>
<td>Art and Music (image and sound processing, graphics, animation)</td>
<td>Nested loops, modules, recursion, event-driven programming, functions and parameters</td>
</tr>
<tr>
<td>Biology, Genetics</td>
<td>Object-oriented programming (including inheritance), string/file processing</td>
</tr>
<tr>
<td>Mathematics, Games, Simulation</td>
<td>Pseudorandom numbers, software engineering, object-oriented programming, algorithms</td>
</tr>
<tr>
<td>Statistics, Finance, Economics</td>
<td>File processing, graphics and visualization, web-crawling, algorithms</td>
</tr>
</tbody>
</table>

Table 1: programming concepts and the applications used to illustrate them
String Processing for Text Analysis

One of the strengths of Python is that it allows for early accessibility to file and string processing. These concepts and an early introduction to conditionals and loops mean that within the first few weeks of class, students can write intriguing and interesting applications involving large texts.

One cross-disciplinary application is analyzing texts of famous literary works (which can easily be obtained online from websites like www.gutenberg.org). The students can study literary styles of different authors (average sentence length, average word length, commonly used words or phrases), or they can alternatively perform word processing tasks on the texts (find/replace, spell check, etc.). These same techniques lend themselves to analyses of historical documents such as diaries, census information and historical texts.

Another compelling application is the problem of analyzing presidential or campaign speeches. For example, students can be given the task of comparing the 2008 convention speeches of Senators Barack Obama and John McCain. They can be asked to compare the number of times each candidate mentioned specific words, drawing conclusions about their priorities or campaign styles. They could also easily modify the speech to have all instances of a certain word replaced by another, creating a new file with an interestingly altered version of the speech. This particular application happens during the third week of class, giving students an early idea of the extensive and interesting programming concepts at their disposal.

Later in the course, when students have learned some basic graphics, and have been taught arrays/lists, searching and sorting, they can apply these skills to writing a program to generate word clouds. A word cloud is a graphical representation of the words from a body of text wherein more frequent words from the text are displayed in larger fonts, providing an interesting visualization of the given text. Such visualizations have recently been popularized by a number of internet websites. This engaging and multi-faceted task requires students to exercise skills such as processing text from a file, counting word frequencies, sorting the words in order of frequency (using parallel arrays or Python dictionaries), searching for and removing any “stop words” (less important words that should not be included in the word cloud such as “the” and “is”), and displaying them in a graphical window in a pleasing and randomized fashion so that more frequent words appear larger and closer to the center.

Functions and Recursion for Graphics and Animation

A nice way to illustrate the power and flexibility of parameterized functions is by using graphics. A basic beginning exercise might be to ask the students to draw a simple shape (like a square or circle) whose color is specified as a parameter. Other properties of the shape can then of course be parameterized to teach students how to create functions with multiple parameters. The use of a graphics package to create custom shapes using student-defined functions transitions nicely into the introduction of boolean functions, which can be used to compute various geometric properties of the shapes being drawn.
Once students understand basic graphics (which also gives them an early taste of object-oriented programming) and loops, it is instructive to have them create graphical animations. Even simple animations are useful for teaching a number of lessons: they can again be used to help students practice writing modular and parameterized functions (e.g., speed or direction of the animation can be parameters), they can be used to show creative uses for loop counters (at this point it is still early in the course and students are still not quite comfortable with the idea of using the built-in definite-loop counter variable within the body of the loop), but it also gives students a sense of the processing speed of the computer. While they may feel many other tasks could be done manually (if slower), animation exercises can concretely affirm to them that there are tasks where the role of the computer is truly indispensable.

More complicated graphics introduce visualization of data. By using graphics as well as functions with parameters, students are asked to take actual data sets from economics (such as census information, employment statistics, etc.) and decide how best to visually impart the data. They often choose data which has particular meaning to their own interests. Although they produce fairly complicated and interesting graphics, this is still a topic that occurs quite early in the course, in the first month.

Students start out in the course writing console applications, and in that setting it is naturally harder for them to relate to the context of the program’s flow (as most students have only used applications with graphical user interfaces). Thus, giving them an early chance to create a graphical user interface with clickable buttons creates a more familiar user-setting that helps to motivate indefinite loops and event-driven programming techniques. Because of their extensive experience as users of GUIs, students can fairly easily grasp the concept of a user’s button-click driving the control of program flow.

Recursion is often a challenging concept for students. An engaging way to illustrate the power of recursion and allow students to really experience it is to have students create drawings of fractals. Using a simple graphics package, students can create a straightforward program that draws straight lines and keeps track of changes in direction. Then, after some practice with simpler recursive images, the students can be given the task of creating more complex, well-known and beautiful fractals, such as the Koch snowflake and Sierpinski Triangle. They can also be given the opportunity to create/devise their own fractal drawings.

**Nested Loops and Modules for Image and Sound Processing**

Image processing is an appealing and convenient way to not only incorporate art into an introductory course, but also to demonstrate nested looping to students. Students are able to get a concrete grasp for the purpose and structure of nested loops when they can physically see each of the pixels of a two-dimensional image being processed. Interesting applications for the students include: changing all pixels of a certain color in an image; analyzing paintings (was Picasso’s Blue Period really “blue”); changing the hue, brightness, or saturation of images; and doing some basic pattern recognition of images by finding outlines of objects.

Sound processing of wav and MIDI files presents more opportunities to illustrate the power of computer programs to manipulate information. It also demystifies the
structure of wav files, showing them to be collections of bytes that can be controlled and changed. Some of the programs for this application include generating simple songs and tunes, doing simple analyses of wav files, and altering wav files (for example, changing the volume).

As students look at the quantity of data necessary for pixel information (and then extrapolate to video) and also look at the size of wav files, this presents an opportunity to talk a bit about compression methods, how important they are, and how crucial good algorithms become. The processing of images and sounds also presents an opportunity to show the power and convenience of external, already-written modules. Students not only learn how to interface with these modules (reinforcing the versatility of functions and parameters, the importance of modular programming, as well as object-oriented programming concepts like encapsulation), but also realize the potential of tapping into a well-established but evolving body of knowledge.

**Pseudorandom Numbers for Games and Simulation**

When programming a game or simulating game play, there is a natural need for random number generation. With this motivation, incorporating pseudorandom number concepts into the curriculum becomes natural.

The famous Monty Hall problem is a great application for demonstrating the power of simulation using random numbers. In it, the game show host, Monty Hall, asks the game show contestant to choose one of three doors. Only one of the doors has a prize behind it. After the contestant chooses a door, Monty opens one of the doors not chosen by the contestant that also does not have the prize behind it. He then gives the contestant the chance to change his/her mind about which door s/he chose. It is well known and easily shown (if counter-intuitive) that the contestant is better off switching his/her choice of door. The students can create a program to simulate this famous game show and via simulation compute the likelihood of winning when the contestant chooses to stick with his/her original choice compared to when s/he switches. It is a nice way to affirm a counter-intuitive theoretical result for introductory-level students.

There are many other games and simulations that illustrate uses of probability, functions and parameters and user interaction. Some of the simulations used in this course have included Black Jack, Monte Carlo simulations for estimating pi, baseball simulations to analyze the occurrence of “streaks” and “batting slumps,” and tennis simulations. As students develop reasonably complicated games, there is also an opportunity to discuss software development, top-down and bottom-up design, and the importance of planning in order to write a cohesive program. Students are required to incorporate these concepts in their large, final project, which is on a topic of their own choosing.

**Object-Oriented Programming for Genetics and Biology**

Object-oriented techniques enter the CS1 course in many places: the structure of Python itself, string processing and file manipulation, introduction of graphics, game simulations, etc. Another place where this concept is introduced to the students is in a
lovely application in genetics. Genomes are composed of DNA molecules which contain sequences of millions of bases (A, G, C, T). These molecules contain chromosomes, which in turn contain genes. It turns out that genes can be identified by using straightforward string processing techniques (they are subsequences with certain easily identifiable starting and stopping codons). This topic presents an opportunity to talk about classes, inheritance and methods for those classes. Data for this application are taken directly from the National Center for Biotechnology Information (NCBI) website. Students are able to download strands of DNA and then pick off the genes. They can then compare genes in different organisms, looking for identical ones or ones that are similar. Once again they are able to see both the computing power that is necessary for such applications but also the possibilities for asking and answering interesting questions about this discipline.

Other areas in science where object-oriented techniques and external data have been used include environmental research information obtained from colleagues. Often this information is contained in spreadsheets. In most institutions one can find colleagues who have data sets that can be mined for interesting conclusions.

Web Crawling for Statistics and Finance

Students bring great familiarity with the web into this class and it is important to show them how they can write computer programs to tap into this resource. The applications in this section range from the straightforward (taking data from a single source) to web crawling (where the program moves from one web site to another). The simpler applications illustrate the use of current data that is easily found online (such as stock prices, or data on recent earthquakes from the US Geological Survey) to produce informative statistics. The more complex programs involve opening a web site and then using information from that web site in interesting ways, including moving to web sites contained in its links. These more complicated interactions lead to a discussion of the complexities of recursion (every computer has limits that can be quickly reached), the importance of searching and sorting algorithms, and the importance of algorithms in general. It is a good place to talk about search algorithms, such as Google’s algorithm for searching the web, and what those mean for computer science.

Sources for Applications

Some of the applications, such as some of the fractal drawings and several of the simulations involving random numbers, are taken from the text by Zelle [11] that is used in the course. Many others come from conversations with colleagues. The authors are happy to share their ideas, experiences and course materials (including labs, homework and class activities).

CONCLUSION

This cross-disciplinary CS1 curriculum has been created and shaped over several semesters. It has been well-received and seems to have attracted an increased number of majors to the computer science department. The fact that the number of female computer
science majors is greater in our department than in neighboring colleges of comparable size and quality might also be attributed in part to our introductory curriculum.

The CS1 course is constantly evolving as different kinds of applications are introduced. In all cases an effort is made to show “real” problems and not artificially-produced ones. In the future there will be applications that incorporate some of Google’s applications and other kinds of mashups. Students are very familiar with using many kinds of Internet applications and it is beneficial for them to see how these services can be brought to bear under their control in a computer program. Another area being examined is computational chemistry, where one of the central pieces of software that allows chemists to study and manipulate visual representations of molecules is written in Python. Efforts are being made to see if an interface is possible so that students can write Python modules that plug into the software and manipulate the data. The philosophy of this CS1 course is that it should illustrate computer science as a dynamic, relevant discipline with beautiful structures and functional techniques.

It is important that in the setting of a smaller liberal arts college, computer science is incorporated as much as possible into the liberal arts curriculum, rather than introduced in isolation from the many other rich areas of study that the students expect to be exposed to. While students should have the chance to appreciate computer science for its own sake, it is also vital that students come away from an introductory course with an understanding that computer science permeates all areas of study and is a fundamental part of a liberal arts curriculum.

REFERENCES


INTEGRATING RESEARCH PROJECTS IN CS1

Deborah Sturm  
College of Staten Island (CUNY)  
(718) 982-2848  
Deborah.Sturm@csi.cuny.edu

Sarah Zelikovitz  
College of Staten Island (CUNY)  
(718) 982-2849  
Sarah.Zelikovitz@csi.cuny.edu

ABSTRACT

This paper discusses our experience integrating research topics into a first course in computer science. We introduced a medical image processing project using two-dimensional arrays in order to expose students to research topics early in their computing education career.

INTRODUCTION

Many studies have reported on the benefits of introducing undergraduates to research in computer science [2,6]. These studies cite engagement of students in learning as well as evidence of increased retention in the major. However, they also report on the challenges of placing inexperienced students in research labs while ensuring that the student and the mentor both benefit from the experience [7]. In addition since research funding is limited, relatively few students profit from this enrichment. These findings have led to an inclusive approach with the integration of research projects into the undergraduate computer science curriculum. These projects are usually included in senior level capstone courses or in advanced electives [1,5,8]. Some have attempted to introduce research topics early in the curriculum [3,4].

Project Goals

Our goal was to expose students at the beginning of their educational career to research topics in computer science. This pilot study was meant to examine whether it is feasible to include relatively advanced real-world projects in a course in which students are just beginning to develop their skill set. To do this, we wanted students to write the
code directly without relying on pre-written classes. Moreover we were interested in whether the students, given this exposure, would be more likely to major in computer science.

One major challenge is that the students in our introductory computer science course are not necessarily computer science majors. This course is required for engineering science and mathematics majors as well. In addition, as in other institutions, the first course has a high attrition rate with approximately half achieving a C or better.

**METHODS**

Two introductory computer science classes participated in this study. This course is highly coordinated with each section assigned the same textbook and programming exercises. The two sections (each with approximately 30 students) were essentially equal except for scheduling with students choosing sections without knowing which was experimental. The control section programmed a tic-tac-toe program. While this exercise is fun and demonstrates indexing into two-dimensional arrays, it is relatively easy and does not necessarily show students how difficult real-world problems can be solved using programming. The experimental section was given an additional lecture with an overview of research topics in computer science with a focus on medical image processing. The motivating project was to process MRI images with possible MS lesions.

The companion programming assignment was to process an MRI image using two dimensional arrays. The students were given the basic code to read a .bmp file by first reading header information and then the image data (figures 1 and 2). In class they went over the code to invert the image (figure 3b). The assignment was to create two binary output images: a mask of the outline of the brain and a thresholded image based on user input (figure 3c).

The IDE used was Visual Studio which allows one to view image files from within the interface. Thus the student could easily view the original image and the processed output image.

**Outline of Exercise Given to Students**

First view the original MRI.bmp image using Visual Studio. Then:

1. Read header info – number of rows, columns, number of colors
2. Dynamically allocate the two-dimensional array
3. Read the image into the two dimensional array
4. Process the image according to the exercise
5. Write it to an output file
6. Open the output file from within Visual Studio
RESULTS

Students were engaged in the lecture, actively participating and asking meaningful questions. Significantly, all students were able to complete the project on time. We administered two surveys to each of the two sections participating in this study to assess their attitudes towards research at the beginning and end of the semester. The questions are shown in Table 2.
One surprising result was that while in both sections over 90% of the students responded that they had never worked on a research project in computer science, over 50% felt that they had a clear idea what research entailed. In each section there were relatively few students officially majoring in computer science (around 25%). At the end of the semester the students’ interest in working on computer science projects had predictably declined. One encouraging finding was that for the students in the enriched section the decline was lower (down 12% compared to 27% in the control group).

Although the surveys did not show a statistically significant difference in the attitudes of students in the two sections, our goal of introducing an achievable research project in a beginning programming course was met.

<table>
<thead>
<tr>
<th>Pre-Survey</th>
<th>Control section</th>
<th>Integrated Research section</th>
</tr>
</thead>
<tbody>
<tr>
<td>I have a clear idea of what research in CS entails</td>
<td>Agree:50%</td>
<td>Agree:50%</td>
</tr>
<tr>
<td>I’ve worked on a Computer Science-related research project</td>
<td>Yes: 4% No: 96%</td>
<td>Yes:7% No: 93%</td>
</tr>
<tr>
<td>I am interested in working on a Computer Science-related research project</td>
<td>Agree: 78%</td>
<td>Agree: 72%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Post-Survey</th>
<th>Control section</th>
<th>Integrated Research section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compared to the beginning of this semester, I now have a better understanding of real-world research projects in Computer Science</td>
<td>Agree:70%</td>
<td>Agree:73%</td>
</tr>
<tr>
<td>I am interested in learning more about research in Computer Science</td>
<td>Agree:66%</td>
<td>Agree:70%</td>
</tr>
<tr>
<td>I am interested in working on a Computer Science-related research project</td>
<td>Agree:51%</td>
<td>Agree:56%</td>
</tr>
</tbody>
</table>

Table 2. Survey questions and results

DISCUSSION

Our pilot study showed that it is feasible to create simple entry-level research-based exercises that students can understand and complete. Moreover, the project provided an opportunity to explain when dynamic allocation and pointers are necessary (when sizes are only known at run-time). We learned that one intervention, in the earliest possible course, is not sufficient to increase motivation in students to pursue research. We hope to introduce a more extensive image processing exercise to form the histogram using a one dimensional array and then finding using this data to generate a threshold value to form the binary output image. Similar more advanced projects that compare adjacent pixel values can be introduced into CS2 that build on the material covered in CS1. In CS2 one can develop image classes with methods to simplify the organization of the code. Further study is needed to determine whether exposure to research topics early on will lead to better retention rates.
ACKNOWLEDGEMENT: This project was partially funded by the CUNY LS-AMP grant under the Integrated Research Strategies program.

REFERENCES


AN INTRODUCTORY COMPUTATIONAL COURSE FOR SCIENCE STUDENTS

Sohie Lee and Ellen Hildreth
Computer Science Department
Wellesley College
Wellesley MA 02481
781.283.3123
slee@wellesley.edu

ABSTRACT

This paper describes an introductory CS course, in a liberal arts setting, designed for science students to learn programming using MATLAB. This course differs from many introductory CS classes in that 1) students are not expected to continue taking more CS classes and 2) MATLAB is used in a general computational context rather than in an engineering setting. Students learn to write software to solve problems, visualize and analyze data, perform computer simulations, and implement and test computational models that arise in a wide range of scientific disciplines. The course culminates with an individualized final programming project in which students apply their MATLAB skills in their particular area of scientific interest.

INTRODUCTION

Science requires computation. There are data to be analyzed, changes to be measured, and predictions to be made. The manner in which established scientists learn about computing, however, is typically haphazard [4]. At liberal arts colleges, from which graduates go on to earn PhDs in the sciences at a higher rate than other undergraduate institutions [1,2], training in computing is essential. We designed a CS0 course to teach science students about computation using MATLAB.

At the time the course was developed, our CS department offered two introductory courses: a Java based course for CS majors, and a course for non-CS majors on web...
design using HTML and JavaScript. Our goal was to create a CS0 course aimed at science and social science majors who do not intend to enroll in further CS courses. Science and social science students need to learn computation, not as computer scientists per se, but rather as scientists who need to compute.

MATLAB was chosen because it is a powerful and widely used technical computing environment with advanced graphics, visualization and analysis tools, and a rich high-level programming language. MATLAB is widely available on campus. One advantage of MATLAB is that students can quickly generate meaningful programs and results, an important factor in introductory programming classes. In our course, MATLAB is a tool for teaching computation, much in the vein of Kaplan [3,4], as opposed to a CS0 course targeting engineering students [6].

CURRICULUM

Our goal was to provide a solid foundation of basic programming concepts, with an emphasis on computing that is particularly useful in the sciences. Accordingly, our topics fall into three main categories: 1) Data representation and organization, 2) Program structure and flow and 3) Data analysis and modeling. In data representation, we cover the various ways that scientific data, both numerical and textual, can be stored in MATLAB. This includes variables, matrices, strings, structures and cell arrays. For program structure and flow, we teach the fundamentals of writing individual functions using conditionals and iteration (including recursion), as well as the higher level skill of how to design a large program using multiple functions and subfunctions. The design process emphasizes the important ideas of abstraction and modularity. We also teach how to create graphical user interfaces (GUIs), and merge the underpinnings of a program with the user interface. For the third category of data analysis and modeling, we cover reading external files of many types, writing files in a useful format, visualizing 2D and 3D data, and selecting and sorting data. We also introduce the powerful MATLAB toolboxes. Specifically, our students learn about the Curve Fitting, Image Processing and Statistics Toolboxes, as well as the Interactive Plotting Tool.

The course meets three times per week for 13 weeks. Each week includes two 70-minute lectures and one 110-minute hands-on laboratory. During lecture, students often engage in group problem-solving. The laboratory revolves around generating MATLAB code that instantiates concepts from the previous week’s lectures. There are typically 8 homework assignments, each requiring 6-8 hours on average, that are completed outside of class. There are also two in-class examinations and a final project.

Our homework assignments have illustrated many different ways that MATLAB can facilitate problem-solving and computation. Students have written MATLAB programs to perform tasks often needed in the world of science. Sample tasks included: examining the frequency of occurrence of structures or events, e.g. identifying a foreign language given occurrences of the most common letters [5]; simulating dynamic processes, e.g. population growth or spread of disease in a population; performing statistical analyses, e.g. removing outliers in a data set, or extrapolating from a data set to make predictions; performing graphical analyses of data, e.g. energy production and consumption data, and supply and demand data; creating synthetic images, e.g. visual illusions and recursive
pictures; image processing, e.g. counting grains of rice using luminance data and analyzing the spectral composition of astronomy images; processing text, e.g. translating nucleotide sequences to amino acids [3] and selecting or sorting numerical or textual information in a database, e.g. manipulating data in a bird species database. Our choice of programming tasks emphasized general problem-solving skills that are relevant in the sciences.

We wanted to build a level of comfort and expertise with MATLAB within a single course so that after completing the course, students would naturally turn to MATLAB (vs. Excel, Maple, and other statistical packages) for routine tasks like plotting and analyzing data as well as for more complex programming needs. MATLAB has the tools for these routine tasks seamlessly integrated into a general programming environment. This provides much greater flexibility for students to customize their data analysis and visualization tasks, and to build a program that can integrate many analysis steps efficiently, and a GUI that allows the user to work with the data interactively. With this in mind, our course culminates in a final project where each student builds a substantial MATLAB program from scratch, drawing upon the wide array of computational resources that it offers.

A SAMPLE ASSIGNMENT: MRI DATA DISPLAY

Our students wrote an interactive program to display MRI brain slices, using data that comes with MATLAB. The assignment required reading in MRI brain data from a file and then a) offering display options in various orientations (sagittal, coronal, horizontal and vertical) and b) showing a movie of brain slices. Students initially wrote a text-based interface that allowed the user to select which orientation and which slice to view (see Fig 1). Several weeks later in the semester, the students revisited this program, and created a GUI with menus and buttons (see Fig 2) that enabled colorful three-dimensional displays of the same data.

Figure 1. Screenshot of sample brain slice (left) and text user interface (right)
The development of the MRI brain slice assignment in two parts allowed us to emphasize some key computational ideas: 1) the ability to read an external file and display and morph its contents, 2) the division of a large program into smaller functions, and 3) the separation of the program’s function from the program’s user interface. The students were introduced to the MRI data early in the semester, before GUIs were taught. The second MRI assignment used the same data, yet the students wrote a completely different and more complex graphical interface that displayed the brain images in three dimensions. We emphasized the separation between the computation that a program performs and its user interface. One goal of this assignment was to provide a model by which the students understand how to decompose a problem into smaller units, and then build a new, more powerful and flexible user interface for that program.

**FINAL PROJECT**

The final project provides students with an opportunity to integrate and reinforce the many skills learned throughout the course. Each student designs a large project from scratch, typically 2-3 times larger than a homework assignment. Most students design and implement the program largely independently. This process has many benefits for our students. Creating their own project builds students’ confidence and develops their self-reliance and resourcefulness. While working on their projects, students sometimes learn new MATLAB that is particularly relevant for their project. Independent programming on a self-chosen topic also helps students stay highly motivated and appreciate the relevance to their field of interest. Many of our course’s final projects have supported research projects with other faculty, providing our students with a source of pride when demonstrating their project to their research advisors. Faculty in other departments have directed their research students to our course, as MATLAB is a
valuable tool in their research. As our final project requires a series of written reports as well as presentations, this strengthens students’ writing and public-speaking skills. Finally, as a long-term assignment spread out over 6 weeks, the final project requires strong time-management and resource allocation skills.

Students chose final project topics based on their own interests. The final project consisted of four phases: 1) a general text description of the project with sketches, including lists of each MATLAB function or script; 2) a detailed code skeleton with roughly one-third of the code written; 3) an informal presentation of the project in progress during class; 4) a final project demonstration with the course instructors, submission of code and documentation. Table I lists a sample of the wide variety of final projects. Virtually all projects incorporated a graphical user interface; many projects read input data files and output results files.

Table I. Final projects in Computation for the Sciences

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Highway traffic jam formation model, airplane flight simulation, mass-spring system, 3D trajectory of ball under gravitational and wind forces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Educational Tools</td>
<td>Interactive periodic table, visualization of electrical and potential fields around charge distributions, economic forecasting, analysis of bonding in chemical structures, simulation tool for teaching game theory</td>
</tr>
<tr>
<td>Research Assistance</td>
<td>Data coding and analysis for motor learning study, data tabulation from internet search of gene and function correlations, analysis of MRI data from the College MRI facility, analysis of Maine coast tidal data, analysis of eye tracker data from perceptual experiments, analysis of NMR spectral data to determine molecular structures, analysis of peak absorbances and phase changes from chemical wavelength and absorbance data, creation of experimental color perception, analysis of eye alignment in historical images of visual artists</td>
</tr>
<tr>
<td>Fun &amp; Games</td>
<td>Mastermind, Snake, Frogger</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>Travel planner, health risk evaluation, decryption and encryption</td>
</tr>
</tbody>
</table>

CONCLUSIONS

Computation for the Sciences has been taught 5 times in 3 years to students whose majors represent many diverse fields (Biology, Chemistry, Physics, Astronomy, Economics, Computer Science, Neuroscience, Cognitive Science and Psychology). The students ranged from first-years to seniors. Within the broader curriculum at our college, this course satisfies a distribution requirement in the area of mathematical modeling and problem solving. Although this was not the intent of this course, several students who took Computation for the Sciences in MATLAB as their first CS course subsequently declared CS majors or minors. The foundations of programming were sufficiently covered such that students have successfully transitioned into more advanced CS classes.
after our course. Feedback from the graduates of our course has been very positive. In the students’ anonymous end-of-semester evaluations, 92% indicated that they would recommend our course to fellow students (78% indicated they would “strongly recommend” the course). One student summarized her course experience (Spring 2009) as follows: “I wanted to learn how to use a tool that would help me become a better scientist, and maybe learn a bit about computer programming. I'd wholeheartedly recommend this course to students who want to learn a programming language that is useful for the sciences.” We have also received positive feedback from faculty in other departments whose students have taken our course.

Perhaps the most compelling examples are those students who, after taking our course, continue using MATLAB for computation in their independent research projects, or in their field of choice. At our college, MATLAB is widely used in teaching and research in disciplines such as Neuroscience and Physics. Some students in these and other scientific fields have made it clear that knowledge of MATLAB programming from our course facilitates advancement in their scientific careers. For example, one graduate of our course uses MATLAB extensively for her Geosciences thesis project, and is able to efficiently analyze and present her data to her advisor, who is not familiar with MATLAB. Other students are using MATLAB in their graduate or professional work, in areas such as physics and medical imaging.

ONLINE COURSE MATERIALS

http://cs.wellesley.edu/~cs112/courseMaterials

REFERENCES


TEACHING COMPUTER SCIENTISTS TO PLAY WELL WITH OTHERS

PANEL DISCUSSION

Charles Welty
Professor of Computer Science
Computer Science Department
University of Southern Maine
Portland, ME 04104
(207) 780-4240
welty@usm.maine.edu

Jesse M. Heines
Professor & Undergraduate Coordinator
Dept. of Computer Science
University of Massachusetts Lowell
Lowell, MA 01854
(978) 934-3634
heines@cs.uml.edu

Margaret Menzin
Professor of Computer Science & Mathematics
Simmons College
300 The Fenway
Boston, MA 02115
(617) 521-2704
menzin@simmons.edu

ABSTRACT

Most undergraduate computer science programs include classes that require team work. This helps our students work well with each other, but does not address the problem of working well with people from other disciplines. Computer scientists have preconceived notions of people in other professions and people in other professions have preconceived notions of computer scientists. These preconceptions can interfere with good working relationships. Computer scientists tend to work on projects of use in an application field that may be unknown to them and, thus, must work with people in that application field.

1 Contact person.

* Copyright is held by the author/owner.
Many computer scientists enjoy the comic strip Dilbert™ by Scott Adams. A problem with the strip is that it paints stereotypical portraits of the various professionals that Dilbert works with. The marketing people lie and make impossible demands. Management is composed of idiotic, power mad people. Advertising people will promise anything. The human resources department is out for your blood. Certainly graphic artists and usability professionals would come under similar fire if they are ever part of the strip. Of course, the computer scientists (or engineers) are also negatively stereotyped as having no lives, being obsessed with hi-tech toys, having poor social skills and, generally, being geeky (see http://en.wikipedia.org/wiki/Geek). The problem is that these stereotypes often contain a component of truth. Computer scientists and others must learn to look beyond the stereotype and see what a person can actually do.

This panel will look at several ways to foster appreciation of other disciplines to help broaden the sometimes narrow perspective of our graduates.

PANELISTS’ POSITION STATEMENTS

Charles Welty

Using a Graphical User Interface Design course to teach computer scientists to play well with others.

A course in graphical user interface design works well as a delivery mechanism to aid computer scientists to learn better what other people do in designing and building a product. Many assignments and projects in computer science courses start with the functionality and appearance of the software defined in the problem statement. On the other hand, part of our GUI course is deciding what product to produce and then going through cycles of design and user testing to determine the final form of the product. In this process students should learn what team members with backgrounds other than computer science bring to the table.

Our GUI course includes presenters with backgrounds in marketing, graphic arts, usability, ethnography, technical writing and, occasionally, management. Presenters from these other disciplines come into class not to teach the computer science students to be able to do the presenter’s job after a single presentation but to show the breadth and depth of the presenters’ expertise. Then, when the two come in contact at the workplace, the computer scientist has more than the above mentioned stereotypical views.

Jesse M. Heines

Exploring interdisciplinary course models.

Like our colleagues at the University of Southern Maine, we at UMass Lowell believe that there is great value in exposing computer science students to the work of their peers in other disciplines. With the help of a National Science Foundation (NSF) grant, we have tried to go beyond bringing in presenters from other disciplines to give students concrete experience in working on interdisciplinary projects.
In 2007, a team of UMass Lowell professors was awarded an NSF CPATH grant to explore the intersection of computer science and the arts through interdisciplinary courses. The team consisted of two professors in computer science, two in art, one in music, and one in theatre. This team developed two types of courses: "synchronized" and "hybrid."

"Synchronized" courses pair two existing of upper-level courses for majors in two departments. The courses remain independent, but the students work together on a joint project developed within the scope of the two courses. "Hybrid" courses are ones that are taught by two instructors simultaneously, one instructor from computer science and the other from the arts. These courses are open to all students across the university and co-listed in two departments. Science students earn Arts & Humanities General Education ("GenEd") credit, while Arts students earn Technology GenEd credit.

Not unexpectedly, each course model turned out to have pros and cons. Our work taught us a number of lessons that we are now using to revise the courses. As we move through our third year of this project and approach the end of our NSF funding, we feel that we have laid sufficient groundwork for at least the hybrid courses to continue to be offered. Enrollments in these courses has increased over time, and the professors involved remain enthusiastic about working together. The university administration has embraced the effort and allowed the professors to count the joint teaching as part of the professors' normal teaching load.

We see these developments as indicators of a successful program that we expect to continue to grow and help prepare students for the interdisciplinary project teams they will encounter after graduation in the workplace.

Margaret Menzin

Working with a course in another discipline

In my Systems Analysis course I try to present students with opportunities to both interview high level stakeholders and to work with domain experts from another field. Last year one project in the course was the design of an informational website for prospective students. My students had to identify the stakeholders and then write to and interview them: many high level administrators at our university (deans, head of the Library, etc.) and groups of other students to determine what content and functionality should be on the site. We also invited to our class people such as the Vice President of Marketing, who spoke about the institution’s approach to presenting ourselves and answered student questions.

After students had determined the architecture of the site we turned to the Video Production course in our Communications Department to provide content on a specified list of topics. Videographers are creative people and they can produce "fun" content, wonderful videos with humor and "voice". Creative people, however, don’t necessarily follow your specifications. After the students in the Video Production course had made more than a dozen wonderful videos, my students had to turn to other sources to find images and videos for the remaining topics which needed visual material. There were also negotiations with the videographers about whether or not certain scenes in some videos
were appropriate for the new site. And there were some other tensions around maintaining a time schedule (which my students had developed on Open Project), a problem well known in all software development.

In retrospect my students and I didn’t get enough "buy in" from the Communications students, and failed to communicate to them the nuances of the site’s purpose, while the Video Production students didn’t get enough of an understanding of what it means to be on “an assignment”. I think some of these problems could have been solved by scheduling the courses at the same time so that the students could have worked more closely together. On the other hand, both groups of students gained an enormous appreciation of what the others had to contribute to a project and what it means to collaborate across professions. As a measure of the success, we are planning future collaborations between the two courses.

ABOUT THE PANELISTS

Charles Welty, professor of Computer Science at the University of Southern Maine for 30 years, is interested in graphical user interfaces and interface usability. An NSF grant originally funded the lab used in the GUI Design course and other courses. He has been working in the field of usability and, earlier, human factors since 1977.

Jesse M. Heines has been on the UMass Lowell faculty for 25 years after 10 years at Digital Equipment Corporation. He has a keen interest in computer science education and computer applications in the arts, particularly those in music. This interest is currently supported by an NSF grant in which two CS professors are teaching interdisciplinary courses with professors in Art and Music. Jesse's teaching focuses on the implementation and evaluation of interactive, user-centered programs with rich graphical user interfaces (GUIs), particularly those employing Dynamic HTML, JavaServer Pages, and XML and XSL and their related technologies. Jesse has a long record of applying and evaluating these techniques in educational settings.

Dr. Margaret Menzin is Professor of Mathematics and Computer Science at Simmons College in Boston. Her interests in computer science focus on database systems, web services and web centric programming (for which she maintains an extensive on-line annotated bibliography), systems analysis and health informatics. She also has long standing interests in both pedagogy and encouraging women to pursue careers in mathematics and science.

PANEL TIMING

First, there will be a brief introduction to the panel and panelists (3 minutes). Each panelist will then present their position for at most 12 minutes, allowing about 5 minutes for questions following each presentation and 15 minutes for questions after all the presentations.
TEST-FIRST DESIGN PEDAGOGY AND SUPPORT FOR
INTRODUCTORY OO PROGRAMMING

TUTORIAL PRESENTATION

Viera K. Proulx
College of Computer and Information Science
Northeastern University
Boston, MA 02115
617-373-2225
vkp@ccs.neu.edu

The hands-on workshop will show how testing and test-first design can be incorporated into every introductory course on object-oriented programming using Java. We will show the syntax for the tests that use our novice-friendly tester library, and describe the pedagogy we use to enforce test-first design from the very first program students write. The tester library is freely available on our web site with extensive documentation that shows how to use it with Eclipse, NetBeans, BlueJ, or just command-line Java compilation.

Test-driven design or test-first design has been used in software industry for years. The reports on using this approach reveal faster development time, more robust program design, and a more readable code — divided into shorter methods and smaller classes. On the other hand, many reports on catastrophic failures caused by software flaws point out to a small overlooked error in the program design. We all have experienced the times when after struggling with a bug in our code we find the problem in a small method that just cannot be wrong.

One would think that all courses on introductory computing would adopt this beneficial test-driven approach. However, this is not the case. A survey of over twenty current textbooks shows only two that attempt to require tests for the methods students design. The rest of them include a perfunctory section that proclaims how important testing is --- with no attempt to include it in the suggested curriculum.

One of the difficulties in enforcing test-first design for novices is the steep learning curve for use of standard libraries for testing. Students struggling with the syntax-heavy programming language (Java) are not able to digest the extra burden of designing tests in a separate environment (such as JUnit), especially if it requires that they override the
equals method before they even understand what are the different kinds of equality between Java objects.

Over the past six years our introductory computing courses enforced test-first design from the very beginning. To make this possible we have designed a supporting test library, tester, that enables the novice programmer to define all examples of data and all tests in an Examples class that represents the client for the code that the student has designed. There is no new syntax, no need to define equality, as all objects are compared for the value of their fields. Tester evaluates all tests defined in the Examples class, pretty-prints the values of all data defined in the Examples class, and reports on all failed tests with a display of both the actual and the expected values, as well as a link to the failed test. Additionally, there is a support for checking whether the actual value matches one of several expected ones (e.g. a random number being one of 1, 2, 3, or 4), or whether the given value is within the given range (as determined by a Comparator or an implementation of the Comparable interface). There is also a support for detecting whether a method when invoked by the given object and the given argument list throws the expected exception --- with the expected message.

Our curriculum focuses initially on mutation-free program design: the result of every function or method is a new piece of data. This makes the test design easier, as all we need to do is to compare the actual outcome with the expected one. However, the tests for imperative methods (that produce void) are not much harder to design and will be covered in the tutorial.

This workshop illustrates hands-on the benefits of test-first pedagogy on a series of typical introductory programming assignments. It prepares the participants to adopt test-first design approach in their introductory (and advanced) courses. The material presented here is orthogonal to the ReachJava curriculum and can be used in every course that focuses on Object-Oriented programming using Java.

Each participant will receive both paper and electronic copy of workshop notes and exercises. Additionally, the tester library web site contains a tutorial, examples, the tester library downloads and documentation. The participants can also access a wealth of labs, examples, assignments, and lecture notes already available on our web pages.

PRESENTER'S BACKGROUND

Viera Proulx is a Professor in College of Computer and Information Science at Northeastern University in Boston, MA. She has been involved in curriculum and software development for introductory computing for nearly 20 years. During the past six years she has been developing and implementing curriculum for data-driven class based introductory course on object-oriented programming, working with the TeachScheme/ReachJava team.

Professor Proulx has led faculty development workshops and several conference workshops on the ReachJava curriculum. The work on the tester library is supported by the NSF DUE-CCLI grant number 0920182.
REFERENCES


FLIPPING COINS: AN ACTIVITY FOR INTRODUCING COMPUTATIONAL THINKING TO NON-COMPUTER SCIENCE MAJORS*

DEMONSTRATION

David Toth
Computer Science Department
Merrimack College
315 Turnpike Street
North Andover, MA 01845
978-258-1208
david.toth@merrimack.edu

ABSTRACT

In this demo, I will present an activity that I use to introduce computational thinking in my non-majors course and outreach talks at high schools. I do this by first talking about what modeling and simulation are and why they are useful. Following this explanation, I introduce a variant of a problem about Alice and Bob. In the original problem, Alice and Bob take turns flipping a coin until one of them gets heads and wins. If Alice goes first, what is the probability that she wins? In class, I introduce this variant of the problem: “Bob suggests to Alice that they play a game. They will take turns flipping a coin until one of them wins by getting heads. The winner will then receive a dollar from the loser. If Bob says he will always let Alice go first, should Alice play the game?” While this problem is trivial for those comfortable with basic probability and familiar with geometric sequences, most of my students who are not computer science majors are not comfortable with math. Thus, I explain that although the problem can be solved with some math, instead we should first model the problem. Once we have modeled the problem, I have my students get some intuition by pairing off and having each pair play the game 20 times. I collect the results and discuss how so few trials are not sufficient to draw conclusions. Finally, I run a simulation that runs the game one million times and demonstrates that Alice should win 2/3 of the time and thus should play the game.

* Copyright is held by the author/owner.
ENSEMBLE - THE ONLINE COMMUNITY CENTER FOR
COMPUTING EDUCATORS*

DEMONSTRATION

Gregory W. Hislop
Drexel University
hislop@drexel.edu

Lillian N. Cassel
Villanova University
lillian.cassel@villanova.edu

Lois M. L. Delcambre
Portland State University
lmd@cs.pdx.edu

Richard Furuta
Texas A&M University
furuta@cs.tamu.edu

Peter Brusilovsky
University of Pittsburgh
peterb@mail.sis.pitt.edu

Edward A. Fox
Virginia Tech
fox@vt.edu

Daniel D. Garcia
University of California, Berkley
ddgarcia@cs.berkeley.edu

ABSTRACT

This demo will introduce attendees to Ensemble, the online community center for all computing educators. Ensemble provides content, communities, and tools for computing educators and students. The content consists of freely available computing education resources stored within Ensemble or at other locations. Ensemble provides federated search, indexing, annotation, reviews, and other services to make these resources accessible, visible, and more useful to the community. Ensemble also provides information streams such as news, notices, and blogs of interest to computing educators.

Ensemble communities support interaction among computing educators via facilities such as discussion forums, posting of working papers, and connections to venues such as Twitter and Facebook. These services support open collaborations such as a CS1 community site and also hosts closed working spaces for groups like the ACM Education Board and the Future of Computing Education Summit working groups.

* Copyright is held by the author/owner.
Ensemble tools provide access to more advanced facilities to help instructors and students access and organize materials relevant to computing education. An example is Visual Knowledge Builder, which provides a workspace for collecting and organizing computing education resources.

Ensemble supports the full range of computing disciplines and also programs that blend computing with other STEM areas (e.g., X-informatics and Computing + X).

ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation NSDL program under Grants DUE 0840713, 0840721, and others.
PSUzzle: WEB-BASED INTRODUCTION TO PROGRAMMING

PRACTICE

DEMONSTRATION

Delbert Hart
Computer Science Department
SUNY Plattsburgh
Plattsburgh, NY 12901
518 564-2775
hartdr@plattsburgh.edu

DESCRIPTION

PSUzzle is a web application where students using programming concepts to solve puzzles. The application is geared towards students taking CS 0 and CS 1 courses and do not have previous programming experience. The puzzles incrementally take students through the process of building small programs to solve problems by using visual programming. Students start with a simple sequence of statements. As they progress they are introduced to conditionals, loops, and functions and are challenged to solve puzzles with the new components as they are introduced. The application also supports exercises where the students translate the visual programs to and from textual representations. The application tracks each student's progress and generates the puzzles based on their current level of achievement. The goal of the application is to give students confidence and experience in creating programs.

PSUzzle differs from more generic environments such as Scratch and Alice in that it focuses on teaching specific programming concepts. PSUzzle is structured as a sequence of exercises tied to the courses it is used with. Instructors are able to track student progress and provide feedback. The demonstration will show both the student's and instructor's use of the tool. During the student portion of the demonstration a walk through of some exercises and their pedagogical goals will be given. The instructor portion of the demonstration will look at how exercises are enabled and the data collected from them.

Instructions and an online demo of PSUzzle is available at http://www.cs.plattsburgh.edu/psuzzle/

* Copyright is held by the author/owner.
BIOGRAPHY

Delbert Hart is an assistant professor in the computer science department at SUNY Plattsburgh. His past research interests have been in distributed computing and information visualization. His current research interests are in developing instructional tools and computer security. He has a DSc in Computer Science from Washington University in St. Louis.

EXPERIENCE

The presenter is the developer of the tool. He has used a preliminary version of the tool during guest lectures in several sections of a CS 0 course. During the spring he plans on working with instructors of CS 0 and CS 1 to make use of the tool in their courses. He has taught CS 0 and CS 1 courses, as well as a variety of other computer science undergraduate and graduate courses.
ROBOTRAN: EVOLUTION OF AN UNDERGRADUATE SOFTWARE PROJECT

Mark Meyer and Kevin Mastropaolo
Computer Science Department
Canisius College
Buffalo, NY 14208
716 888-2432
meyer@canisius.edu

ABSTRACT
Student software projects occasionally evolve into useful, long-lasting applications, but are seldom documented well, making maintenance by later students and faculty extremely difficult. Dealing with this type of legacy code poses challenges but presents interesting opportunities for students to learn in a deep way why software engineering is important. Skills of structure discovery can be taught and will be useful in a student’s future career in the software industry.

INTRODUCTION
When we undertook the creation of a software simulator for the popular Lego Mindstorms RCX in 2005 [4][5], we did not realize how daunting the task would be, nor how useful and long-lasting the resulting product would be. The simulator passed to other students who all struggled to understand the complex system in order to debug and expand it. This paper details their adventures and attempts to discover some redeeming lessons from the experience.

APPROACHES TO LEGACY CODE
Most software engineering texts explain how to build maintainable systems from the outset. Only a handful of books give practical advice on how to walk into a legacy system and understand it or read its code [3][8]. The difficulty of understanding a large
CCSC: Northeastern Conference

system seems to defeat most programmers due to “long distance,” irregularly patterned dependencies between the modules and sections of a program, even when modern object oriented languages are used [3]. Furthermore, agile methods, which have challenged the traditional view of software documentation, do not appear to help when software needs to be maintained later [7].

Some studies of how programmers approach an unfamiliar program focus on metacognition strategies by which the programmers attack the problem of learning a program’s structure [7]. Surprisingly, programmers continue to rely on the actual source code much more than documentation, structure diagrams or other descriptive tools [6]. In fact, many documentation artifacts rank fairly low on the chart of how important they are rated by programmers, including data models, requirement descriptions, test plans, data dictionary, user manual, architectural model, glossary, and so forth [6]. Some types of global documentation artifacts are consulted once to get a general understanding of the system and are never consulted thereafter [6].

THE ROBOTRAN PROJECT

In 2005 Canisius College built and equipped a robot lab in which to teach elementary robotics, using money from a NASA grant. A number of LEGO Mindstorms RCX robot kits were purchased and used in several introductory courses. In order to teach students programming, we designed a simple imperative language with minimal syntax, called Robolang, and an IDE called Robotran. This IDE converted Robolang programs to their Lejos [2] equivalent, which were then downloaded to the RCX via the infrared tower and run autonomously in the robots.

In order to give students a tool that would allow them to test out their homework solutions even when they did not have an actual robot at their disposal, we designed and created a simulator the next year. David Puehn was the principal architect of the simulator and he also redesigned the IDE for student programs, using Java and Swing. An early version of David’s simulator was showcased in Boston at the AAAI conference in 2006. David did not use a formal software engineering methodology for Robotran, partly because he was the only programmer.

With the imminent advent of the NXT (next generation) LEGO Mindstorms in 2006, David tried to retrofit his simulator. This turned out to be a major problem because Lejos NXJ, the new version of Lejos for the NXT robot, had a number of different classes in the library and its ROM model was quite different from the RCX. David decided to generalize the structure of the simulator software so he created base classes for entities such as Robot and Program and then subclassed them for particular architectures. This extra layer of indirection made the structure supposedly easy to expand in the future but, in the short term, obfuscated the architecture of the application, making it harder to understand, debug and modify.

ATTEMPTING TO DOCUMENT THE PROJECT

By 2007, David had moved on to other projects. A number of bugs lingered as well as a roster of unfinished features. One of the most desirable of these was collision
detection, whereby the simulated robot would stop when it ran into the wall or obstacles that we placed in its on-screen world. In spring 2007, we offered a seminar section that would attempt to document Robotran as well as enhance it. Six students worked on this project in two teams: three worked on documentation and three on bug fixing and programming. The latter group investigated a number of collision detection strategies but failed to get it working.

The documentation group did not fare much better. They investigated several automated UML tools, including Green UML [1] developed at SUNY-Buffalo, that automatically generate class diagrams. Originally, they wanted to create both a programmer’s manual and internal javadoc documentation to enable future developers to more easily maintain Robotran, but got mired down in the complex structure. One of them wrote a number of scripts to comb through the Java code and create concordances and maps but these long lists of names did little to explicate the structure.

As a part of the seminar, the documentation group attempted to enumerate the ways a person goes about “discovering” a software project. Lacking specifics about how to do this, the students attempted to study their own thought processes and document the various ways they used to unravel mysterious code. Not surprisingly, this turned out to be quite difficult, as most self-introspection usually is, and it seemed to interfere with the more immediate task of producing the desired documentation for Robotran. As a result, this part of seminar saw little progress. In hindsight, the goal of learning how to “discover” a software system is so large that it should be its own seminar topic.

PASSING THE TORCH TO A NEW STUDENT PROGRAMMER

David’s last year with the Robotran project was consumed with writing a paper on his approach for an international conference, explaining his clever method of simulating the RCX without starting from the ground up. In the fall of 2008, Kevin Mastropaolo came on board the project and picked up where David left off. Without first absorbing the entire structure of Robotran, Kevin dived right in and began fixing bugs and adding features. His remarkable ability to divine precisely where to find bugs in a Java program that was 11,000 lines long and his success in porting Robotran to the NXT platform has led to this paper.

Kevin describes how his early programming experiences helped him tackle the Robotran system. “In seventh grade, I began to learn how to program on the graphing calculator we used for math class. Being curious as to how the games on the calculator worked, I went into the code and began to change parts to see what happened, not having any idea if it would break the program irreversibly or cause something interesting to happen. I then ran the program, saw what changed, and continued until I understood the code. TI’s BASIC language was more difficult than any modern programming language since the possible variables were limited to two characters, precluding descriptive variable names. Fast forwarding to the Robotran project, the skills I learned hacking around in middle school turned out to be useful in a new context.

The first step of fixing bugs was to see what Robotran did and how it did it. Therefore, the initial task was simply to run the program, see what it did, and compare features and procedures to the class names. After having a general idea of the program's
abilities and limitations, I found it easier to locate the code controlling the various parts. The search feature of Eclipse became indispensable to be able to make any significant changes. Frequently, trying to find names related to errors led to the frustration of having to dig through several layers of object-oriented code, where seemingly everything had a superclass and was calling some other class. But this was made much easier by the “open call hierarchy” and “open declaration” context features of Eclipse, which would bring me to the place from which the item I was looking at was drawing code. Unfortunately, after applying some fixes even more things broke, requiring even more tracing to see what I broke or a complete rethink to attempt to find another section of the code relating to the problem. After several weeks of this, I began to develop a subconscious mental map of what affected what, which classes controlled other classes, and which classes controlled the largest sections of the program. Difficulties occurred even then because there were repeated class names, and the attempted generalization to include NXT code meant there were objects for both the robot and its RCX subclass. The virtual ROM also caused problems while reviewing the code, and so the concept was scrapped when building the NXT version. Adding features was another beast entirely. Many times new additions required tracing through code to determine the hierarchy of controls. Expansion of the program also seemed to be far more likely to introduce other errors. Several of the program’s objects for the robot’s world were in arrays, causing many expansions to the code to throw index out of bounds errors.

Coming into a large project with incomplete documentation, and where the principal programmer was for all intents and purposes hit by the proverbial bus was easier than expected for me because of the perspective I brought into the project. While some people feel the need to know how everything works, especially in a project they have inherited, I was perfectly willing to allow large sections of the code to remain black boxes to me. By not worrying about every section of the code, I was able to focus on the parts that had to do with my current task and trust that the rest of the program would continue to do its job. After several iterations of this, a plot began to form in my mind of the entire program, much the same as if I had spent several weeks just looking through the code and documentation to learn the program. I was ahead of that curve, though, because not only had sections been completed during those weeks as I learned the program, but I had learned better how the program interacted by actually writing and changing the code within it.”

SOFTWARE ENGINEERING LESSONS LEARNED

Through the failures of the seminar group and Kevin's success, we gleaned some insights about legacy student software projects, insights that may be transferrable to more typical classroom settings.

There are two classes of student developers: creators and post-developers. A creator is the original designer, coder, tester and documenter, all rolled into one. Post-developers are the students who follow and who are often asked to fix the original program and extend it.

Recommendations for the creators start with simplicity. They should not overdesign the project or make it unduly complicated, flexible and extensible because that comes with an intellectual cost that will probably overburden any post-developer. Creators
should also provide some kind of high level documentation, even if only in the form of an audio or video recording explaining what the program does, how it does it, what the underlying code structure is, and what problem areas may become tar pits. Software engineering teachers might consider providing these basic skills. There is a fine line between encouraging quickly produced and shoddy documentation versus polished designs that are carefully thought out and well documented. By always emphasizing traditional, large-system software designs, software engineering teachers give students huge and cumbersome tools that they will never use for smaller designs, which are probably the only kind they’ll see in school.

Naturally, creators should try to use hygienic software practices like sticking with accepted object oriented principles and using reasonably descriptive identifiers. Even one line of explanation next to a variable can be a godsend later. Writing javadoc documentation is useful, but if the time cannot be found for this, then a minimal explanation of what a method is supposed to do will serve as the basis for a future cleanup of the code. Until IDEs encourage writing method headers and comments first, and de-emphasize actual code, this will always be a struggle.

Post-developers face an entirely different set of intellectual challenges, starting with the most basic of discovering the underlying skeleton of the system. Though a post-developer may have some idea of what the system is supposed to do and what it looks like when it runs, actually poking through the source code opens up an entirely different world whose connection to the user experience is very indirect. Because they didn’t design and code the original system, post-developers do not have that all-important mental map that creators have. Even the mental maps of creators will deteriorate rather quickly without continued involvement with the software. This is where the creators can help enormously with taped interviews or audio explications, by helping post-developers quickly gain the outline of a mental map.

By and large, post-developers will have to rely on discovering the structure themselves. Initially we hoped that technology would help, such as concordances of variable names, UML diagram generators and other tools. However, those tools are not able to help with the more general mental map. Introspection skills, which can be taught to some degree, can help post-developers develop this mental map by encouraging them to explore the tangled jungle of code and not get overwhelmed by it. They can document their progress, refine an emerging mental map by drawing diagrams or making notes or by writing about the structure as they uncover it. The mental map in the mind of a programmer is a kind of hypothesis that is refined as new details come to light.

“Programming by searching” describes how we zoomed in on the code details of Robotran. Eclipse allows on-the-fly concordance generation by hovering over a method name and it helped Kevin enormously in his hunting for relevant areas of the code. Sometimes he only knew a part of a method name, or suspected that a particular identifier existed. A search of the code turned it up somewhere near where he needed to go.

Summarizing what seemed to work, we note that a post-developer is ideally someone who is adventurous and curious, but not overwhelmed by a lack of structure at the outset. He or she needs to be persistent and willing to know only a little bit at a time and to be able to search intelligently. Being vocal and willing to write about what is going on their head is a great plus if the project is to continue. Perhaps we should seek
those who keep diaries or journals. In short, a post-developer should be someone who can build his or her own mental map and be willing to share theirs with future programmers.

Reflecting on the failures of the seminar group, we note that the students were very disparate in abilities and motivation, but their non-involvement in programming and subsequent lack of direction led to aimless wandering through the forest of code. The students who were assigned the documentation task did gain some understanding of the project, but they did not communicate much with the group who were trying to extend the program to enable collision detection. This was very different from Kevin’s work, where his understanding of the program was entirely motivated by his desire to fix and extend it. In summary, documentation will never be very good if there is no good motivation to produce it. Everyone harps on the fact that documentation should accompany every phase of the project from inception to maintenance, but the reality of student work requires that we take a new, creative approach to understanding and thence maintaining large student programs.

CONCLUSION

Since most computer science majors will face the horror of being assigned to a legacy system at least once in their career, computer science education should prepare them in a more scientific fashion by perhaps incorporating code discovery techniques in a software engineering course or requiring that a student take an internship or work on a project that is fairly large and complex. This experience should not be a “sink or swim” experience where the student is nearly drowned in a sea of bad code but rather an opportunity to utilize new techniques. Students who are inducted into this level of software boot camp may come away with a greater appreciation of the need for traditional, methodical software engineering techniques, as well as skills for surviving in the messy real world of legacy code.

REFERENCES


COMPUTER SCIENCE MEETS INDUSTRIAL ROBOTICS: A VISUAL SERVOING PROJECT FOR A COMPUTER VISION COURSE

Sebastian van Delden
Division of Math and Computer Science
University of South Carolina Upstate
Spartanburg, SC 29303
864 503-5292
svandelden@uscupstate.edu

ABSTRACT
This paper describes a visual servoing project that has been incorporated into a traditional upper-level computer vision course in a typical ABET accredited undergraduate computer science curriculum. Traditional computer vision concepts are reinforced in a hands-on industrial robotics setting where students implement a closed-loop vision algorithm using an industrial robotic arm. As industrial robotic arms and other technology become more readily available to academia, it should be investigated as to how this equipment can be integrated into a traditional computer science curriculum in order to: improve the quality of instruction; generate more interest in both fields of computer science and industrial robotics; and train a highly skilled manufacturing workforce.

INTRODUCTION
Manufacturing environments continue to evolve into sophisticated marriages of robotic equipment, computer technology, and humans. It is clear that the inability to adapt either equipment or human assets will result in the loss of American jobs to international companies. “Automate or Evaporate” is the saying in the manufacturing world. Industrial robotic systems improve productivity by increasing throughput and enhancing the quality
of manufactured goods. It is amazing that robotic automation has existed since the early 1960s - long before the age of personal computers, the internet and email.

North America has historically lagged behind many countries in the area of industrial robotics and manufacturing. A recent 2009 report by the International Federation of Robotics [1] compares the number of robots that are being employed by Asia, Europe, and North America. Figure 1 shows how North America has been and is estimated to be behind in this area in the coming years. Although North America lags behind other countries in this area, the integration of robotic technology in North America has historically grown steadily. Even though growth has recently stagnated, the forecast is still expected to continue to trend upward starting in 2010-2012 once the worst of the current economic crisis has abated [1].

On the computer science front, it is no secret that the number of students declaring computer science to be a major has decreased steadily since the 2000 bubble. Figure 2 shows the historical trends in computer science majors [2]. It seems as though the decline may perhaps finally be leveling out as it did in the early 1990s after the 1987 bubble.

Manufacturers say that they face a serious shortage of highly skilled workers who can fix and program robots and other equipment in a 21st century factory [3]. The integration of industrial robotics and computer science, and potential job opportunities that come with this marriage, will help to attract more students into the area of computer science and technology. Both industrial robotics and computer science areas of academia and industry will benefit from this marriage. The work presented in this paper provides a concrete example of how industrial robotics equipment can be incorporated into a computer vision course in a computer science curriculum.

![Figure 1. Recent and future lag of North America behind Asia and Europe in the number of industrial robots being used. Source: The IFR Statistical Department.](image-url)
RELATED WORK

Recent years have seen industrial robotic equipment becoming more available to the world of academia primarily because of two reasons: (1) many U.S. companies are donating their used equipment as they upgrade to current state-of-the-art machines, and (2) many robot arm makers now have special educational packages that make such equipment more attainable. Our robotics laboratory features a total of seven robotic arms, most of which were donated to us by the Stäubli Corporation.

Unlike mobile robots which are popular in academia [4-9], industrial robotic arms have traditionally been prohibitively expensive and thus there has been little work done on how to incorporate industrial robotics equipment into a computer science curriculum. In previous work, we have shown how such equipment can be incorporated into a traditional artificial intelligence course [10]. In that work, we presented a course long project that incorporated several traditional artificial intelligence concepts into a game playing robotic arm system. This paper is similar to that work as here we present an interesting industrial robotics project that encompasses several traditional computer vision concepts. A series of published articles which lay a framework of how industrial robotics equipment can be incorporated into several areas of computer science will aid computer science faculty as they pursue robotics equipment.

PROJECT OVERVIEW

A key point of this work is to integrate industrial robotic equipment into existing computer science courses, not to introduce new robotics courses into a curriculum. The
intent is that hands-on applications of traditional computer science topics using high-tech industrial robotic equipment we reinforce the material and increase interest in the computing field. The following project touches on several traditional topics found in popular computer vision texts [11-13]. The whole project concept is explained very easily:

*A round, black checkers piece is lying on a white background on which a black triangle of similar size is painted. Using a robotic arm with a camera and “pointy tipped tool” mounted to the end, push the checkers piece so that it comes to rest on top of the triangle.*

The initial concept of this project is based on a more complex project that we observed during a conference presentation [14]. The project is easy to setup and is shown in Figure 3. An interesting aspect of this project is that its implementation can be done in several ways so that it could either be a small 2-3 week programming assignment or a semester long project. In the following two sections, we outline the basic robotics material (transformations and Euler angles) that must first be covered followed by several of the computer vision aspects of the project which can include: thresholding, segmentation, structuring elements, erosion, Canny edge detection, active contour tracking (Snakes), smoothing using Gaussian masks, and Hough transforms.

**TRANSFORMATIONS AND EULER ANGLES**

Some basic industrial robotics will need to be covered initially so that students know the basics of how to operate and program the machines. On Stäubli robot arms featuring the V+ language, this takes a couple of 75 minute class periods. More time, however, must also be spent on 3D transformations, basic trigonometry and some matrix algebra in order for a student to understand the basics of how robotic coordinate systems work. This actually works out particularly well here as these areas are often already part of a computer vision course.

Figure 4 shows a portion of the project setup onto which several 3D coordinate systems (or frames) have been overlaid. Frames can represent coordinate systems or movements from one system to another. A frame consists of rotation and translation information which are stored in a 4x4 matrix called a transformation. The locations of the tool frame \(\{T\}\) and triangle frame \(\{G\}\) are known with respect to the robot’s world frame, i.e. the robot knows how to get to these locations. However, the checkers piece
frame \{P\} and camera frame \{C\} are not known to the robot. The transformation \{X\} that determines how the tool \{T\} gets to the checkers piece frame \{P\} must be calibrated – which is shown later in the next section. Locations in one system can be mapped to another system by multiplying the transformation with the location:

\[
^A P = \begin{bmatrix}^A T \end{bmatrix} {^B P} \tag{1}
\]

where \(^A P\) is a point in coordinate system A, \(^B P\) is a point in coordinate system B and \(^A B T\) is the transformation matrix that maps points in system B to system A. The 3x1 column vector \(^A P_{BORG}\) is the translational difference from A’s origin to B’s origin, and \(^A B R\) is a 3x3 rotation matrix comprised of three orthonormal column vectors that map the orientation of each axis in A with respect to B. In the below matrix, \(\alpha\) is the rotation around the moving Z in degrees, \(\beta\) is the rotation around the moving Y in degrees, and \(\gamma\) is a second rotation around the moving Z in degrees – i.e. a Z-Y-Z Euler angle configuration which is the internal representation used on Stäubli V+ machines.

\[
^A B R = \begin{bmatrix}
\cos \alpha \cos \beta \cos \gamma - \sin \alpha \sin \gamma & - \cos \alpha \cos \beta \sin \gamma - \sin \alpha \cos \gamma & \cos \alpha \sin \beta \\
\sin \alpha \cos \beta \cos \gamma + \cos \alpha \sin \gamma & - \sin \alpha \cos \beta \sin \gamma + \cos \alpha \cos \gamma & \sin \alpha \sin \beta \\
- \sin \beta \cos \gamma & \sin \beta \sin \gamma & \cos \beta
\end{bmatrix} \tag{2}
\]

**VISUAL SERVOING AND COMPUTER VISION TOPICS**

A visual servoing system is a closed-loop system in which images of a scene are repeatedly captured and compared to a target scene. Each time an intermediate image is compared to the target image an error vector is computed which indicates how to move the robot so that the error between the intermediate and target image is minimized. See [15] for a detailed tutorial on visual servoing.

In this project, the target image is when the checkers piece is on top of the triangle. When each intermediate image is snapped, the exact angle must be computed so that the piece is pushed towards the triangle. Because of the “pointy-ness” of the tool, the checkers piece will randomly slide off during the push movement either to the left or right.
side of the tool’s tip. During each iteration, the push angle must be recomputed. The compete algorithm along with reference to computer vision techniques is as follows:

1. Automatically determine a threshold which can separate background pixels from object pixels. This can be done automatically because of the fact that there are only two objects on a white background. Every possible pixel threshold (0-255 for a typical grey scale image) is considered and the number of objects in the resulting binary image is counted using a “hole counting” algorithm [11]. The average of all the thresholds which result in two objects in the image is the final threshold.  

   **Computer vision topics covered:** working with images, binary images, thresholding, and hole counting.

2. Determine which object is the triangle and which is the checkers piece. This can be accomplished by performing “erosion” on the image with a “disk structuring element” [11]. This will not only identify the checkers piece but also produce its pixel centroid in the image. This is the simplest way to differentiate the objects in the image, however, this portion of the project could be extended by implementing more complex algorithms. For example, the pixel centroid of the circular piece could be determined by first finding the edges in the image using the “Canny” edge detection algorithm [11] followed by the circle-finding version of the “Hough transform” algorithm [11]. A pre-cursor to these steps would be smoothing using a Gaussian filter [11] to minimize noise in the image. These algorithms are typically always covered in a computer vision class and would extend the complexity of the project considerably. Although more complex, the students would learn the benefits of these techniques. For example, these algorithms solve the problem even if the size of the triangle or checkers piece changes during execution, or other objects (noise) is present in the image.  

   **Computer vision topics covered:** morphology, erosion, structuring elements, and possibly also: smoothing, Gaussian masks, Canny edge detection and Hough transforms.

3. Center the checkers piece in the image. The relationship between the X and Y axes of the camera frame \( \{C\} \) and tool frame \( \{T\} \) must be calibrated at this point. This calibration can be manually accomplished simply by looking at the camera and tool to determine which axes in the two systems are most closely aligned. Centering can then be accomplished by iteratively moving the tool along its appropriate X and Y directions until the pixel centroid of the piece is in the center of the image.  

   **Computer vision topics covered:** visual servoing, and simple camera calibration.

4. Once centered, the transformation \( \{X\} \) that relates this image to the actual piece must be calibrated – really only the translation information is important. This can be done by manually moving the tool to the piece and noting the initial and final frames of the tool – which is easily accomplished on Stäubli machines. From these locations the translation is computed by simply subtracting the X, Y and Z components of the locations.  

   **Computer vision topics covered:** matrices and transformations.

5. Since the centroids of both the triangle and circle have been determined, as well as the transformation \( \{X\} \) that relates \( \{T\} \) to \( \{P\} \), it can now be computed where the tool must go to initiate a push of the piece towards the triangle. This can be accomplished by determining the standard straight line formula \( y = Ax + b \) using
the two known points. Note that vertical lines, of course, pose a problem for this equation and need to be handled. Depending on the resulting slope of the line, an “appropriate” new $x$ can be plugged into the line equation to get its corresponding $y$. The new $x$ is simply the checkers piece $x \pm$ some predetermined number of millimeters. The sign depends on the sign of the slope of the line. The resulting new $(x,y)$ location is away from the center of the piece and oriented exactly on the side opposite to the triangle. The pointy tool should go to the newly computed $(x,y)$ location and then move straight to the triangle frame $\{G\}$. See Figure 5 for an illustration of this step. This process could be enhanced by implementing a deformable contour following algorithm like the Greedy Snakes algorithm [11]. The advantage of this algorithm is that the shape of the checkers piece or triangle can still be tracked even if their shape changes during execution.

Computer vision topics covered: possibly deformable contour tracking, greedy snakes.

- Steps 2-5 are repeated until only one object is present in the image. The checker piece on top of the triangle will image as one object in the binary image.
- Note: since everything occurs on a flat table surface, depth can be hardcoded which simplifies this project tremendously.

CONCLUSIONS

Industrial robotics equipment will become more commonplace in academia as American companies upgrade to new machinery and robot makers continue to offer special educational incentives. This equipment could play a key role in teaching several traditional computer science courses. In this paper, a detailed project for a computer vision course has been outlined. This project was introduced into the fall 2009 computer vision course at USC Upstate and was well received by the students who were eager to spend numerous hours outside of class working on it. Two videos are available online in which student groups demonstrate their implementation of this work.

Please visit http://faculty.uscupstate.edu/svandelden and click on “Robotics Videos”.

ACKNOWLEDGEMENTS

We would like to sincerely thank the Stäubli Corporation for their generous donation of robotic equipment as well as their continued support of research at our institution.
REFERENCES


ARCHITECTURE OF A JAVA FRAMEWORK FOR
DEVELOPING GENETIC ALGORITHMS IN AI CLASS

T M Rao and Sandeep Mitra
Department of Computer Science
The College at Brockport, State University of New York
Brockport, NY 14420
585-395-5176
{trao, smitra}@brockport.edu

ABSTRACT

In this paper we present the architecture of a flexible, object-oriented Java solution framework for implementing genetic algorithms (GA) solutions to NP-hard problems. The framework realizes problem-independent features of any GA solution. Its flexibility lies in the fact that it can be easily configured with components specific to a particular solution. We discuss the classroom usage of our framework and also present how instructors can use this framework to vary the level of difficulty of a GA programming project, depending on the desired learning outcomes. Finally, we present a comparison of our framework with others that are available on the Internet.

1. INTRODUCTION

The Artificial Intelligence (AI) course at The College at Brockport covers traditional AI topics such as problem solving using search, game playing, expert systems, evolutionary computing, etc. Students enter this class with a good background in data structures (stack, list, etc.), algorithms and considerable programming experience in Java. It is an extensive programming course with students expected to complete three or four major programming assignments. Each programming assignment is devoted to a specific AI problem-solving technique. Students are required to implement the technique, solve one or more problems using it and experiment with these solutions.
The AI instructor faces several challenges when designing the programming assignments. Besides the traditional dilemma as to whether to require programming in the traditional AI languages (e.g., Prolog), the instructor has to decide whether to require the students to implement the programs from ‘scratch’ or to provide a supporting framework that may be built upon in constructing the solution.

In recent years, the first author, who has taught the AI course for a number of years, has leaned towards using Java, rather than Prolog, for at least some of the programming assignments. This was done to utilize the programming expertise in OOP with Java that the students already have. Using Java made it feasible to assign programming projects of greater complexity (Prolog is still used for some projects – e.g. expert systems). Furthermore, we realized that several of the AI programming projects were quite complex, and requiring the students to program them from ‘scratch’ would mean having them complete three or four major software development projects in a one-semester time frame. A student carrying a full-time course load simply would not have the time to complete these projects in a manner that ensured that the learning objectives were achieved. Thus, the choice we faced was to either cut the number of projects, or provide supporting frameworks and require multiple projects. As it is advantageous to have students experience programming multiple techniques, we opted for the latter approach.

We note that in an educational setting, providing a framework that has several features of the problem already implemented may make the project ‘too easy’ and thus compromise the learning objectives of the course. This concern can be handled by using a framework with an architecture that is flexible enough to enable the instructor to easily override the default implementations of certain features with student-written code. Similarly, it should also be possible for the instructor to provide a partial implementation of a feature the framework requires to be fully coded by the user. Students would then simply complete these partial implementations. It should be possible to configure the ‘core’ framework to use these alternative implementations easily. The architecture of such a flexible framework is the main subject of this paper.

AI problems can be categorized into families based on the techniques used to solve them – e.g., puzzle solvers typically use the state space search (SSS) technique and belong to the SSS family. We have developed flexible framework architectures for three such families – SSS, Two-Person Zero-Sum Games (TPZS) [4] and Genetic Algorithms (GA). The development of the GA flexible framework, and its use in class, is discussed in this paper.

2. ARCHITECTURE OF THE GA FRAMEWORK

Genetic Algorithms [2, 3] (GA) are used to compute good solutions to NP-Hard problems. It is well known that all available algorithms to solve such problems have exponential time complexities. Evolutionary techniques such as GA are used to compute satisfactory solutions to such problems in a reasonable amount of time, especially for large input data sizes.

In a GA, candidate solutions to a problem are modeled as individuals in a population. Depending on the problem, the population is initialized either randomly or strategically to a certain size. Each individual has a fitness value which represents its
quality as a solution. We seek to find the individual with the best fitness. The population goes through a pre-determined number of cycles of evolution. In each cycle, a series of genetic operators are applied to modify the population. The standard operators are evaluate, select, crossover and mutate. Other operators to prevent the population from dying out may be added. The problem-solver keeps track of the all-time-best performing individual till the end of the computation. Thus, the main protocol of a GA consists of:

1. Initialize: Initialize the Population randomly or in some problem-specific way.
2. Loop N times (where N is a predefined constant)
   a. Evaluate: Compute the fitness of each individual in the current population. Update the all-time-best individual.
   b. Select: Select a subset of the current population for crossover, using the selection strategy.
   c. Crossover: Select pairs of individuals using the crossover candidate selection strategy and perform crossover to produce a set of new individuals. The likelihood of the two selected individuals’ crossing over is determined by a predefined crossover probability.
   d. Mutation: Mutate (slightly perturb) a small number of individuals. The mutation happens with a predefined probability (mutation probability).

Our GA framework’s architecture is shown in Figure 1. At the highest level of abstraction, it provides several Java interfaces. These are as follows:

- IIndividual requires methods computeFitness() and mutate().
- IFitnessFunction requires fitness(IIndividual i).
- IPopulation requires methods initialize(), evaluate(), select(), crossover() and mutate().
InitializationStrategy requires method `getInitialPopulation()` that returns an IPopulation.

SelectionStrategy requires method `applySelection(IPopulation p)` that returns a list of selected individuals.

CrossoverCandidateSelectionStrategy requires `selectForCrossover(IPopulation p)` that returns an array of pairs of individuals selected for crossover.

CrossoverTechnique that requires `applyCrossover(Pair p)` that returns another Pair containing the result of crossing the pair in parameter ‘p’.

MutationCandidateSelectionStrategy that requires method `selectForMutation(IPopulation p)` that returns a list of candidates selected for mutation.

MutationTechnique that requires method `applyMutation(IIndividual i)` that is expected to mutate the individual ‘i’, altering its contents.

The GA framework provides default realizations for many of the genetic operators by providing implementations for several of the strategy interfaces above. The only strategy interfaces that must be implemented by the programmer are InitializationStrategy, CrossoverTechnique and MutationTechnique, as these are problem-dependent. The default implementations are generally straightforward approaches using user-supplied numerical values. For example, the default implementation of SelectionStrategy uses an elitist selection policy, where it simply selects the top N% of individuals. The framework reads the value of N from a configuration file editable by the user. Other parameters such as the population size, crossover and mutation probabilities, etc. are also specified in the same configuration file. For example, the default implementation for selecting pairs of individuals for crossover chooses the members of the pair randomly, and creates P pairs, with the user supplying the value of P in the configuration file. Also, the default implementation of the strategy for selecting a candidate for mutation chooses randomly, based on the user-supplied mutation probability value MP provided in this same file. It is important to note that the names of the default implementation classes – e.g., DefaultSelectionStrategy (see Figure 1) – are also provided to the framework through this configuration file. The framework uses the Java Reflection API to create the desired instances of these classes.

In addition to these defaults, the framework provides an adapter class for the IIndividual interface, called AbstractIndividual. The only functionality this class provides is that it can be configured with the user-implemented fitness function object. The programmer would find it convenient to make the problem-specific individual inherit from this class. The Population class in the framework encapsulates a list of individuals. It is also the main ‘driver’ class, containing the ‘solve()’ method, which implements the main GA protocol. The programmer simply creates an instance of this class, and calls the ‘solve()’ method to launch the GA. Note that it is the Population object that reads the configuration file and creates the instances of the classes specified in it. Setting up the data in the configuration file properly is an important programmer task. The programmer’s obligations in using the GA framework require the writing of the following classes: the problem-specific individual class, the fitness function class, classes for the initialization strategy and the crossover and mutation techniques. Optionally, classes may be written to override the framework defaults. Thereafter, the programmer must ensure
that both the desired class names and configuration data values are properly set up in the configuration data file.

3. CLASSROOM USE OF THE GA FRAMEWORK

The GA framework was used in class during Spring and Fall 2009. Students were introduced to the framework in the classroom through a fully solved example – a relatively simple one involving a mathematical function \( f(x) = \sin(\pi x/256) \), \( 0 \leq x < 256 \). Students were shown how the individual candidate solutions for this example could be coded as 8-bit patterns, and the fitness function is \( f(x) \) itself. A simple crossover technique was shown, and all framework-provided defaults were used.

The second problem discussed was a special case of the Traveling Sellers Problem (TSP), with all cities located along the perimeter of the unit circle and the graph fully connected. Cities were labeled from ‘0’ to ‘\( N-1 \)’, with ‘0’ being the start city. This example was used because the optimal solution is known, and the result obtained from the GA could be checked to see how close it was to the optimal. Students were shown how the individual in this case could just be a string of the form “0 permutation of numbers from ‘1’ to ‘\( N-1 \)’ 0”, with each such string being a valid candidate solution thanks to the fully connected graph. They were also shown how to write a crossover technique that would ensure the generation of a pair of new strings that were both valid tours. Finally, they were asked to experiment by changing the numerical values in the configuration file such as population size, number of iterations, various probabilities associated with the default strategy implementations, etc. to see if the quality of the solution could be improved. The solutions obtained were displayed in graphical form, and obtaining the solutions for 16 cities, then 32 cities, etc. created excitement in the classroom.

The students were asked to apply the GA to one of two possible NP-Hard problems, the Assignment Problem and the Knapsack Problem [1]. The students were asked to design a representation for the individual, write the fitness function and program the strategies for which the GA framework provided no defaults. They were asked to experiment with problem sizes ranging from 8 to 32, iterations from 50 to 5000, and population sizes from 5 to 20. The students had no difficulty in understanding the framework and in fact were very enthusiastic in using it. Two-thirds of the students were able to complete the project in time and received full credit for it. This contrasts to our experience three years ago when only one out of 9 students completed the assignment from scratch. In classroom surveys done since 2007 for this and other frameworks, students clearly indicated a preference to use the frameworks. We think that the results are consistent with our expectation that our approach enables students to be more productive in the limited time available for these projects. One of the comments from a successful student: “I like the fact that all I have to do was extend the needed abstract classes, or implement the needed interfaces, and then all the methods are set in stone. There was no guess work beyond that of my own imagination. I found that the GA framework provided made things easy to implement. I would really have liked to try more algorithms with this framework.”

The architecture of our GA framework is highly flexible. An instructor who wants to lower the complexity of the assignment might ask the students to use some or all of the
framework-defined default strategies. On the other hand, requiring the students to override the defaults increases the complexity of the assignment. As explained above, this overriding can be done by changing the names of the classes in the configuration file. Invasive code changes, and a full re-build of the system, are not necessary.

4. COMPARISON WITH OTHER FRAMEWORKS

There are several Java frameworks for GA available (e.g. JGAP [2], JCLEC [6] and the framework provided as a companion to the Russel-Norvig (RN) book [5]). We compared our framework with these for appropriateness in achieving the learning objectives of our GA instruction. The main learning objective is that the students acquire the ability to formulate a GA solution to a new problem. This requires that they be able to design a suitable representation for the individual, a fitness function, write algorithms to initialize a population with these individuals, and develop the crossover and mutation techniques. While the RN framework is a companion to a popular AI textbook, JGAP and JCLEC are large collections of packages supporting genetic algorithms and many other evolutionary techniques. These are geared towards a much wider audience that includes graduate students and GA researchers.

JGAP provides basic classes/interfaces to implement a Gene, Chromosome (i.e., Individual), Genetic Operator, etc. It enables programmers to define their own operators, and the order in which they should be applied. These preferences are encoded in a class called Configuration. Alternatively, the programmer can choose to use all the default implementations of the operators, and just define the Individual and Fitness Function. Similarly, JCLEC also provides a large number of interfaces and abstract classes. It provides default implementations for selection, crossover (recombination), mutation, etc. In addition, it provides default representations for the individual using integer arrays, real arrays, etc. In comparison, our GA framework (much smaller in size and only supporting genetic algorithms) requires the programmer to at least code the initialization strategy and the crossover and mutation techniques. We believe that requiring these to be written is essential to gaining a full understanding of a GA – there are no defaults provided for these. Also, as stated above, our framework enables modifying the implementations of the operators by simply writing new class names in a configuration file. There is no need to change code in a Configuration class, and re-compile it. This feature has proved especially useful in classroom demonstrations.

The RN framework requires the programmer to encode the individual as a String object, and define a fitness function. It enables the configuration of numeric data such as the population size, number of iterations, and the mutation probability. It does not allow the programmer any mechanism to define her own genetic operators – the defaults provided by the framework must be used. Encoding every individual as a String can prove to be problematic – especially for students. Also, the lack of ability to define your own versions of the operators is clearly a drawback and can have a serious impact on the number of iterations to produce good quality solutions.
5. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented a flexible GA framework that allows the instructor to provide students with appropriate software support to achieve desired learning objectives. Students can focus on problem-specific data representations and algorithms that enhance their understanding of the problem being solved. Our classroom experience indicates that students who put in a few hours of effort in understanding the architecture and API of the framework had no problems in using it to complete the homework. In the future, we plan to develop similar flexible frameworks for other AI techniques like neural networks and particle swarm optimization.

REFERENCES

EXPANDING STUDENT ENTHUSIASM FOR, AND
UNDERSTANDING OF, INTRODUCTORY CS

PANEL DISCUSSION

Bridget Baird, Professor of Mathematics/Computer Science, Connecticut College
Moderator and Contact Person
bbbai@conncoll.edu
Box 5412
Connecticut College
New London, CT 06320

Panelists:
Martin Allen, Mellon Postdoctoral Fellow in Computer Science, CTW Consortium
Christine Chung, Assistant Professor of Computer Science, Connecticut College
Norman Danner, Associate Professor of Computer Science, Wesleyan University
Madalene Spezialetti, Associate Professor of Computer Science, Trinity College

Professors in computer science are challenged by the wish to make CS1 courses more appealing to a broader audience, both to induce more students to enter computer science and also to open up students to the possibilities of the field. This challenge is especially acute when it comes to underrepresented groups. The panelists come from three liberal arts colleges that have formed a consortium in computer science to examine topics of mutual interest. Among other activities, the three departments have shared a number of postdoctoral fellows, expanding the range of approaches brought to their research and teaching. Further, they have collaborated on a number of initiatives to clarify what a liberal arts degree in computer science should look like, and to deepen and diversify their curricula, particularly at the introductory CS1 level. While the panelists all come from liberal arts environments, they have different approaches to the CS1 course. During the course of this panel they will discuss their views and experiences on the following list of topics:

- What languages and lab structure hinder or help students to understand the concepts of CS?
- What techniques hinder/help women and other underrepresented groups to both understand and enjoy the CS1 course?

* Copyright is held by the author/owner.
It is often the case that there is a wide range of prior experience among students in CS1. How do we challenge and engage these students who have more technical experience without discouraging other students?

What are the markers of success in a CS1 course: taking another course, enjoying the course, future involvement in some capacity with CS, using the knowledge in another course, not being “turned off”?

What is the role of current technology (if any) in a CS1 course (for example, smart-phones, web apps, mashups, social networking)?

Each of the panelists will comment on each of these topics; audience interaction on these topics will also be sought during the course of the panel.

The panelists bring a rich array of experiences to the panel. Martin Allen is a postdoctoral fellow, teaching at all three colleges in the CTW consortium as part of the Mellon grant program, which intends to bring new doctorates from research-school backgrounds into the liberal arts college environment. Given his recent experience at a relatively wide range of different and distinct institutions, he will discuss the varying issues that arise, and approaches one can take, when teaching CS to students with a diverse range of experience, interest, and preparation. Christine Chung recently received her doctorate but brings a wide range of expertise and experience. She has taught introductory CS at both the college and high school levels and she earned a master’s degree in secondary mathematics education. Norman Danner has taught introductory CS at UCLA and Wesleyan, from 200-student lectures to 15-student classes, from no objects to objects-only, and from programming-intensive to more writing than programming. Madalene Spezialetti’s research focuses on the development and use of video and animation in Computer Science education. She received a SIGCSE Special Projects Grant to support her work in the production of video scenarios, which are short films designed to develop computational thinking skills by presenting computing-based problems in the context of extemporaneous, real-life situations.
UNDERSTANDING NSF FUNDING OPPORTUNITIES

TUTORIAL PRESENTATION

Scott Grissom
National Science Foundation
4201 Wilson Blvd, Suite 835
Arlington, VA 22230
(703) 292-4643
sgrissom@nsf.gov

ABSTRACT
This session highlights programs in the National Science Foundation (NSF) of particular interest to computer science educators. Topics include a description of program goals, guidelines, review process as well as strategies for writing competitive proposals.

INTRODUCTION
NSF supports projects to improve education in science, technology, engineering, and mathematics through several programs in its Education and Human Resources (EHR) directorate, as well as in its research directorates, including Computer and Information Science and Engineering (CISE). This tutorial presents a description of some education-related programs in the EHR and CISE directorates, and enables participants to interact with the presenters concerning specific project ideas that could be appropriate for the various programs.

SPECIFIC PROGRAMS DISCUSSED
Complete details about each of the following programs can be found on the NSF websites for the Division of Undergraduate Education (DUE) [1] and the Directorate for Computer & Information Science & Engineering (CISE) [2].

- Course, Curriculum, and Laboratory Improvement (CCLI)
- CISE Pathways to Revitalized Undergraduate Computing Education (CPATH)
- Federal Cyber Service: Scholarships for Service (SFS)
- Research Experiences for Undergraduates Sites (REU Sites)

* Copyright is held by the author/owner.
WRITING COMPETITIVE PROPOSALS

NSF programs are quite competitive but there are simple strategies that investigators should be aware of to improve their chances of success. First and foremost, read the Program Solicitation carefully. The goal is to help reviewers quickly understand what you intend to do and that you have given sufficient thought of how you intend to do it. Organize the proposal to address the essential components described in the solicitation. Use headings, boldface and bulleted lists to help the reader quickly understand the organization of the proposals. Address each point thoroughly but succinctly. And finally, start well before the submission deadline. Include sufficient lead time to allow colleagues to provide feedback and for your research office to approve the proposal. Specific directions for completing a proposal can be found online [3].

REFERENCES


EXPERIENCES BUILDING A COLLEGE VIDEO GAME DESIGN COURSE

Paul E. Dickson
Hampshire College
School of Cognitive Science
893 West St.
Amherst, MA 01002, USA
(413) 559-5861
pdickson@hampshire.edu

ABSTRACT
This paper details some trials and tribulations of the design and implementation of a video game design course that not only enables students to create video games but also to learn standard computer science skills. The decisions that led to the platform decision of the iPhone/iTouch are discussed along with the consequences of that choice. The project-based and dynamic curriculum that encompasses real-world problem solving is described. Initial successes and failures of this course and the lessons learned are addressed as well as the level of student excitement surrounding it.

INTRODUCTION
In Spring 2009 we decided that a video game design course would be a good elective within our computer science program. As our program does not have a set major with a required number of courses and electives, our electives must have a broad appeal in order to attract enough students to make them feasible. Video game design was decided upon because it has a definite “cool factor” among students and because it would give our students the ability to show their peers what they have learned in a way not often possible in computer science.

It was decided that computer game design could be a valid addition to our curriculum if the course focused on design from the ground up and did not just use...
prewritten game development software. We envisioned that video game design could be used to teach students about real-time processing and development, resource management, real-world environments, and device limitations as well as give them good practice programming large chunks of code and working in teams. Huang [5] describes some of the advantages of game-related assignments in relation to student engagement, teamwork, development of algorithms, etc. All of our choices about the design of this course appear to fall within the framework of good use of computer games creation within a computer science curriculum as described by Sung [12]. Sung’s review was published near the end of the semester this course was taught.

RELATED WORK

Courses in computer game design generally are taught in schools that offer degrees in game design or as single elective courses offered within an established computer science or engineering major. While some traditional universities like DePaul University [3] offer a complete major in video game design, most schools that offer these courses tend toward career education, like ITT Technical Institute [6]. In either case the courses taught do not relate to the one we chose to offer because no single course covered the breadth of material we hoped to include.

Single courses in game design such as those taught Bruce Maxwell [10] and Kevin G. Stanley [11] were also discovered. These courses focused on game design from start to finish and helped lead to our course.

A new area where games are appearing in computer science curricula is in CS1/2 courses, where games are used to bring students into a major [12]. This type of integration is not applicable to our course, which is intended for more advanced students.

PLATFORM DECISIONS

The first decision to be made about his course was what platform to use for development and we considered Microsoft’s Xbox, a toy game development platform, Flash, OpenGL, and the iPhone/iTouch. Our decision was based on the best combination of feasibility, ease of entry, and students being able to show off their work.

We ruled out using a toy game development platform from the start since such platforms would likely never be used again by students after the end of the semester. We decided against Flash because while it is ubiquitous and would have made it easy for students to show off their final games, it does not have the cool factor that would really motivate students. In addition, every person who has played a Flash-based game has played at least one really bad Flash-based game, and therefore it would be less exciting because students would not have sufficient time to create a really good game. Creating games that relied on OpenGL on the computer was eliminated because students would have had to make sure that OpenGL was able to run on a given machine before they could show off a game. The largest factor in disregarding the computer-based games was that the platform did not lend itself to enabling students to show what they had created.

Our initial preference was to develop on the Xbox since Microsoft had previously given out Xboxes to universities that wanted to build courses around them. Also,
Microsoft made the development software available without charge and students would be able to show off their final games. Discussions with Stanley [11] and a look at the fine print disabused us of these ideas. First, Stanley described how only a small portion of the student projects in his game development class had actually been ported to the Xbox because of the large hassle involved with the device compilation, defeating the purpose of developing on the Xbox. Second, Microsoft does not allow you to burn discs of games, which means that downloading a game to a console requires a yearly paid subscription. These barriers to students showing their work, in addition to the fact that we are primarily an Apple campus and Microsoft’s game development suite only runs on Windows, dissuaded us from using the Xbox.

The platform we decided on was the iPhone/iTouch. Grissom [4] does a good job of describing the excitement and possibilities for using iPhones in courses. Apple has a university developer program [1] that makes it free for schools to enable students to download their created software to devices. In addition, many students own iPhones/iTouches, making it easy from them to show off what they have created for at least the time they are in school and under the school’s development license. Developing for a hand-held device also has the cool factor as students can show off what they have created anywhere they go and to anyone they meet. A final advantage is that it is a platform with limited input capabilities, which limits what students may attempt to create, a good thing in a course where students will only have limited time to create games.

**COURSE FORMAT**

The goal of this course is to teach computer science through the creation of video games. The semester is divided roughly into 3 parts. The first third of the course is devoted to students learning specifics of the platform. Students learn how to use Apple’s Xcode development environment [2] to build interfaces that include buttons, text fields, labels, multi-screen applications, and other aspects of the iPhone SDK. They are also introduced to Objective-C (the development language used by Apple) and the steps involved with downloading applications to devices. Each lecture takes place in a computer lab, and students work through examples as they are presented and then are given time to modify them in class before new material is introduced.

The second third of the course is devoted to an introduction to OpenGL ES, the graphics library that runs on the iPhone. The goal of this section is not to comprehensively teach students OpenGL ES but instead to teach them just enough to get started building games with the reasoning that anything specific that they need and are not been taught they can pick up on their own as needed. Therefore, basic point, line, triangle, coloring, translation, rotation, touch recognition, and collision detection are covered. Again, students follow along with material, typing it in while it is presented and playing with it during class. In many cases, work created during class is presented by students to the entire class to show off how the material can by used in ways well beyond that fathomed by the lecturer.

The final third of the course relates to their final project games. To begin this section each student presents an idea for a final project game to the class. Students then decide which game they want to work on and form teams (this semester, 5 teams for the 17 students in the class). The rest of the semester is spent working on these final projects.
Each week one lecture is devoted to going over some minutia of game design or drawing (e.g., texture integration into OpenGL) that may be of use to the design teams in general. The other lecture of the week is devoted to in-class work on the games. This lecture often begins with mini-lectures given by students about interesting points and solutions that they have discovered over the course of their work on the final project games.

Course References

Finding an appropriate set of reference materials for the course turned out to be one of the greatest difficulties associated with it. Few books exist about iPhone development, and the only iPhone game development books with code were published during the semester the class was given. The textbook settled on was Beginning iPhone 3 Development [8] because it gave good worked-through examples that showed off both how to use Xcode and how to write programs that utilize the iPhone SDK. Even the choice of this book illustrates one of the issues associated with iPhone and other cutting edge classroom development. It was discovered 3 weeks into the semester that students had a newer version of the text than did the professor as the professor’s copy was bought 3 months before and was hence out of date and had been updated. Quickly outdated source material proved a recurring theme throughout the course.

The best references found for OpenGL use on the iPhone were online tutorials [7, 9]. These tutorials both give good concrete examples of how to use OpenGL ES 1.0 on the iPhone and proved invaluable. The downside was that they were 4 months old when the class began and were out of date since Apple upgraded the iPhone to OpenGL ES 2.0. This would not have been a problem except that all tutorials were written using a template in Xcode that no longer exists. No written texts that have concrete examples of OpenGL on the iPhone have yet been found by the authors of this paper.

The iPhone is such a new device that its standards and code keep getting massive changes and upgrades and there is limited platform stability. Over the first 5 weeks of the semester, we had to install 3 different versions of Xcode because Apple kept updating the iPhone software and Xcode. The upgrades were required since any device that had been updated (done without thinking by many students to their own devices and by the instructor as well) could not have software installed on them without a software upgrade (free but over 5GB to download each time). These updates fortunately did not appear to include any updates to the templates used in Xcode but often changed the look of menus and made small changes to software functionality.

Assignments

The course includes 4 substantive assignments, one of which is the final project. The first assignment is intended as an introduction and to let students have some fun. They have just learned how to build a basic interface and are asked to use buttons, labels, and text fields to build an application that specifically tailors insults to the name that was typed into the application. Students are given the opportunity to use what they have learned but also to express themselves and have fun with the assignment.
For the second assignment students are asked to build an application that has the user enter a name and date of birth and then click a button to enable the application to supposedly search the web for images of that person. When the button is pressed the application returns an image of a monkey or something similar. This assignment familiarizes students with how to build multi-screen applications and flip between the screens, something important if they wish to have a settings page for their eventual games. Again, creativity was encouraged and many students made their applications with special provisions to show images of themselves when their own information was entered.

The third assignment is to build Tic-Tac-Toe or a similar game of their choosing. This assignment is intended to give them both a first opportunity to build an application that uses OpenGL to draw and to create basic opponent logic into an assignment. This was the first assignment where pair programming was allowed and encouraged. As with all previous assignments, students were given a week to complete it but unlike previous assignments students ran into a lot of trouble. The task of building the interface, graphics, and handling user touches proved the limit of what any students were able to accomplish. The hurdle of getting graphics up on their own for the first time when none had experience prior to the class with OpenGL or with handing user touches to a device proved so high that none had enough time to begin game logic. In future offerings of this course, this assignment will probably be broken up into two parts, the first related to interface design and the second to program logic.

The final assignment is the final project. For this the students work in groups on games the students have envisioned and chosen to work on. Students are given class time to work as described above and a lot of leeway to design and work on the games as they choose. Students have 5 weeks to take a game from an idea to a fully functional, if simple, game. This first semester the 17 students in the course broke up into 5 groups of 3 or 4 members each. The groups were formed based on interest and therefore some were stronger than others. By the end of the semester, 2 groups had succeeded in creating games, 2 groups had made significant progress, and 1 group barely made any progress. This latter group had only 3 students, 1 of whom stopped participating in the course and failed.

CONCLUSIONS

At the conclusion of the course, a class meeting was held to discuss the course in general and improvements that could be made. The instructor raised the issue of chaos caused by software updates, out-of-date references, and hardware problems. Student reactions were extremely positive: being able to show their work on an iTouch far outweighed these concerns. The disorganization of the course did not bother them primarily because they knew this was its first offering.

Working with the iPhone gave students a better look at more real-world programming problems. Software development in a classroom does not translate well to dealing with poorly documented technology. The students learned a lot from dealing with hardware and under-development documentation. They suggested decreasing the time spent on the introduction to the iPhone and increasing the time spent on the final project, which was the part of the course that taught them the most. Working in groups on their final project pushed them to learn more than did the individual smaller assignments.
earlier in the semester. They commented that a larger final project with more evaluation during the process would be an improvement. They also commented that because there was no available expert on iPhone development, they had to learn more on their own and thus now better understand the device and their code.

Though only 2 of the 5 final project games were completed, the project itself was a success. All 4 of the groups (14 of 17 students) that made real progress towards final games learned a lot from the experience. They were able to apply object-oriented design principles, data structures, archiving schemes, etc. from other classes to this project and learned a significant amount from doing it. All students had more confidence in their ability to program and handle large projects at the end of the semester. The only downside of the final project, from the student perspective, is that they did not have enough time. At least 2 of the groups have continued to work on their game after the semester ended.

Overall, this course was a success. The work students do in computer science often lacks a “wow” factor and therefore we sometimes have trouble motivating them to invest in their work. This course reverses that trend and as one student was overheard to say, “All my friends want a copy of this game once we get it finished”. If friends are this interested, students will go above and beyond to create a great game and will learn a lot more from the process.

REFERENCES

[9] Maurice, S., iPhone OpenGL ES Tutorial Series,
http://web.me.com/smaurice/AppleCoder/iPhone_OpenGL/Archive.html,
retrieved November 16, 2009.

http://www.cs.colby.edu/maxwell/courses/cs369/J09/, retrieved November 16,
2009.

[11] Stanley, K., College of Arts and Science, University of Saskatchewan,
http://artsandscience.usask.ca/college/directory/display.php?bioid=1184,
retrieved November 16, 2009.

[12] Sung, K., Computer games and traditional CS courses, Communications of the
MOVING CS50 INTO THE CLOUD

David J. Malan
Harvard University
School of Engineering and Applied Sciences
malan@post.harvard.edu

ABSTRACT
In Fall 2008, we moved Harvard College’s introductory computer science course, CS50, into the cloud. Rather than continue to rely on our own instructional computing infrastructure on campus, we created a load-balanced cluster of virtual machines (VMs) for our 330 students within Amazon Elastic Compute Cloud (EC2). Our goals were both technical and pedagogical. As computer scientists, we wanted more control over our course’s infrastructure (e.g., root access), so that we ourselves could install software at will and respond to students’ needs at any hour without an IT department between us and our systems. As teachers, we wanted easier access to our students’ work (as via su) as well as the ability to grow and shrink our infrastructure as problem sets’ computational requirements demanded. But we also wanted to integrate into the course’s own syllabus discussion of scalability, virtualization, multi-core processing, and cloud computing itself. What better way to teach topics like those than to have students actually experience them. Although Amazon supported our experiment financially with credits, it was not without costs. Serving as our own system administrators cost us time, as did some self-induced late-night technical difficulties. But the upsides proved worth it, as we accomplished our goals. We present in this paper what we did right, what we did wrong, and how we did both so that others can more easily build their own home in the cloud.

INTRODUCTION
Computer Science 50 is Harvard College’s introductory course for majors and non-majors alike, a one-semester amalgam of courses generally known as CS1 and CS2. Most of the course’s students (94%) have little or no prior programming experience. Although
the course introduces some fundamentals of programming in Week 0 by way of Scratch [20], a graphical programming environment developed by MIT’s Media Lab, the course immediately transitions to C in Week 1, where it spends much of the semester. Among the topics covered during that time are abstraction, encapsulation, data structures, memory management, and software development. The semester concludes with an introduction to web programming by way of PHP, SQL, and JavaScript plus XHTML and CSS.

With so much material to cover, the course itself moves quickly, demanding of students upwards of 15 hours per week outside of class, much of which is spent on programming assignments. With the exception of Problem Set 0, much of the course’s workload involves a command line. Students write C code with their choice of text editor (e.g., emacs, nano, or vim), compile that code using gcc, and then debug that code using gdb and valgrind. Toward term’s end, httpd and mysqld enter the picture as well.

The course’s problem sets, meanwhile, vary in their computational and I/O needs. Whereas Problem Set 1 has students implement some bite-sized C programs, Problem Set 2 tasks them with implementing ciphers and cracking DES-encrypted passwords. Problem Set 5 has students recover megabytes of photos from a dd image of an “accidentally formatted’ CompactFlash card. And Problem Set 6 hands them a 143,091-word dictionary that they need to load efficiently into a speller-checker.

With 330 undergraduates and 30 teaching fellows, the course tends to consume its share of cycles. Over the years, the course has relied for its needs on a load-balanced cluster of servers comprising Alphas running Digital (Tru64) UNIX or, more recently, Xeons running Ubuntu Linux. Via SSH (or, formerly, Telnet) can students log into that cluster, where a shell (tcsh) and NFS-mounted home directory await. Once done with their homework, students submit it electronically to the course’s own account on that same cluster via a setuid script.

Unfortunately, those home directories include only 100 MB of storage. Though plenty for source code, it has proved insufficient for students’ inevitable core dumps as well as for problem sets involving large datasets. Because we, the course, do not manage the cluster, having those quotas increased is never as easy as would be ideal.

Quite often, too, has the cluster suffered technical difficulties that get resolved no sooner than next business day. Unfortunately, our students tend to work 24/7, and prolonged technical difficulties not only induce stress, they affect the syllabus’s tight schedule if we resort to extensions.

Because this cluster is used not only by our students but also by other students, faculty, and staff, we do not have root access. When students need help, they must email us their code or go through the process of submitting electronically, neither of which is ideal for efficiency. System-wide installation of software, meanwhile, requires the help of IT, who often prefer not to install or upgrade software, lest it affect other users. And so we often resort to installing (sometimes with ease, sometimes with difficulty) software we need in the course’s own account (e.g., within /home/cs50/pub/), thereafter altering students’ $LD_LIBRARY_PATH, $LD_RUN_PATH, $LIBRARY_PATH, $MANPATH, and $PATH via a .cshrc that their shells source upon login.
And so we have long craved our own cluster of systems that we could optimize for the course’s technical and pedagogical goals. But with that vision comes a need not only for cash but also for space, power, and cooling, none of which we, as a course, have in any supply. In fact, the university’s own supply of those resources is increasingly limited. Moreover, we would prefer not to assume responsibility for technical problems (e.g., failed hardware) that might otherwise distract us from our students. And we simply do not need an entire cluster of systems every day of the year, as our computational needs vary with problem sets’ deadlines.

We thus moved CS50 into the cloud, building within Amazon Elastic Compute Cloud (EC2) [3] our own load-balanced cluster of x86-based virtual machines (VMs) running Fedora Core 8. Not only did we gain the autonomy we sought, we also gained pedagogical conveniences, every one of which helps with 330 students. The ability to see and help students debug code prior to submission \textit{in situ} (as via $\textsf{su}$), by itself, proved invaluable, eliminating unnecessary exchanges of code via email and premature electronic submissions.

In the section that follows, we elaborate on what it means to move into the cloud and provide background on EC2 specifically. We then offer implementation details on how we built our new home in the cloud. Thereafter, we present our results and admit to mistakes we made during the term. We then conclude and reveal our plans for next year.

**BACKGROUND**

Although it once represented, often in cartoon form, any network beyond one’s own LAN, “the cloud” now refers to on-demand computational resources whose provision usually relies on virtualization. In more real terms, cloud computing means that courses like ours (or, more generally, users) can pay for access to servers when and only when we actually need them. Those servers just so happen to be VMs, otherwise known as virtual private servers (VPSes), that live alongside other customers’ VPSes on hardware that we ourselves do not own.

Amazon EC2 offers precisely this service, as does an exploding number of competitors (e.g., Linode [16], unixshell# [24], and VPSLAND [25]). One of the largest providers, though, EC2 offers multiple availability zones, “distinct locations that are engineered to be insulated from failures,” in two regions (US and Europe). In the event that some availability zone becomes, if ironically, unavailable, an EC2 instance (Amazon’s parlance for a virtual machine) can be re-launched in another zone altogether. Users need not even be informed of such changes thanks to EC2’s elastic IP addresses, which can be re-mapped within minutes from unavailable instances to available ones.

An instance, meanwhile, can boot any one of several operating systems by loading at startup an Amazon Machine Image (AMI). Amazon currently offers AMIs pre-configured with Debian, Fedora, Gentoo Linux, OpenSolaris, openSUSE Linux, Red Hat Enterprise Linux, Oracle Enterprise Linux, Ubuntu Linux, and Windows Server [3], but customers can create [7] and share [6] their own too. Instances’ (virtual) disks are typically ephemeral, though, whereby any changes made to an instance’s filesystem (vis-à-vis some AMI) are lost upon shutdown (though not upon restart). But EC2 provides persistent storage via a service it calls Elastic Block Store (EBS) [2], whereby an EBS
volume (which can be 1 GB to 1 TB in size) can be attached as a block device to any one instance. Changes to that volume persist even after an instance is shut down, and that volume can be re-attached later to another instance altogether. EBS also supports snapshots, whereby copies of volumes can be archived using Amazon Simple Storage Service (S3) [4].

Instances can be created and managed from any networked computer using EC2’s Web-based management console, Java-based command-line tools [10], or SOAP- and HTTP-based APIs [7]. A number of third-party tools facilitate management too [8]. We ourselves are fans of the Elasticfox Firefox Extension [9]. Links to the EC2 resources we found most helpful appear in this paper’s appendix.

Although “cloud” computing is perhaps the buzzword du jour, we have found that few, if any, services offer precisely what EC2 does: an ability to provision on demand within seconds any number of (Linux) VMs without human assistance or additional contracts. Perhaps related in spirit to EC2, though, are “grid” frameworks like Google’s App Engine [14], Microsoft’s Windows Azure [19], University of Chicago’s Globus Toolkit [13], University of California’s BOINC [11], Princeton’s PlanetLab [23], and University of Washington’s Seattle [12]. But frameworks like these, inasmuch as they provide APIs and managed environments for execution of distributed applications and services more than they do familiar command lines and binaries, represent solutions to problems that we did not have. We simply wanted some VMs that we could wire together in order to implement a topology and workflow already familiar to us, per Figure 1.

![Figure 1](image)

**Figure 1.** Our cluster in the cloud comprised one “front-end” VM providing LDAP, NAT, and NFS services to multiple “back-end” VMs. From their laptops on campus or elsewhere, students SSHed throughout the semester to cloud.cs50.net, which mapped via DNS to an elastic IP address bound to our front-end VM, which routed each student to the back-end VM with the fewest connections.

**IMPLEMENTATION DETAILS**

For an introductory course like ours, it was not our goal to provide every student with his or her own VM but, rather, to provide, for the sake of uniform experience, a managed environment in which all students could implement problem sets. Although our own local infrastructure was imperfect, its overarching design had always worked well: a load-balanced cluster on which each student had his or her own shell account and home

114
directory. And so we sought to replicate that design inside the cloud, thereafter improving upon it.

Although EC2 provides firewallsing services between instances and the Internet at large (via what it calls security groups), it does not offer load balancing (except on high-numbered ports) for non-HTTP traffic, which we very much needed for SSH’s sake. And so we had to implement our own load balancer. (In the interests of simplicity for students, we were not willing to run sshd on a port other than 22.) We were determined to avoid a DNS-based solution (whereby a fully qualified domain name resolves, via an A record, to multiple IP addresses), as DNS caching by servers and clients can induce dead ends lasting hours or days.\(^1\) Prior to term’s start, we knew of no non-DNS solutions in use within EC2’s cloud, and so we set out to implement our own.\(^2\)

After much trial and error, we came up with a solution. We first spawned two instances using EC2’s AMI for a 32-bit installation of Fedora Core 8 (ami-2b5fba42).\(^3\) We then created a TCP tunnel between those two instances using VTun [18].\(^4\) We assigned one end of the tunnel an IP address of 192.168.1.1 (much like a home router) and the other an IP address of 192.168.1.10. We henceforth referred to 192.168.1.1 as our “front-end” VM and 192.168.1.10 as a “back-end” VM, per Figure 1. We next enabled IP forwarding on the front-end:

```bash
# Enable IP forwarding on the front-end:
echo "1" > /proc/sys/net/ipv4/ip_forward
```

Meanwhile, we told the back-end to route any traffic from the front-end back through the front-end:

```bash
# Route traffic through the front-end:
echo 80 lvs >> /etc/iproute2/rt_tables
ip route add default via 192.168.1.1 dev tun0 table lvs
```

We repeated this process for additional back-ends (with IP addresses 192.168.1.11, 192.168.1.12, etc.). Next, using Linux Virtual Server (LVS) [26], we told the front-end to route incoming SSH connections to one of the back-ends according to LVS’s Weighted

---

\(^1\) In the event a host in a cluster goes down, its IP address can certainly be removed from a load-balancing A record. If some client (or some other DNS server between it and the host) has cached that address, though, the client may continue to look for the host at that address until the cache actually expires.

\(^2\) We are indebted to M. David Peterson [17] for his help with our quest.

\(^3\) We avoided EC2’s 64-bit AMIs lest we discover mid-semester (i.e., too late) that some software we want only exist in a 32-bit flavor.

\(^4\) We also considered OpenVPN [22] but preferred the simplicity of VTun.
Least-Connection heuristic, whereby the back-end with the fewest existing connections wins the next one:\(^5\,^6\)

\begin{verbatim}
  ipvsadm -C
  ipvsadm -A -t 10.253.131.161:22 -s wlc
  ipvsadm -a -t 10.253.131.161:22 -r 192.168.1.10:22 -m -w 1
  ipvsadm -a -t 10.253.131.161:22 -r 192.168.1.11:22 -m -w 1
  ipvsadm -a -t 10.253.131.161:22 -r 192.168.1.12:22 -m -w 1
\end{verbatim}

At the time, 10.253.131.161 was the private IP address that EC2 itself had assigned to our front-end instance. Finally, we enabled network address translation (NAT) on our front-end:

\begin{verbatim}
  iptables -t nat -F
  iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 10.253.131.161
\end{verbatim}

And we mapped cloud.cs50.net via DNS to an elastic IP address bound to our front-end.

Not only did our front-end ultimately serve as a load-balancing NAT router, it also provided LDAP and NFS services to our back-ends as well. Usernames and groups lived in a tree on the front-end, and home directories lived on an EBS volume attached to the front-end that was then exported to each of the back-ends. Backups, meanwhile, were managed by rsnAPSHOT [21]. In fact, even though we preached the virtues of source control to our students, we took it upon ourselves to take snapshots of their home directories every 5 minutes. Thanks to its use of rsync and hard links, rsnAPSHOT maintains snapshots efficiently; it does not maintain multiple copies of files that have not changed. The sheer number of panic attacks we averted among students by empowering them to access so many versions of their code easily justified the additional gigabytes.

We installed on each of our back-ends gcc, gdb, valgrind, and more.\(^7\) On all of our instances did we also install Webmin [15], a web-based GUI that facilitates system administration. Ultimately, we relied entirely on freely available software and tools, almost all of it installed via rpm and yum.

It’s worth noting that EC2 offers different types of instances (i.e., classes of virtual hardware). We began Fall 2008 running 3 “small instances” (m1.small), each of which had 1 (virtual) core rated at 1 EC2 Compute Unit (the equivalent of a 1.0 – 1.2 GHz 2007 Opteron or 2007 Xeon processor) and 1.7 GB of RAM. Because problem sets’ computational needs rose over time, we ended the term running as many as 6 high-CPU medium instances (c1.medium) at once, each of which had 1.7 GB of RAM and 2 (virtual) cores, each rated at 2.5 EC2 Compute Units (i.e., 2.5 – 3.0 GHz).

Our front-end was indeed a potential single point of failure in our cluster. However, because an EBS volume (e.g., students’ home directories) can only be attached to one instance anyway, we considered that weakness an acceptable risk, particularly since we

\(^5\) We utilized Weighted Least-Connection instead of LVS’s Least-Connection heuristic so that we could take back-ends out of rotation temporarily simply by setting their weights to 0.

\(^6\) So that the course’s staff could still SSH to the front-end itself, we ran its instance of sshd on a TCP port other than 22.

\(^7\) For efficiency’s sake, we actually configured just one back-end initially, thereafter burning our own AMI based on it, and then spawning additional instances of that new AMI.
could re-spawn an identical instance within minutes if necessary, in a different
availability zone no less. Anecdotally, we’re happy to say that our front-end never
actually went down.

RESULTS

At the very start of Fall 2008, we briefly relied, as usual, on our local cluster of
systems. On 3 October 2008, though, cloud.cs50.net debuted among a subset of
students. Two weeks later, we moved all 330 students into the cloud, where they
remained until term’s end in January 2009. Table 1 details our cluster’s consumption of
resources during that time.

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Disk</th>
<th>I/O Requests</th>
<th>Bandwidth</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep</td>
<td>2,275 Hrs</td>
<td>125 GB</td>
<td>45,348</td>
<td>14 GB</td>
<td>$274</td>
</tr>
<tr>
<td>Oct</td>
<td>3,425 Hrs</td>
<td>108 GB</td>
<td>93,257,314</td>
<td>191 GB</td>
<td>$657</td>
</tr>
<tr>
<td>Nov</td>
<td>5,484 Hrs</td>
<td>199 GB</td>
<td>337,019,916</td>
<td>239 GB</td>
<td>$1,252</td>
</tr>
<tr>
<td>Dec</td>
<td>5,206 Hrs</td>
<td>300 GB</td>
<td>427,639,962</td>
<td>52 GB</td>
<td>$1,184</td>
</tr>
<tr>
<td>Jan</td>
<td>5,208 Hrs</td>
<td>300 GB</td>
<td>1,502,614,186</td>
<td>62 GB</td>
<td>$1,298</td>
</tr>
</tbody>
</table>

Table 1. Resources consumed by our virtual cluster between September 2008 and
January 2009. CPU refers to the number hours our instances collectively ran; between
October and November, we transitioned from single-core instances to dual-core instances.
Disk refers to space used (for home directories and snapshots) on EBS volumes. I/O
Requests refer to our I/O activity on our EBS volumes, as measured by iostat.
Bandwidth refers to network traffic to and from our cluster. Per the upward trends over
time, our students’ utilization of cloud resources rose with problem sets’ computational
needs.

We ultimately judged our new home in the cloud a success, as we gained precisely
the technical autonomy and pedagogical conveniences that we had set out to claim. Not
only did the cloud empower us to install software and change settings at will, it allowed
us to examine students’ code and reproduce errors therein in situ via su and sudo,
thereby expediting a common scenario. To be sure, we could have had these same
powers on campus had we run our own cluster with actual hardware. But cloud.cs50.net required no space, no power, no cooling from us. Someone else (i.e.,
Amazon) kept an eye on the hardware’s hard drives and fans. And provisioning more
capacity for students was as simple as clicking a button. In a word, reliability and scalability proved easy (or, at least, easier) in this cloud. In fact, EC2’s own Service Level
Agreement (SLA) [1] commits to 99.95% uptime, which, we daresay, is even higher than
we’ve experienced on campus in terms past.

But our experience was not without hiccups, most the result of mistakes made by
this paper’s author. Simply preparing the cluster took multiple weeks (perhaps 80 hours
in total), largely because we tackled at once so many packages unfamiliar to us. Our
quest to implement a load-balancing NAT router within EC2’s confines took particularly
long but proved well worth the journey. We temporarily ran out of disk space on our
EBS volume midway through term, the result of our having been conservative with space.
And we experienced I/O delays around some problem sets’ deadlines until we realized
that OpenLDAP’s slapd was keeping open too many file descriptors.
Although throughput between our virtual cluster and computers on campus was high, we did suffer latency, whereby X applications (e.g., emacs) performed poorly. We are not yet confident that we can eliminate that particular problem next time around.

Our time in the cloud cost us less than $5,000 (roughly $15 per student), although Amazon defrayed those costs with EC2 credits. In fact, Amazon now offers Teaching Grants “supporting free usage of [EC2] for students in eligible courses” [5], so our arrangements are by no means exclusive. Had we been more conservative with VMs and more attentive to usage, scaling our cluster’s size up and down more frequently, we are confident that we could have accomplished our goals for less, perhaps even half this amount.

CONCLUSION

In Fall 2008, we moved Harvard College’s CS50 into the cloud via Amazon EC2. We turned to the cloud in search of technical autonomy and pedagogical conveniences, both of which we realized in our new home. We even added capacity to our virtual cluster as the semester progressed. Yet we did not need to find space, power, or cooling on campus for cloud.cs50.net. And we did not even need to find dollars, thanks to support Amazon has since institutionalized for educators at large.

To be sure, building this new home did cost us time, but we think it time very well spent. Not only did we accomplish our goals, we introduced 330 students first-hand to the cloud. We not only told students about scalability, virtualization, multi-core processing, and cloud computing, we had them experience each. Ironically, had we not told them that cloud.cs50.net lived somewhere other than campus, most would not ever have known. But that in itself perhaps speaks to the potential of cloud computing itself.

In Fall 2009, not only will we continue to spend time in the cloud, we will also make available a stand-alone replica of cloud.cs50.net, a virtual machine that students can download and run on their own Macs and PCs, even offline. Our goal now is to distribute a whole “course in a box.”

APPENDIX

To get started with EC2, we recommend these resources:

Amazon Elastic Compute Cloud
http://aws.amazon.com/ec2/

Getting Started Guide
http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide/

Developer Guide
http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/
ACKNOWLEDGEMENTS

Many thanks to Kurt Messersmith, Tracy Laxdal, and Deepak Singh of Amazon Web Services, to M. David Peterson, and to CS50’s own Glenn Holloway and Keito Uchiyama, without whose support this experiment would not have been possible. And many thanks to Fall 2008’s 330 students and 30 teaching fellows for taking and teaching a course in the cloud.

REFERENCES


LEARNING COMPUTER SCIENCE CONCEPTS USING

IPHONE APPLICATIONS

Judith O’Rourke, Ian MacDonald, David Goldschmidt
Department of Computer Science
The College of Saint Rose
432 Western Avenue, Albany, New York 12203
{orourkej, macdonai, goldschd}@strose.edu

ABSTRACT

Students use computers and technology at increasingly younger ages. Nonetheless, enrollment in degree-bearing Computer Science programs is not keeping pace with the need for professionals in the field of software development [4]. Even in today’s technology-rich world, Computer Science is still often viewed as an esoteric and cryptic field of study. While students tend to be fluent in using computers and handheld devices, understanding how these devices work and especially how they are programmed largely remains a mystery. To address this problem and attract potential Computer Science students at the middle and high school levels, we have developed iPhone applications that present fundamental Computer Science concepts in the form of a game. In particular, we have implemented Bubble Sort and Binary Search applications. Our approach “talks” to the younger generations by using a medium they are already comfortable with.

INTRODUCTION

Though we live in an abundantly technological age, enrollment in degree-bearing Computer Science programs falls short in producing computing professionals for the American workforce [4]. We find this to be surprising given the widespread use of computers and technology at increasingly younger ages. To introduce Computer Science concepts to future would-be Computer Science students and professionals, we propose a series of instructional games for the iPhone, or more generally, for any programmable mobile device. Figure 1 shows a sample screen shot of a Bubble Sort application.
Our fundamental approach is to reach the younger generations by using a medium they are already comfortable with and even prefer over more traditional classroom tools and software, including a myriad of Web applets and applications.

![Figure 1: Bubble Sort application, as shown in the iPhone Simulator](image)

**PROGRAMMING iPhone OR iTouch DEVICES**

Programming applications for iPhone and iTouch devices requires an Intel-based Macintosh running Leopard (OS X 10.5.3 or later) and Apple’s free software development kit (SDK). The SDK contains Xcode (3.1.4), which is Apple’s integrated development environment (IDE).

Xcode includes tools for creating, debugging, and compiling Objective-C source code. In particular, the SDK includes the Interface Builder (IB) application that provides a GUI in which elements are dragged from a palette of predefined controls and dropped into a design window. The iPhone Simulator application (see below) is also included in the SDK, allowing programmers to run and test their applications.

Note that Apple’s free SDK does not allow programmers to download applications to an actual mobile device or distribute applications via Apple’s iPhone App store. Instead, a programmer must join Apple’s Developer group for a fee, though this fee is waived for academic institutions. On the plus side, Apple’s Developer group provides many tools and resources to learn and develop iPhone applications, so the financial cost may be appropriate when these benefits are taken into consideration.

**Simulation Environment**

The iPhone Simulator provides a direct means to test mobile applications without having to download to an actual iPhone or iTouch device. Aside from basic functionality, the iPhone Simulator also allows you to simulate rotating the device to the left or right, as well as performing the “shake” gesture.
Though invaluable to the programmer, the iPhone Simulator does currently have some limitations and pitfalls. It cannot simulate the iPhone accelerometer or camera features, which we have therefore avoided. Further, careful attention must be given to mimicking a finger touch (via the click-and-drag of the mouse) when implementing the touchesMoved() and touchesEnded() methods. While such finger touch movements are natural on an actual iPhone, simulating those same movements via a mouse is unintuitive.

Another limitation of developing iPhone applications is that the development, testing, and deployment must entirely be performed on Apple hardware. Because many schools do not have Apple computers available or determine that such computers are too expensive, this is a hindrance to integrating iPhone applications or their development into the curriculum. To address this problem, writing applications for mobile devices other than iPhone/iTouch can be accomplished using other platforms, including the Java ME platform.

**Programming in Objective-C**

Objective-C is an object-oriented programming language designed by Brad J. Cox in the early 1980s [2]. Objective-C is a combination of the imperative language C (in particular, ANSI C) and the “pure” object-oriented language Smalltalk, one of the first object-oriented languages. Because Mac OS X applications are written using the Cocoa framework, which is written in Objective-C, it is no surprise that iPhone/iTouch applications also use this same language. More specifically, these devices use the Cocoa Touch framework, which is a scaled-down version of Cocoa.

As with many high-level languages, Objective-C’s corresponding Xcode IDE provides color-coded text editing, syntax error highlighting, auto-completion, and other such beneficial features. Note that Objective-C source code can be programmed and compiled using Xcode or the GNU Objective-C compiler directly in a terminal window.

**A BUBBLE SORT APPLICATION**

Fundamental to Computer Science, bubble sort is often one of the first sorting algorithms students learn. Though certainly not the most efficient sorting algorithm, bubble sort is simple to understand and simple to implement. By using bubble sort, students learn fundamental skills in both algorithm design and programming.

The bubble sort algorithm typically requires a nested loop. Within the inner loop, each pair of elements is swapped if discovered out of order. For each iteration of the outer loop, a correct element “bubbles” up (or down or right or left) to its final position [1].

Our approach is to teach bubble sort using an interactive interface in which students walk through the algorithm to sort a small list of unsorted numbers. Figure 1 shows the starting point of the Bubble Sort application. When a student clicks the “Start” button, the first pair of numbers (e.g. 7 and 2) “drop down” and are shown in green (see Figure 2).

The student must decide whether to swap the two numbers. To do so, the student drags the “green 2 tile” into the first position where the 7 used to be. To keep a given pair
of numbers in the same order, the student must drag either of the green number tiles into their proper position. The application then presents the next pair of numbers (e.g. 7 and 5) and the user keeps going until the list is sorted. Upon completion, the application appears as shown in Figure 3.

If the student attempts to move a green number tile into the wrong position, the green number tile will move back to where the student dragged it from. The expectation is that the student will realize his or her mistake and correct it. In doing so, the mechanics of the algorithm are reinforced. In addition, the student may click on the “How it works...” button to view a description of the algorithm, including pseudo-code.

Figure 2: First decision point in the Bubble Sort application

Figure 3: Successful completion of the Bubble Sort application
A BINARY SEARCH APPLICATION

Before students learn bubble sort, they often study linear search and binary search, gaining an understanding of how much more efficient the binary search algorithm is. Given a list, the problem is to find a target element in the list (or determine that the target element is not in the list at all). The linear search algorithm steps through each element of the list until the target is found; if not found, the target has been compared to every element of the list. Therefore, in the worst case, \( n \) comparisons are made, where \( n \) is the size of the list.

A more efficient approach is the binary search algorithm, which requires the list to already be in sorted order. The binary search algorithm uses a binary tree to store all values in sorted order [1]. At each node of the tree, the left branch contains all elements less than the given node; the right branch contains all elements greater than or equal to the given node. For each iteration of the algorithm, if the given node is the target element, the search is successful; otherwise, the search continues “down” either the left or right branch, eliminating up to half of the subtree at each iteration. This process continues until the element is found or a leaf node is reached and no further descent down the tree is possible.

Making Binary Search Interactive

Similar to our bubble sort application, our approach is to introduce binary search using an interactive graphical interface. As shown in Figure 4(a), the student is presented a target element to search for (e.g. 39). Initially, the student is not shown the entire
binary search tree; instead, the root node of the tree is shown, along with arrows pointing to clickable tags “go left” and “go right.”

At each iteration of the binary search, the student decides whether to follow the right branch or the left branch. For each correct decision, the student sees the next node. From Figure 4(a), when the student correctly selects “go left,” the path from node 47 to node 22 is shown in green (see Figure 4(b)). If the student makes an incorrect choice, the node “shakes,” showing the student that his or her choice was incorrect.

Figure 5(a) shows the next step in the example; and Figure 5(b) shows the final step. When the target node is found, the entire tree is revealed.

INTEGRATING iPHONE APPLICATIONS INTO THE CURRICULUM

The key to our work lies in the planning of how to utilize the aforementioned applications in various classroom environments. Our underlying goal is to encourage and enlist students in degree-bearing Computer Science programs at the college level. Since this often begins at the middle and high school levels, we are developing plans to collaborate with such schools to integrate both the use and the programming of iPhone/iTouch applications into the corresponding curricula.

One example of how we plan to accomplish this is to develop summer programs for students and teachers alike, including “teach-the-teacher” sessions. For younger grades,
use of our iPhone applications (and other similar apps) may spark early interest in Computer Science, once again through the use of technology already very familiar to such younger generations. Through simple observation, we find many in these younger generations that constantly have their iPhones or mobile devices in hand, communicating with others, playing games (e.g. Sudoku), and the like.

A College-Level Course in Programming Mobile Devices

Also as part of our work, Dr. O'Rourke offers a new course in programming mobile devices (beginning in the Spring 2010 semester). In part, we offer this course to generate interest in Computer Science among non-majors, perhaps “converting” some exploratory or undecided majors into Computer Science majors. The sole prerequisite is one semester of an object-oriented programming course.

While this new course focuses on hands-on iPhone application development, the course also requires students to build interactive learning tools for Computer Science, including sorting algorithms, search algorithms, greedy algorithms, probability, and binomial expansion. Students develop programming skills while learning and reinforcing Computer Science concepts. Further, the developed iPhone applications may be used in other introductory classes (and in middle and high school programs) to demonstrate Computer Science concepts and encourage more students to study Computer Science.

DISTRIBUTING TO THE MASSES

To distribute the applications discussed here, along with future applications, we plan to make them available for free via the iPhone App Store. Students in our introductory Computer Science courses and related service courses will be given direct links to the applications. Those students without an iPhone will be provided opportunities to use college-owned devices, including actual iPhone/iTouch devices and simulation systems. And, as described above, to extend beyond just our college-aged audience, we aim to reach out to local middle and high schools, in part to develop future Computer Science students.

CONCLUSIONS AND FUTURE WORK

Beyond the bubble sort and binary search applications, we plan to develop applications illustrating such concepts as encryption, routing algorithms, bin packing, Sudoku puzzle generation and solving, pairwise sequence (string) alignment, phylogenetic tree building, and many others. Development of these new applications will involve undergraduate students in the aforementioned Programming Mobile Devices course, as well as students from within our graduate program.

While the iPhone continues to enjoy a significant market share, other Java-based mobile devices are beginning to rival this current market leader. Therefore, we also plan to develop Java-based versions for each of our instructional applications.

Finally, with the much-anticipated release of the Apple iPad, future applications may take advantage of a larger screen size and more extensive functionality.
REFERENCES


EVOLUTION OF CAPSTONE-COURSES IN SOFTWARE ENGINEERING

A FINISHING SCHOOL

Timoth C. Lederman
Department of Computer Science
Siena College
Loudonville, NY 12211
518 783-4197
lederman@siena.edu

ABSTRACT

This paper describes the two-semester capstone courses in Software Engineering offered in the Computer Science Department at Siena College. These courses have been offered for 27 years and they have recently become a graduation requirement for the major. The courses tie-together a variety of educational themes and goals related to Software Engineering, professional behavior, and preparation for professional careers. Over the years, the syllabi, content, format, and teaching-approach has changed substantially. This paper describes the two Software Engineering courses, and how the courses have changed, and how faculty and students perceive the courses.

INTRODUCTION

Siena College is a primarily undergraduate Liberal Arts College located near Albany, NY. The College has 3,100 full-time students. The Computer Science Department has 13 full-time faculty and approximately 80 Computer Science majors. The curriculum for the Computer Science major is 11 courses (34 credit hours); and, students may take up to 13 courses in Computer Science. The two Software Engineering courses are now a graduation requirement for majors and serve as a senior-year capstone experience.
Since these courses are a part of a Computer Science curriculum that closely matches the ACM/IEEE Computing Curricula [1], these courses are significantly different from a capstone course recommended for a Software Engineering major [2]. First and foremost, our courses serve to introduce the students to the theory and practice of Software Engineering. The students are guided by lectures and examples. In both courses, the students work in teams, applying the principles of Software Engineering on a real project, with a real client, in a realistic environment. This is theater in one sense, but the students are learning their parts step-by-step and act-by-act, without knowing the nature or purpose of future acts – the students learn their parts, play their parts, feel their parts, and often take ownership of their parts. By the end of the course, in a successful year, the students understand the purpose of each act and are proud of their final accomplishments. However, learning software-development techniques and learning how to organize people and activities are not the only themes and goals in the course. From the beginning of the course, the students are informed that they are in a “Finishing School” that is (1) an academic-culmination that draws upon all their other courses; (2) an experience that should give them confidence that they are ready to go out into the “real world”; and, (3) an opportunity for them to become aware of their professional responsibilities.

OVERVIEW

During the first semester, the class meets on Mon/Wed/Fri at 8:15 am (as a “welcome to the real world”). Students are graded on their knowledge of principles that are presented and discussed in class (and reinforced by their readings). There are two short-essay exams (50% of the course grade). The other 50% of the course grade is based on performance on the project. During the second semester, the class meets once a week (6-9pm) and the format, pedagogical style, and grading change (70% project and 30% class participation). The general format is: (1st Hour) either a lecture on a topics related to the project or related to professional behavior, or a presentation by an alumni on a topic related to tools or techniques used in software development. After Q&A, the speakers discuss their professional careers, the professional development they have undertaken, and how to balance professional and personal lives. These are valuable opportunities for mentoring and networking. (2nd Hour) a student-led discussion on issues related to software development: professional ethics; legalities; privacy; working in a technological environment; and, social-professional responsibilities. These discussions are tied to The Case of the Killer Robot [3]. Each week, one team leads a class-discussion on a Killer-Robot chapter. Near the end of the semester, a team outlines and compares three codes of professional Ethics (ACM Code, IEEE Code; and, IEEE Software Engineering Code). (3rd Hour) each team has a quick “walk-through” with the instructor (“what’s been done since last week; what’ll be done by next week; and are you having any problems”). These exercises model a professional activity, and they provide a reality check.

The students learn several definitions of Software Engineering, but the courses give emphasis to Roger Pressman’s definition “Software Engineering is a discipline that integrates process, methods, and tools for the development of computer software.” [4] Our courses give emphasis to Pressman’s four Ps: “People, Product, Process, and Project.” Our students learn Software Engineering principles, tools, and techniques; they follow a process; and, they produce deliverables throughout the year.
Throughout the project portion of the two courses, students are provided feedback at major milestones. There are also mini-milestones wherein students are required to produce examples (related to their projects) of what is covered in lectures and readings. In the early years of these two courses [5], the lecture portion of the course was separated from the project-activities of the students. Students were provided an outline of the components of each phase of the project. Tools and techniques were presented in class. However, students worked almost entirely on their own (as teams) to develop their own understanding of what needed to be done – feedback was only provided at five major milestones over the two semesters. In those earlier years, sometimes students found themselves pursuing non-productive directions - the result of misunderstanding what needed to be done, or the result of miscommunication with a client. The addition of mini-milestones (website-critiques, User Case Narratives, UML diagrams, Requirements Inventory, DFDs, Data Dictionary, and Test Plan) has resulted in the students making better use of tools and techniques. The scheduling of more regular meetings with clients, and the course-requirement of rapidly producing/publishing meeting notes has resulted in students moving more correctly and more productively through the processes. Students are still expected to figure out what needs to be done, but they are now provided more guidance and feedback throughout the project. The use of similar strategies to enhance student-success in a team-project-based course has been reported at other institutions [6].

THE PROJECT – THE TEAM – THE PROCESS

Over the past 10 years, the general requirements for all projects have been: (1) a web-based application; (2) a database must be developed and used; and, (3) a scripting language must be used (usually PHP). Most of these topics have been covered in earlier upper-level elective courses in the curriculum, but not all students are familiar with all these areas. While the Software Engineering course is a capstone built on theories, content, and skills learned in earlier courses, this capstone assumes that the students having gained the ability to learn new skills on their own, as they would after graduation.

The clients give a 2-3 minute presentation on the first day of class. On the second day, the class votes on team leaders (nominated by faculty, by students, and by self-nomination). The team leaders then meet privately with the instructor to randomly select their project and to choose team members (sand-lot-style selection of members; room-mates are not allowed on the same team). Team-sizes range from 4 to 8 members, but 5 to 6 members are an optimal size. Team leaders serve for one semester. The instructor chooses different team leaders for the second semester – students apply for the position and formal interviews are held. Every Team Member has a job title (e.g., team leader, testing leader, librarian/documentarian, webmaster/mistress, project manager, development leader, or systems administrator). Every team member must participate twice in the five formal presentations to their clients. Teams are encouraged to work on their communication skills by rehearsing – the instructor provides feedback after every presentation. Guests, faculty, and students always remark on the improvement they observe throughout the year in the content and in the delivery of the formal presentations.

The teams are announced in the third class. By the second week of the semester, the teams will have chosen a team-name and a project-name, and they have scheduled bi-weekly team meetings and bi-weekly meetings with their clients (less often in the second
semester). Team leaders meet privately with the instructor at least once a week. During lectures, the instructor provides “just-in-time” instructions on what the students should be doing and how they should go about their activities. Class discussions and activities are normally involved with Software Engineering theory and practices and how the students are to carry out these practices in their projects. The Software Engineering model that is followed is the classic Waterfall Model, which is convenient for setting due-dates (Milestones) over the two semesters: (1) Software Plan, (2) Requirements Specification, and (3) Preliminary Design (in the first semester); and, (4) Detailed Design and (5) Acceptance Test (in the second semester). The course syllabi, calendars, reading material, documentation, and eight years of team websites are published on the course website [7].

THE CLIENT

The client has a problem that is suitable for a software development project. Sometimes the instructor approaches someone to consider defining a project and becoming a client; and, sometimes a client knew about the nature of the Software Engineering courses and contacts the instructor. In the early years of the courses, projects were often batch-oriented and involved scheduled updating and reporting. Also, during the early years of the course, the clients were often off-campus individuals or organizations. For those projects, client-meetings could easily and adequately be scheduled once or twice a month. Such clients could be at distant locations (one client was a tax-assessor’s office 40 miles away). In those projects there was no need for the students to meet regularly with clients during the Detailed-Design stage (or during Development & Testing). If a meeting was cancelled, rescheduling was often a problem for both the clients and students.

More recently, the clients have been on-campus individuals, and it has become common for clients to be faculty in Computer Science. This is a consequence of requiring an iterative development of requirements, necessitating two team-client meetings per week. More recently, the projects have been web-based, with complicated user-interfaces. Regular interaction with clients has become a necessity, not only in the Requirements Specification “discovery” phase but also while screen-prototyping, which often extends to the end of the Detailed Design phase of the project. Prospective clients are briefed (screened) by the instructor prior to the beginning of the Fall semester. Clients are informed that first-and-foremost the project is an academic exercise – there is no promise of working-software being delivered at the end of the academic year. Clients are asked to avoid guiding the students, especially during the requirements-gathering stage and the design/development stages - students have to figure things out on their own.

THE DELIVERABLES

Half of the first-semester course-grade and all of the second-semester course-grade is based on the teams’ and students’ performance in carrying out their project. At each of the five Milestones, the students/teams are evaluated on the following: documentation; oral presentations; team evaluations (by each member and by team leaders – of every member of the team); updated resumes; and, team websites. At the Acceptance Test (at the end of second semester), each team submits a CD/DVD that contains: the team’s
website; the lyrics and music to the team song; and, a README.TXT file that explains what files are where. An enjoyable and greatly anticipated tradition of the courses is an end-of-year Party for students, clients, faculty, and alumni/speakers. At this party, each team presents a Team Song (of widely-varying quality, but recorded for posterity). To gain entry to the party, the students must submit a written course-evaluation called The Legacy, which is sealed and not read until after Commencement. The students are encouraged to include in The Legacy their opinions about what works, what does not work, and how to improve any aspect of the courses. The final “deliverable” takes place at the end-of-year departmental party/luncheon, where all the team songs/videos are played for the enjoyment-of (and abuse-from) faculty and underclass majors.

CAREER PREPARATION

In the early 1990s, a small amount of lecture/discussion would be directed towards career development/preparation. Emphasis was placed on strategies for finding a job, activities related to graduate-school application, and advice on how to prepare for a job-interview. Over the past 10 years, these activities have been broadened and formalized to include: lectures and speakers on types of graduate education and how to apply for each; strategies for finding a job and the application process (cover letter, resume, interview, and follow-up); attendance at a Job Fair & Networking event (and strategies for making the best use of these activities); and, a Mock Interview. The students are required to submit a revised resume at the beginning and end of both semesters. The resumes are reviewed by the Career Center and must contain the URL of their teams’ websites, and information about their roles on their teams. Before the Mock Interview, students are informed of technical-interview and behavioral-interview questions. For the Mock Interview, a job-opening is posted by a mock company. Students submit a cover-letter and resume. They arrive at the Mock Interview with a folder that contains a copy of the job-posting, the cover letter, a resume, and a list of references. They are supposed to have learned about the job and the company (via its website). The mock interviewer is an alumna/us. A thank-you letter must be sent within 24 hours of the interview. Most of these career-preparation activities take place out of class, thus taking up very little class time.

PROFESSIONALISM

The students are familiarized with a variety of social and professional “etiquette-skills” via lectures, guest-presentations, readings, and a Dining-Etiquette Seminar/Dinner (with guests that include clients, faculty, and guests). This seminar covers how to respond to an invitation, how to know what to wear, how and what to order (food and wine), place-settings/glasses/cutlery, and “signals” to a waiter. In the seminar and in class, social and cultural taboos are discussed. Throughout the year, students learn to show up on-time (starting with an on-time class-attendance requirement), how to respond to an invitation, how to dress-for-success. “Full dress” business attire is required at: five formal project-presentations; the Dining Etiquette Dinner/Seminar; a Networking event; and, at a campus-wide student-presentation event. Learning and practicing social-etiquette skills starts in the first class session when the students are instructed on how (and when) to shake hands; a skill that is regularly practiced with clients, and at the Dinner/Seminar,
and at the Networking Event, and at the Mock Interview. The students are instructed how
to make introductions, how and when to maintain eye-contact, and how to become known
and remembered (as being professional and socially adept). Short lectures, readings, and
discussions on other etiquette-related topics include the use of cell-phones/PDAs, and the
use of email/telephone/voicemail. Students are informed of ways to follow-up with
informal and formal thank-yous (and when to do so). Social/professional etiquette skills
are exercised (and thus reinforced) throughout the year.

OUTCOMES - THE LEGACY

The effectiveness of the Software Engineering courses is judged in a variety of
ways: on-line student-evaluations (9.7/10 for 18 respondents over two years); The
Legacies; and, feedback that is regularly received from alumni, internship sites, and
employers. In almost all cases, the courses are deemed to be of significant importance as
a capstone-experience in Computer Science. By the end of the two-course sequence, most
of the students indicate that they have learned a substantial amount of new and valuable
information and skills (comments about too much work are common, as are comments
about the value of working on a team and being prepared for “the real world”). Every
summer, the instructor reads the Legacy documents and course-evaluations, and
reconsiders every aspect of the courses. Adjustments and improvements in content, in
teaching techniques, and in pedagogical strategies are then incorporated into the next
year’s curriculum. Within a year after graduation, the alumni almost universally report
that they are benefitting from the theory, vocabulary, experiences, and discipline that they
gained through these courses. Students who enter graduate programs in Computer Science
and Information Technology indicate that they have enough theoretical knowledge to
handle the more rigorous content in graduate-level Software Engineering courses; but,
more importantly, alumni indicate that they have the practical, technical, and
organizational skills to perform and produce at the highest level. These two courses
demonstrate that our students have attained a level of professional and personal growth
(a “progression in leaning” goal [8]) that our faculty thinks is valuable from a capstone
course. The Software Engineering courses, a Finishing School, seem to successfully serve
to give the students self-confidence and an identity as professionals who have a sense of
responsibility to themselves, to their teams, to their clients, to society, and to their
profession.

REFERENCES


SUPPORTING AGILE PROCESSES IN SOFTWARE ENGINEERING COURSES

Delbert Hart
Computer Science Department
SUNY Plattsburgh
Plattsburgh, NY 12901
518 564-2775
hartdr@plattsburgh.edu

ABSTRACT

Software processes must be customized based on the project and the team working on the project. This paper discuss our use of agile software processes in an introduction to software engineering course. An overview of the strengths and weaknesses of tools used in the course over the years is given. We then present a new web application for software engineering courses.

INTRODUCTION

A software process is a blueprint for how team-members create a specific software product. Processes vary in their specifics, but typically include planning, requirements, design, coding, and testing. One of the surprises that many students encounter in a software engineering course is that coding itself is only one (often small) component in software development.

Software processes can be labeled as traditional or agile. There is no specific criteria for what is considered agile and what is traditional. One way to characterize the difference between them is by their approach to communication. Agile methods encourage frequent communications, traditional methods encourage correct/complete communications. More information about the distinction between traditional and agile methods can be found elsewhere, e.g. [1]. In the last decade agile methods have gained in popularity for some types of projects. One of the areas that agile methods has been successfully utilized in is computer science courses (e.g., [11, 13, 15, 16]).
The next section describes some of the benefits of agile methods in the classroom. Then we describe our experience with software engineering tools and how well they worked in the course. The last section describes a new tool that we are developing to support agile methods in software engineering courses.

**COURSE OVERVIEW**

Our software engineering course is taken by students from a traditional computer science major and by students in an information technology major. Students can take the course immediately after their first programming course. The software engineering course has an emphasis on requirements and design. The course has a lecture component that introduces students to the concepts and methodology of software engineering. There is a project component where the student gain experience with applying the concepts to either one or two projects that span the semester.

Having used both traditional waterfall based and agile based software processes for software engineering courses, our experience has been that students respond better to the agile based processes. Students vary widely in the skill sets that they arrive to the course with. Using an agile process for the course provides the instructor more opportunities to adjust the project to reflect the students' interests and capabilities.

Motivationally the agile methodology is a good match for students. For example, students working on a traditional requirements specification could mechanically follow the frameworks and outlines provided, but they lacked the experience to intrinsically see the value in having a good requirements document. In contrast students can see the rationale of writing down User Stories as a first step in describing a software product. The other work products in an agile process are likewise easier for the students to see the value in compared to the corresponding work products in a traditional process. Better understanding of the work products also improves the student's ability to perform technical reviews.

Students develop a number of different work products during the Introduction to Software Engineering course. The work products are assigned based on an increasing level of technical analysis. Initially, students work on User Stories, which are paragraph long descriptions of a piece of functionality. In a professional environment User Stories would be written by the customers. In the course the User Stories get students to start thinking about what the product should do. Some semesters we have introduced acceptance testing at this stage, other semesters we have delayed the testing topic until later. Next, students develop detailed Use Cases based on the User Stories. The Use Cases provide an event sequence that elaborates on the specific actions that can occur. At this point students can start to do some basic UML class diagrams, identifying the possible objects, attributes, and methods that need to be present. From here a variety of different work products can be developed. Function points can be derived to estimate the relative amount of work needed to complete the product, and/or a specific User Story. The Use Cases can be further developed into sequence diagrams and then swimlane activity diagrams. After taking the product from User Stories to the design documents students are given the option of which work products they want to work on for the rest of the semester. How the students interact with each other changes over the course of the
semester. Initially, students can work almost totally independently because they are each
developing their own User Stories and Use Cases. As the design work begins and reviews
start, they need good support for asynchronous collaboration.

Many agile methods operate on a daily schedule. Scrum uses a daily meeting to
bring team members up to date on the current status of the project and to facilitate
communication among the team. Many others require that a developer's work be
integrated into the main build on a daily basis and then checked against a suite of
regression tests. These frequent interactions help prevent small problems from
snowballing into larger delays or difficulties.

Although frequent interactions are important, daily interactions are infeasible in an
academic setting. In our experience weekly or semi-weekly interactions work well.
Students have a work product due on Monday morning of each week and the instructor
typically gets feedback to the students on their submissions by Tuesday morning's class.
This quick turn around time for the feedback is important for students to make progress
on their projects. When we have been unable to get students feedback within a day or two
we noticed a drop in project momentum. Students often need help with the technical
aspects of what they are working on. How to create good designs, how to specify good
requirements, etc. Since the student builds upon his past work it is important to provide
early feedback. If it is not readily apparent what changes a student has made, then it can
take an instructor much longer to give pertinent feedback. This is especially true as class
sizes increase. Students benefit from quick feedback from their instructors by correcting
errors in their work before they invest more effort into work products derived from
defective ones. This need for quick feedback necessitates good tool support for the
instructor to be able to easily see the changes that each student has made and be able to
place it in the larger context of the project.

Over the years we have used a variety of different software engineering tools to help
the students create and edit the various work products. The next section, talks about our
experiences with several of these tools.

TOOLS

Software engineering tools are important to the successful development of projects.
In a one semester course though there is a trade-off between utilizing a feature-rich tool,
with a corresponding learning curve, and a specialized tool for the academic setting. Most
software engineering tools are geared towards professionals where the cost of a steep
learning curve can be amortized more easily than in a 15-week course.

Another difference between professional developers and students is in their
interaction patterns. Professionals work on a project every day and have established
mechanisms for regular communication. Students enter a course with a variety of
preferences and constraints on how they collaborate and correspond with each other.
Hence we have found it useful for students to be able to work relatively decoupled and
asynchronously. One drawback to agile methods in the classroom is that this difference
in interaction style is more apparent. Agile methods are heavy on communication and
collaboration. For some teams of students there is no problem in communication, but
other teams of students have problems communicating frequently. The only guaranteed
times that students can collaborate is in or near class time, and with the course only meeting twice a week that is not sufficient to advance the project.

Tools are important in a software engineering class by facilitating the creation and sharing of the work products associated with a software project. Some important aspects of software engineering tools are: 1) collaboration model, 2) versioning support, 3) work product support, 4) availability and 5) granularity support. The collaboration model supported by a tool describes the modalities available for teammates to collaborate. Versioning describes the ability of the tool to track the history of changes to the work product. Work product support is which types of work products that the tool can handle. Availability is an important factor in how easy it is for the student to use the tool on their own machines. In our discussion we focus on open-source tools that are easily available for students to download and use. Granularity describes the size of change that is supported. Finer grained change support facilitates student collaboration and instructor evaluation of student work.

Eclipse[4] is an integrated development environment that uses a plugin architecture to support a wide variety of features and capabilities. Eclipse has been our primary tool for code development and unit testing, and it has been very capable in these areas. For requirements, design, and reviews it has not been as useful. Eclipse does have several plugins that support UML diagrams, but the semesters that we tried them the students found the interfaces to the UML diagrams non-intuitive. Versioning is achieved via a plugin for a source code management system (Subversion). There has been two problems that students have had with Eclipse: 1) navigating and understanding all of the features present and 2) installing and managing plugins.

There are other good integrated development environments that we have not used in the software engineering course. BlueJ[3] is a widely used environment for Java development. It has strong support for design diagrams as part of the development process. JGrasp[10] is another editor with good design support. Eclipse itself has a huge number of plugins that we have not tried as part of the course. One interesting plugin, Green[6] provides round-trip editing, which allows changes in the code to be reflected in the design documents and vice-versa.

UML is a design notation and is widely used for software engineering courses[9]. For the past couple of semesters we have used ArgoUML[2] for creating UML diagrams. The students have found this tool to be more intuitive than the other UML creation tools that they tried. It is written in Java and is freely downloadable for use, either as a standalone product or as a plugin to Eclipse. We used the standalone version because of some difficulties we encountered getting the plugin to work. The main disadvantage that we had with ArgoUML is that it is not well integrated with our other work products or with our review system. Another difficulty is that it saves the UML models as a single xml file, which makes it difficult for students to work on different parts of the same model concurrently. Another similar tool for UML editing is UMLet[7]. Vector and raster drawing programs are another option that students have used for creating UML diagrams. The advantage of these tools is ease of use. The disadvantages are lack of domain specific support for UML and difficulty in tracking changes between versions.

User stories and use cases are simple enough that text files can be used. Students use Subversion to commit them to the repository where other students can check them out and
review them. We adopted a common file format and naming scheme for the files. Reviews were appended to the file. One of the problems we had using text files is how to manage changes. Students could have used the diff capability of Subversion to find out what had changed, but preferred to keep the changes explicitly in the file. We found this to be interesting, since performing the diff operation was a single command. They preferred a solution, that (in our opinion) was less accurate, but was simpler.

Another option that we have tried for user stories and use cases was to use a wiki. Wikis are simple to use and can be useful as a project collaboration tool[8]. A strength is the simple versioning model and ability to see changes made. One of the main drawbacks of wikis is the unstructured nature of the data makes it difficult to support software engineering work products. While the students found it easy to work on individual work products, it was more difficult to see the overall status of the project or changes in related work products. After trying both most students preferred to use text files for use cases and stories. We have not experimented with extensions such as SemanticWiki or wikis such as Twiki that do support structuring.

We use Trac[5] as a web interface to the Subversion repository. It is useful to browse the history of changes. From an instructor's point of view it makes it easy to see recent changes made by students. One semester we tried using the issue tracker associated with Trac, via the Mynln interface in Eclipse. The students were not regular about keeping the tasks up to date via the issue tracker and creating the issues in a timely manner. Trac also has a built-in wiki. One nice feature of Trac's issue tracker and wiki is that it is easy to create hyperlinks between commits, issues, and wiki pages.

We have experimented with tools such as Bugzilla and issue trackers for project management tasks. Currently we do not use these tools because the first half of the course is dictated by our coverage of the various work products, and in the second half of the course we coordinate activities via other communication channels. Project management tools are useful though and frequently used to coordinate projects in software engineering courses (e.g., [12] ).

AGILE WEB

Although there are good existing tools for the work products that the students create, there is a need for a simple tool that can handle all of the work products. In a one-semester course the cost, in terms of time in class and/or project momentum, limits the number and sophistication of tools utilized. An ideal tool would:

1, Allow the work products, starting with the User Story and going through coding and testing to be tightly integrated to facilitate traceability.

2, There would be support for reviews to be tightly integrated with the products, and there would be fine-grained tracking of changes.

3, The tool would be simple enough to minimize the learning curve needed to use the tool, yet have enough features to support how students would use the software. The software should be able to be used by the student on whatever computing platform they have.
These features are not extraordinary, but these needs are different enough from the needs of professional developers that existing tools are not a good match.

The interaction paradigm that we chose is one similar to a wiki. Anybody can modify anything, but a complete history is kept of all of the changes to make it simple to roll back to an earlier version. We chose to develop this tool as a web application to allow students to interact with the tool from anywhere with the same interface and centralized access to data. Instead of pages the tool supports different types of work products. In our initial version of the tool the work products supported are User Story, Use Case, Class, UML Class Relation diagram, UML Sequence diagram, and Review.

After logging in the student chooses which project he is working on. On the project home page is a list of stories, along with links to associated use case and sequence diagrams. Going to a specific work product brings up an editor to modify that work product and/or to create/read reviews of the work product.

The interface consists of a set of Javascript widgets. Each Javascript widget provides a self-contained set of functionality. For instance, the review widget allows creation of a review for an arbitrary work product. The widget responds to button presses and other actions in its area through local changes and also via AJAX communications with the web server. Most of the widgets deal with a single work product and have three modes: edit, view, and compare. The edit mode creates buttons and fields necessary to edit the relevant data in the work product. The view mode displays the work product in a non-editable form, and the compare mode shows the differences between two versions of the same work product.

The user interaction screens are built up as a set of widgets to support reusability. Most of the widgets are utilized in multiple user screens. Another reason for this type of design is to create an architecture accessible for students to build their own extensions. Since widgets are self-contained we plan on making the creation of new widgets for the tool a project for future offerings of the course.

On the web server side, the tool is written in Perl and utilizes the Catalyst framework. The server has a Representational State Transfer (REST) architecture. The resources that the web server supports are the work products that we identified above. One additional resource is a comment resource. Comments can be attached to any work product and can be set as either public or private comments. Comments are used to provide guidance, hints, or other information that is not part of a review. We do not put student grades in any form in either comments or reviews. A Postgresql database is used to store all of the project information. The database maintains the different versions of the resources, timestamping each version of the resource. The database prevents data from accidental deletions by only allowing newer versions of resources to be created.

The only work products outside the scope of the web application are source code and unit testing. Integrated development environments, such as Eclipse, provide enough useful features (e.g., refactoring) that it is worthwhile to use a separate tool for the source code. Most of our students already have some experience with code development in Eclipse.
SUMMARY

Agile methods work well for student projects in an introductory software engineering course. The work products associated with agile processes are easier for students to get started with and seem to be better at keeping the students engaged in the project. There are many excellent tools for software engineering, the problem with most of them for an introduction to software engineering course is that they do not span all of the work products, there is a learning curve associated with each that saps momentum from the project, and they do not support instructors well in providing quick feedback to the students. The new tool that we have created addresses many of these concerns, and our initial experience with it has been positive. We plan on continuing to use and develop this tool in future offerings of the course. Further information regarding the tool can be found at http://www.cs.plattsburgh.edu/webse/

REFERENCES


TEACHING AN ITERATIVE APPROACH WITH ROTATING GROUPS IN AN UNDERGRADUATE SOFTWARE ENGINEERING COURSE*

Karen Anewalt, Jennifer A. Polack-Wahl
Department of Computer Science
University of Mary Washington
Fredericksburg, VA 22401
anewalt@umw.edu, japwahl@gmail.com

ABSTRACT

In order to provide a realistic software development experience, the University of Mary Washington Computer Science Department offers a one semester project based software engineering course. We use realistic projects to help the students learn the entire software development process. However, during the period of 2004 - 2008 two issues became apparent. First, students were discouraged because of the failure to complete projects, even though they were relatively small in nature. Second, viable clients began decreasing since word had gotten out that students were unlikely to deliver working products at the end of the semester. In this paper we report on how we changed the course to use the iterative software process model and simultaneously changed our team management technique to require student teams to change membership over the semester. The original focus of this research was to investigate methods to increase the quantity of projects completed and increase student satisfaction at the end of the semester. This paper presents a combination of old and new research. The older portion of the research, creating dynamic teams, was implemented in the academic year of 2008-2009 and results were previously published. Combining dynamic teams and using iterative software development was used in academic year of 2009-2010, background and results are presented in this paper. Overall, both changes improved the client experience and added a touch of reality in the software engineering course.

* Copyright © 2010 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.
Furthermore, by combining dynamic teams and the modified iterative approach the project completion rate for the course increased from 30% to 80%.

BACKGROUND

Much of the motivation for the course revision was sparked by the need for greater industry relevance and increased student motivation. The traditional method of teaching software engineering emphasizes up-front planning and following a linear path through development [4]. While this allows the instructor to cover the traditional academic software engineering topics, it does not lend itself to the real world professional approach. The approach also made it difficult for students to prioritize client needs and as a result, few clients received working software at the end of the course. We decided to apply a different software process to the course and selected the iterative approach.

The iterative approach has been used by a variety of universities. At the Milwaukee at of Engineering, iterative development of a classroom example and an object-oriented process based on two commercial software tools to improve the teaching of software engineering [10]. However, students typically did not have an opportunity to apply the lessons learned from the class example to a real world project [10]. More recently at the Wilhelm-Schickard Institut für Informatik used an iterative approach in a graduate-level, two semester course successfully. As a result of using the iterative approach the students became more confident about the expected outcomes as they completed each iteration [3].

Another example of iterative software development is extreme programming. At Victoria University of Wellington extreme programming was integrated into an existing document-centric software project course [7]. Using extreme programming had a positive effect overall, for the students and the clients. Students said there was no substitute for actual project work and that the work was rewarding, such that they were able to produce working code whereas in the past this was not the case. Clients were pleased that the code produced was more aligned to their needs and they got a working product in every case. The one downside was that some clients wanted more documentation than extreme programming practices produced.

While these are only a few examples of using the iterative approach in a software engineering course, they show that overall this approach can provide a positive experience for both students and clients. The iterative approach permits students to learn from previous iterations and incorporate previous experience and feedback into the next iteration.

In addition to using a new software process model, we also incorporated a new group management method: dynamic groups. We were introduced to the concept of dynamic group composition by Donald Schwartz at the 2008 International Conference on Frontiers in Education: Computer Science and Computer Engineering. He explained how he used rotating groups in his software engineering course and how using this approach improved the quality of documentation created by the teams [9].
COURSE DESCRIPTION

The course described in this paper is an undergraduate single semester software engineering course taught in a computer science department at a small liberal arts institution. The course is a 14-week, semester long course and meets for a two-hour session twice a week. The students enrolled in the course are computer science majors in their junior or senior year. The course was offered using the rotating group approach in the fall 2008, spring 2009, and fall 2009 semesters. Enrollments in chronological order were 17, 20, and 15 students. In the fall of 2009 the modified iterative approach was fully integrated into the course offering.

In the course, students apply the software lifecycle concepts to a group project for a real-world client. Most of the clients are faculty from other departments on campus, although occasionally the client is associated with a non-profit organization in the greater community. During the fall of 2008 and spring of 2009 the students walked through a traditional software lifecycle with the projects and complete a requirements document, project plan and schedule, design document, and test plan. The students’ course grades are heavily influenced by both their ability to create the software documents and implement functional software that meets their clients’ needs.

During the fall of 2009 the students walked through a modified iterative approach to the software lifecycle. Class time was used to teach the concepts necessary to create software iteratively and apply the techniques discussed in class to their current projects. At the beginning of the semester two hours were used to discuss the methodology and then students spent two hours applying the principles to their current project and presenting the status of their projects to the class and clients. Grades were determined by assessing documentation, the team’s ability to meet responsibilities, success in implementing functioning software, presentation of the software, and an exam.

MODIFIED ITERATIVE APPROACH PRACTICE

Many instructors dismiss the iterative approach because of the time constraints of a single semester course. We also struggled with the issue of how to make “good” iteration lengths within a 14 week semester. Different methods have been tried, such as two-week iteration cycles, three-week iterations, and 4 - 7 short iterations. Using the iterative method has many pros and cons.

Instructors have noted the following positive outcomes of using an iterative approach:

• Increased learning and success of students work [7],
• Ability to give students feedback about their design and implementation quickly [6],
• Opportunities to practice new skills while interactively developing new software [4],
• Increased student motivation [4].

Instructors noted the following negative outcomes when using an iterative approach:

• Students using iterative development with documentation development tried to avoid documenting and postponed documentation until the end of the project [5].
The amount of work an instructor must do by filling in for missing clients and providing continuous feedback is increased [10].

The instructor must make sure the projects selected are development-ready and ideas are concrete in nature [5].

Students do not get enough done in two weeks to actually call it iteration [6].

Based on this research, during the fall of 2009 we used a floating iteration period based on how much course material was covered in a four hour time span during the week. This led to three iterations. At each iteration and the preliminary stage the groups memberships were changed.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Weeks</th>
<th>Material &amp; Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preliminary 1-2</td>
<td>Introduction, Project Plan, Gathering Requirements, Validating Requirements</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9-11</td>
<td>Version Control, Implementation, Testing Document, Mini-Design of reworked requirements, Presentation to Class and Client</td>
</tr>
<tr>
<td>3</td>
<td>12-14</td>
<td>Implementation, Maintenance, Testing, Mini-Design of Lower Priority Requirements, Weekly Presentations to Class and Client</td>
</tr>
<tr>
<td>Release</td>
<td>15</td>
<td>Final Presentation and Release of Software</td>
</tr>
</tbody>
</table>

Traditionally we have used Ian Sommerville’s *Software Engineering* book [11] for this course. However, this text did not support the iterative approach well. Therefore, we switched to *Head First Software Development* by Dan Pilone and Russ Miles [8]. The book quickly but effectively leads the student through the software development process, but it also assumes that students have worked with UML and Object Oriented Analysis and Design. In our course, all but three students did have this background. Students unfamiliar with these topics were able to assimilate the assumed knowledge quickly through in-class practice and peer learning.

During the first week of the class the students were grouped and presented a simplified requirements document. During the first couple of weeks the students learned about interacting with clients, developing and expanding requirements, project planning, and the iterative approach. Within a short period of time the groups met with their client, prioritized requirements and completed a combined requirements specification and project plan document.

Following the requirement specification student learned about designing and the associated documentation. All in class activities were designed to go over concepts learned and also incorporate practical application of concepts using the current group projects. By week six the design document of the high priority requirements was completed, and implementation was started. Class time was used to implement their current projects, discuss and develop testing documentation, and talk about and use version control software. During the second and third iteration mini-design and test plan
documents were created but not graded. During the later iterations students interacted with the clients on a daily basis, modified code based on client feedback and added the lower priority requirements.

**USING DYNAMIC GROUPS**

Schwartz introduced us to dynamic group composition, a method for rotating student group membership. He used this approach in his software engineering course and described how it improved the quality of documentation created by the teams [9]. The participants are moved from project to project to simulate what happens in a real world software development environment.

During each semester the course used 5-6 unique projects for real-world clients. Students rotated from project to project and the composition of the various groups also changed. As a result, students had the opportunity to work on at least three different projects and with 10-16 peers during the software development lifecycle.

At all phases of the software lifecycle, the groups were instructor assigned and were told who the team lead was during that iteration. This allowed students to be grouped so that each project had a known leader associated with it. The groups were created initial based on a job application and input from faculty members who knew the participants. The rotations during the semester were based on three factors: one team member had to stay on the current project, the others members must not have worked on the new project before and what skills did each new member bring to the team. By distributing talent in this way all groups had the potential for success at each phase.

After each phase of the software development process, students completed an anonymous evaluation of their group members. These assessments were tabulated and provided to students only at the end of the semester. The assessments included:

- what the students contributed themselves,
- what the students would do different in the next iteration,
- did all members contribute equally,
- did the group work well together,
- were disagreement settled quickly and politely,
- did group members meet deadlines,
- were you encourage by group members to work on the project, and
- would you like to work with your group members again.

**RESULTS**

The number one benefit of using dynamic groups was that it provided students with an opportunity to follow another person’s design, edit and manage code created by another, and test/debug unfamiliar code. This was the same with and without using the iterative model.
Results using just dynamic teams from the instructor’s viewpoint there were fewer group breakdowns when team members underperformed, students received practice in reading other people’s work, and students felt more comfortable in rating peers honestly. From the students’ viewpoint, changing groups was a positive experience and they were challenged to adapt to multiple personalities and skill sets. These results were the same when we combined the iteration with rotating groups.

When combining dynamic teams and the modified iterative approach the project completion rate for the course increased from 30% to 80%. All the projects were from other departments located on the UMW campus and were hosted on BlueHost. The projects included a submission form and database for the UMW Institutional Review Board (IRB), a recruit tracking database for UMW Women’s Soccer Team, a UMW tour guide scheduler and database, an IT equipment tracking database, and a social networking website for UMW Business Department.

In all projects, only the highest priority requirements selected by the client were completed by teams which allowed students to feel confident and satisfied with their contributions. Because the high priority requirements were the foundations of the desired software, their completion made the software usable for clients. Client satisfaction was higher using the iterative approach.

In the fall of 2009 all projects completed their highest priority tasks and we believe this was directly related to the iterative approach. The IRB software successfully completed user registration, login, creating board members, submit application, upload files with application, assign application to board member, view assigned application, approve/deny/request revision application, and request board review. The Soccer software successfully completed user registration, login, view/add/search/authorize/delete recruits, an administrator feed of recent recruit information upon logging in, a virtual notebook for each recruit and automatic email notification of activity of recruits. The Tour Guide software successfully completed creating tours, automatically creating semester tour guide schedule, approve/deny/update tour guide application and a CVS dump. The IT software successfully completed a front end that allowed general user to search for a classroom that fulfills their technology needs, a backend that allows the administrative to manage and organize the school’s technological equipment and supplies, as well as the spaces the equipment and supplies are located in. Finally, the Business Social Network registered and confirmed users, created a forum for alumni and students to make connections, post/search/delete job vacancies and resumes, search users, and creating an administrator. Grade included one F and then the rest was evenly distributed in the A and B range.

Even with these requirements all clients wanted more. The IRB project that was delivered but did not have any ability to archive or delete processed applications. While it works for now, once they have one or two years of applications the list of applications would be numbered over 200 and viewing would be congested. The IRB has decided to continue using it and is looking for a future student to continue work on the software. The tour guide software does not automatically indicate a backup person if the primary tour guide is unavailable. The admissions office decided to use the product and they will schedule backup tour guides by hand. Students have continued to work on three of the
projects during the spring 2010 semester as individual study projects. The software for IT, Soccer, and the Business Department are being upgraded to add more features.

Overall, both changes improved the client experience and added a touch of reality in the software engineering course. Students had a positive real world experience and the clients were extremely excited about their projects outcomes.

REFERENCES


INSTRUCTIONAL ASPECTS OF STUDENT PARTICIPATION IN HUMANITARIAN FREE AND OPEN SOURCE SOFTWARE

PANEL DISCUSSION

Contact Person:
Heidi J. C. Ellis, Associate Professor and Chair,
Department of Computer Science and Information Technology
Western New England College
1215 Wilbraham Rd.
Springfield, MA 01119
413-782-1748
hellis@wnec.edu

Ralph Morelli, Professor
Department of Computer Science
Trinity College
300 Summit St.
Hartford, CT 06106
860-297-2220
ram@trincoll.edu

Gregory W. Hislop, Associate Professor
College of Information Science and Technology
and Department of Computer Science
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104-2875
215-895-2179
hislop@drexel.edu

Norman Danner, Associate Professor
Department of Mathematics and Computer Science
Wesleyan University, Science Tower 655
265 Church St.
Middletown, CT 06549-0128
860-685-2185
ndanner@wesleyan.edu

ABSTRACT

Active participation in Free and Open Source Software (FOSS) projects can provide students with large-scale collaborative software development experience. Frequently these experiences include interacting with an international group of professionals. The ability to participate in an active project empowers and motivates students to learn. FOSS projects of a humanitarian nature further motivate students by providing students with the satisfaction of improving the human condition in some manner. Thus Humanitarian FOSS (HFOSS) provides a fertile environment for student learning.

* Copyright is held by the author/owner.
However, faculty members may be hesitant to become involved with an active HFOSS project for a variety of reasons including the learning curve for the project and/or development environment, dearth of technical support, lack of clear entry path into projects, unclear deliverables, lack of assignments and grading rubrics, concern for the quality of student work, and more. Many faculty members express concern about balancing the flexibility required for real-world development with the need for assessment items to fulfill academic requirements such as learning objectives.

This panel will present the experiences of four faculty members who have taught one or more courses that have involved students in HFOSS projects. Each faculty member will briefly present \( a \) one HFOSS teaching experience, \( b \) a particular learning activity for students, and \( c \) one roadblock they had to address. Five minutes will be allocated for brief questions after each panelist’s presentation. At least 20 minutes will be allocated at the end of the panel for audience questions and participation. It is hoped that the audience will both add to the collective knowledge on teaching using HFOSS by sharing their experiences as well as benefiting from the information presented.

**PANELISTS**

Heidi Ellis will discuss the use of software engineering templates for deliverables for students involved in HFOSS projects. The main issue to be addressed is structuring gradable deliverables including rubrics.

Ralph Morelli will describe introducing both majors and non-majors to HFOSS principles. The focus will be on a software development course and activities modeled on a typical FOSS development community and process.

Greg Hislop will present an approach to helping students gain a basic understanding of HFOSS tools and processes. The discussion will address the issue of introducing students to the HFOSS development environment.

Norman Danner will discuss approaches for involving students with a range of experience and backgrounds in complex HFOSS projects. The discussion will provide perspective both from the standpoint of interacting with the project administratively and ameliorating the student learning curve.

**BACKGROUNDs**

**Heidi Ellis** was one of the initial participants in HFOSS efforts at Trinity College and a founding member of the HFOSS effort. She is PI on the NSF SoftHum project which focuses on course-level details for supporting student open source experience within the classroom. She has experience in involving software engineering students in HFOSS projects.

**Ralph Morelli** has led the development of the HFOSS project and is PI on multiple NSF multi-institution CPATH grants that fund the HFOSS effort. Dr. Morelli is also a co-PI on the NSF SoftHum project. He has involved introductory and advanced students in HFOSS projects within courses and in summer internships.
**Gregory Hislop** is a co-PI on the NSF SoftHum project and is leading an effort to expand active student involvement in open source software at Drexel University. He is exploring open source participation in the context of introductory computing and upper level software engineering courses.

**Norman Danner** is a co-PI and PI on the NSF HFOSS CPATH grants and has been involved in the HFOSS effort for several years. He has supervised summer HFOSS interns and in his most recent software development course, students contributed to an existing major humanitarian FOSS project.
THREAEAS EARLY∗

TUTORIAL PRESENTATION

John Beidler
Department of Computing Sciences
University of Scranton
Scranton, PA 18510
beidler@scranton.edu
570.941.7774

Yaodong Bi
Department of Computing Sciences
University of Scranton
Scranton, PA 18510
bi@cs.scranton.edu
570.941.7774

ABSTRACT

The goal of this tutorial is to present an approach and supporting artifacts for incorporating concurrent programming in the CS 2 and CS 3 courses employing Java Threads. This work represents a three-year effort to upgrade the CS 2 and CS 3 courses using an approach that balances objects and algorithms. The approach employs several Java Threads course modules that can be added to these courses by bringing Java threads into these courses by integrating thread-based material into the existing course material.

INTRODUCTION

Threads are hard, or so we are told. Threads are hard if they are introduced in a cavalier fashion and the potential pitfalls are not pointed out to the students from the beginning. If left to their own devices, students will quickly get frustrated with threads as their code gets caught in inadvertent thread deadlocks. Examples must include software that appears to be logically well constructed but have inadvertent deadlocks during execution. This tutorial is presented in three parts:

1. Thread Guidelines: Note, these are guidelines, not commandments. Threads programming does not require following these guidelines, but our experience has demonstrated that these guidelines keep CS 2 and CS 3 students out of trouble.

2. The Course Modules: We present four modules, two for CS 2 and two for CS 3, that incorporate the applications of threads into topics that are normally covered in these two courses.
3. The Artifacts: The artifacts range from simple producer-consumer models, to two person (three threads) game models viewed as 2-producer-1-consumer models, to a (multiple threads) framework for tackling graph and digraph problems using threads. These artifacts also include some innocent looking pieces of software that demonstrate the frustrations of inadvertent deadlocks.

THREAD GUIDELINES

Keep in mind that the goal of these guidelines is to get students started and keep them out of trouble, especially inadvertent deadlock. The first part of the tutorial presents the guidelines and the reason for each guideline. These guidelines are reinforced through the course modules and artifacts, explained below. The guidelines are:

i. Judicious use of synchronized methods. We emphasize that when a developer employs a synchronized method they are employing a locking mechanism. As such the use of locks should be well focused with strongly cohesive methods, methods whose code has a single purpose, to achieve the goal that required the application of a lock, no more, no less.

ii. Shared data/information objects. If information must be shared between threads, construct classes that contain the shared information and have the threads access and modify the information through, possibly synchronized methods. [We’ll explain the “possibly” in the tutorial.]

iii. Separate the Threads. Clarify thread construction, starting, and interaction. We encourage the separation construction of threads, not having one thread as an attribute (composed) within another thread. Sometimes, this requires a little extra effort to handle inter-thread communications. But the result leads to avoiding some potential deadlock issues.

iv. Get the Boolean flags right. One flag = one purpose! Don’t try to do too much with one flag.

v. Assertion test the wait/notify conditions: Build assert statements using DeMorgan’s Laws to verify the correctness of thread execution conditions. Since many wait conditions are compound conditions sometimes it is easier to work backwards and build the thread execution condition,

\[ A \text{ or } B, \]

then use Demorgan’s Laws to construct the wait condition,

\[ \text{not}(A \text{ or } B) = (\text{not } A) \text{ and } (\text{not } B). \]

MODULES

Module 1 (CS 2)

Module 1 focuses on introducing threads that are employed in the IteratorFactory, an abstract class that assists software developers in constructing Iterators over arbitrary containers. Students employ the IteratorFactory to construct Iterators over various containers (lists, stacks, queue, trees, graphs, etc.). Afterwards they are introduced to the
internal structure of the IteratorFactory, which follows the standard producer-consumer model. As part of the review of the abstract class, the thread guidelines are introduced.

**Module 2 (CS 2)**

Module 2 looks deeper into Java threads. We cycle back and reintroduce the basic producer-consumer model. This time the model is enhanced by looking at the construction of two-person games using three threads - two player threads and an umpire thread corresponding to two producers (the players) and one consumer (the umpire). A collection of artifacts are employed. The artifacts focus on playing the game called Rock-Paper-Scissors.

**Module 3(CS 3)**

Module 3 reintroduces both the IteratorFactory and Rock-Paper-Scissors. The IteratorFactory is reintroduced as an assignment – the replacement of the simple mailbox and flag with a finite queue, a typical real-world producer-consumer model enhancement.

Rock-Paper-Scissors is reintroduced with what appear to be a simper set of Java classes. However, when these artifacts are employed the software frequently hangs in deadlock. This provides an opportunity to explore deadlocks and the role of the Thread Guidelines in avoiding deadlock.

**Module 4(CS 3)**

A hashtable-based framework is introduced to tackle graph and digraph problems. Two classical problems are presented, the Border-Crossing problem and the Four Color problem. The students are introduced to the classical queue based solution to the Border Crossing problem and an exhaustive stack, or recursive, based solution to the Four Color problem. Threads are introduced into the hashtable based framework and both problems are reintroduced with thread based solutions. In the process, either the Four Color problem or the Border Crossing problem is given as an assignment. It should be noted that the thread based four color solution serves to create a lot of interest in threads because of the speed with which it will home in on a correct map coloring.

**ARTIFACTS**

The last part of the tutorial reviews the artifact sets. There are three sets of artifacts. The instructor has some flexibility in presenting the artifacts and determining what they wish to reveal and what they wish to leave for assignments. There are three collections of artifacts, the IteratorFactory artifacts, the Rock-Paper-Scissors artifacts, and the graph/digraph artifacts. As one might expect, the IteratorFactory is the simplest. It has two variants, a simple flag/mailbox implementation and a finite queue model.

The Rock-Paper-Scissors artifact set is the most complex in that it contains a working model, a model that raises deadlock, and an incomplete variant that can be used as a framework for assignments. Similar assignments can be constructed based on other two-player and umpire games.
Some care must be taken when deploying the Graph/Digraph artifacts. Specifically, the instructor must decide which elements will be presented in class and which elements will be reserved for student assignments. We expect to see these artifacts expanded as we gain more experience employing threads to resolve other graph problems. All tutorial artifacts are available at,

http://www.cs.scranton.edu/~beidler/threads/

ABOUT THE PRESENTERS

Drs. Beidler and Bi have cooperated on numerous projects and have authored several joint papers. Combined, they have over fifty years experience teaching Computer Science courses. Dr. Beidler’s research has been in fields of Computer Theory, Data Structures and Algorithms, and Web Programming. Dr. Bi research interests have been in the areas of real-time programming, operating systems, and databases. In the past few years, along with several other members of the department, they have been experimenting with ways of introducing new material into their CMPS 144 and CMPS 240 courses, which somewhat parallel what most people would refer to as CS 2 and CS 3. This tutorial presents our approach and supporting artifacts for introducing threads in CS 2 and CS 3.
BANTAM JAVA COMPILER PROJECT: EXPERIENCES AND EXTENSIONS

Marc L. Corliss
Mathematics and Computer Science Dept.
Hobart and William Smith Colleges
Geneva, NY 14456
corliss@hws.edu

Joshua Davis
Mathematics and Computer Science Dept.
Hobart and William Smith Colleges
Geneva, NY 14456
joshua.davis1@hws.edu

David Furcy
Computer Science Department
University of Wisconsin Oshkosh
Oshkosh, WI 54901-8643
furcyd@uwosh.edu

Lori Pietraszek
Electronic Systems
Lockheed Martin
Syracuse, NY 13202
lori.a.pietraszek@lmco.com

ABSTRACT
A compiler course is a critical component in an undergraduate computer science education. In this paper, we discuss our experiences in teaching a compiler course using the Bantam Java compiler project, which enables students to write a Java-like compiler. Both instructors and students have found Bantam Java effective in their courses. We describe several significant extensions to the Bantam Java toolset, which give instructors more flexibility and also lower their overhead when using it in a course. These improvements include an extended source language, a Java Virtual Machine target, an Ant-based build process, and optimization and interpreter assignments.

1 Some of this work was done while Lori Pietraszek was a student at Hobart and William Smith Colleges.

* Copyright © 2010 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.
INTRODUCTION

A compiler course can have a tremendous educational impact on an undergraduate computer science student. A compiler course, especially one with a significant implementation project, improves the programming skills of the students in the course. In writing a compiler, students implement the source programming language and thus gain a deeper understanding of it, as well as other similar languages. If this language is used frequently in other courses and professional settings, then student programmers will benefit greatly from such a compiler course. As Hall et al. point out [14], "programmers with a solid background in compilers tend to excel in their profession." Indeed, in a project-based compiler course, students gain considerable software development practice, including team-work skills. A compiler is large and complex. Developing a significant piece of software and managing its complexity is a valuable experience for students.

A compiler course also helps students better understand computer systems. They see concretely how high-level programs such as those they have written in other courses get mapped into low-level machine language; the compiler course demystifies the gap between high-level programs and the low-level machine. Students also gain some insight into the impact of language design and implementation choices (e.g., type safety) on the security of computer systems.

One problem in teaching compilers is finding a suitable language and compiler infrastructure that enables students to implement their own compiler in the classroom. In this paper, we describe our experiences with the Bantam Java classroom language and compiler toolset [13]. We have found (anecdotally, at least) this course effective at helping students better understand programming languages and computer systems, and become better software developers.

In addition, we describe several significant extensions we have made to the Bantam Java toolset, which increase its scope, add instructor flexibility, and make it easier for instructors to incorporate the toolset into their own classroom. These extensions include the following:

- **Extended language** with more language features (e.g., arrays) for use in ambitious courses.
- **JVM target** for building a compiler that generates code for the Java Virtual Machine (JVM).
- **Optimizer** component and assignment for teaching code optimization.
- **Interpreter** component and assignment for teaching program interpretation.
- **Alternate build process** for building the toolset via Ant [4], in addition to Make [3], enabling students and instructors to install the toolset on non-Unix platforms such as Windows.

BANTAM JAVA COMPILER PROJECT

The Bantam Java compiler project [13] was designed specifically for the classroom, and in particular, for use in a compiler course. The project is well documented. It includes a comprehensive student lab manual [12] (which has been updated to include the
extensions from this work) as well as the API for the compiler generated from javadoc [19]. This documentation, along with source code to be completed by the student and solution code for instructors, is freely available at the web site:

http://www.bantamjava.com/

The source language, Bantam Java, is a significant subset of Java, a language used in many computer science curricula. The language emphasizes object-orientation, since most modern languages are object-oriented (e.g., C++, C#, Ruby), and students are less familiar with these features and their implementation.

Although there exist other educational compiler projects, most of them are either significantly different from commercial languages [7] or not object-oriented [8, 21]. The MiniJava project [9] does not have these drawbacks. However, its documentation is tightly integrated within a textbook, which reduces instructor overhead but also instructor flexibility. Furthermore, some of the publisher-provided solution code for MiniJava is incomplete. Finally, object orientation is treated as an optional add-on rather than a fundamental feature of the source language.

One problem with any educational compiler source language (and Bantam Java is no exception) is that the language must be small to ensure the compiler is implementable in a single semester. Thus, Bantam Java cannot contain all of the features of Java. Some important features like arrays had to be left out of Bantam Java. One contribution of this work is to create an extended language that contains some of the omitted features.

A virtue of the Bantam Java Compiler Project is that it includes several customizable components. For example, the instructor can choose between two sets of lexer/parser generators (JLex [10]/Java Cup [15] or JavaCC [18]). The instructor can also choose between two targets: MIPS/SPIM and x86/Linux. Furthermore, the (free) lab manual can be used in concert with any traditional compiler textbook [6, 9, 11]. One contribution of this work, is the inclusion in the toolset of additional customizations, namely: the definition of an extended language, the support of a new target architecture (JVM), and the addition of optimization and interpreter components and assignments.

<table>
<thead>
<tr>
<th>Institution</th>
<th>Instructor</th>
<th>Semester</th>
<th>Lex/Parse Generator</th>
<th>Language Version</th>
<th>Target</th>
<th>Optional Assignments</th>
<th>Build Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chinese Univ. of Hong Kong</td>
<td>Kam-Wing Ng</td>
<td>Spring ’09</td>
<td>JavaCC</td>
<td>Base</td>
<td>MIPS</td>
<td>None</td>
<td>Make</td>
</tr>
<tr>
<td>Dublin City Univ.</td>
<td>Geoff Hamilton</td>
<td>Fall ’08</td>
<td>JavaCC</td>
<td>Base</td>
<td>MIPS</td>
<td>None</td>
<td>Make</td>
</tr>
<tr>
<td>Hobart &amp; William Smith Col.</td>
<td>Marc Corliss</td>
<td>Spring ’09</td>
<td>JLex/Cup</td>
<td>Base</td>
<td>MIPS</td>
<td>Optimizer</td>
<td>Make</td>
</tr>
<tr>
<td>SUNY Geneseo</td>
<td>Scott Russell</td>
<td>Fall ’09</td>
<td>JLex/Cup</td>
<td>Base</td>
<td>MIPS</td>
<td>None</td>
<td>Make</td>
</tr>
<tr>
<td>Univ. of Wisconsin Oshkosh</td>
<td>David Furcy</td>
<td>Fall ’08, 09</td>
<td>JavaCC</td>
<td>Base, Extended</td>
<td>JVM</td>
<td>None</td>
<td>Make, Ant</td>
</tr>
</tbody>
</table>
Table 1: Institutions that have used the Bantam Java Compiler Project along with some of the customizations they have used.

<table>
<thead>
<tr>
<th>Wofford Col</th>
<th>David Sykes</th>
<th>Fall '08</th>
<th>JLex/Cup</th>
<th>Extended</th>
<th>MIPS</th>
<th>None</th>
<th>Make</th>
</tr>
</thead>
</table>

EXPERIENCES WITH BANTAM JAVA

During the academic year since the Bantam Java toolset was first released [13], it has been used at six institutions worldwide (see Table 1). While one objective of the extensions presented in this paper is to make Bantam Java even more attractive to potential adopters, the feedback we have received so far from both instructors and students has been overwhelmingly positive.

Instructor feedback. There is evidence that the current adopters are satisfied with the toolset. In fact, all of the instructors who have used the toolset plan on using it again. The fact that one of them (Hamilton) has used the toolset in a two-course sequence demonstrates the flexibility that our toolset affords instructors. Furthermore, one of these instructors was so enthused about this project that he has become a full-fledged member of the project (and a co-author on this paper!). Here is some of the feedback we have received from two instructors who are not collaborators:

- I found the set of projects to be reasonably challenging, but gave a very good grounding in the art of compiler writing.
- I couldn't have done the course as well as I did without using Bantam.

The second instructor also points out the virtue of a project-based compiler course:

- [Students] spoke so highly of the course and how it helped them to tie a lot of other courses together (including theory of computation, computer architecture, and programming languages) that we are adding it to our catalog.

In fact, all of the instructors who have used the Bantam Java project had favorable comments.

There were also some minor negative comments. For instance, one instructor, who used the extended language, found arrays to be too challenging for students. He thus removed arrays from the code generation assignment. He also thought that the parser solution code (only accessible to instructors) was hard to follow. We plan to address these criticisms in the next version of the toolset. But overall, this instructor was happy with Bantam Java.

Student feedback. Since "students [are] not that excited about implementing someone else's toy language" [5] and "tend to believe that languages designed by their instructors are not 'real'" [20], we were looking forward to students' reactions to the experience of building a compiler for (a subset of) Java, currently one of the most relevant programming languages around. We were rewarded with quite a few positive student
comments (these come exclusively from the courses taught by two of the co-authors), such as this one:

<table>
<thead>
<tr>
<th>Component</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Base, Extended*</td>
</tr>
<tr>
<td>Target</td>
<td>MIPS, x86, JVM*</td>
</tr>
<tr>
<td>Lexer/Parser Generators</td>
<td>JLex/Java Cup, JavaCC</td>
</tr>
<tr>
<td>Optional Advanced Assignments</td>
<td>Optimizer*, Interpreter*</td>
</tr>
<tr>
<td>Build Process</td>
<td>Make-based, Ant-based*</td>
</tr>
</tbody>
</table>

Table 2: Bantam Java toolset customizable components. * indicates aspects added in this work.

I found that [I] have gained a better understanding of the semantics of Bantam Java and Java programming languages because I had to actually go through each part and think about it carefully.

Several students also recognized the inherent benefit of completing the programming project. Here are quotes from two of them:

The projects taught me a lot about building a compiler hands on that I could not have figured out from just learning the theory.

These projects have definitely helped in understanding the semantic analyzer and code generator. Actually, I don't think I would really understand either of them well at all without doing these projects.

The overall tenor of student feedback is best summarized by this other student comment:

I found this project to be challenging and interesting [...] however [...] it became more of a fun challenge than an obstacle.

EXTENSIONS TO THE TOOLSET

In addition to using Bantam Java successfully in our courses, we have made several significant extensions to the toolset, which will provide instructors with even greater flexibility while lessening the overhead of incorporating Bantam Java into the classroom. Table 2 shows the aspects of the project that can now be customized. The starred options represent extensions made available with this work. We discuss these options below. Table 1 shows how each institution currently using Bantam Java has applied (or not applied) these customizations.

Extended language. We have developed an extended language for use in more ambitious, longer, or graduate-level courses. This language includes all of the features of the original (or base) language plus arrays, for loops, break statements, increment/decrement operators, and arbitrary placement of return statements. With these
extra features, the extended language is significantly closer to full Java. Although these features also add a significant amount of difficulty to the implementation of the compiler, students may find it more compelling to implement a language closer to Java. Instructors may now choose between these two versions of Bantam Java, based on the level of their students and the length of the course. The extended language has been used in two courses (at University of Wisconsin Oshkosh and Wofford College).

**JVM target.** Following up on the recommendation at SIGCSE 2006 by Waite [20], we have also added a new target for the compiler (in addition to MIPS/SPIM and x86/Linux): the Java Virtual Machine (JVM). An obvious benefit from this extension is that students can now use and implement the same target as in industrial-strength Java compilers [1, 2]. When targeting the JVM, the compiler generates text-based Java bytecodes (or Java assembly code), which can be converted into Java class files via the open source tool Jasmin [17, 16]. The Bantam Java JVM target has been used in one course already (at University of Wisconsin Oshkosh) with positive feedback.

**Optimizer component.** We have developed an optimization component and an accompanying assignment, which enables students to build their own optimizer. In the assignment, students translate the abstract syntax tree (the main program representation in the Bantam Java compiler) into control flow graphs, which are then optimized. Students must perform several optimizations on the program, including common-subexpression elimination, in which students must implement a data-flow analysis, and dynamic dispatch removal (converting unambiguous dynamic dispatches into static calls). This optimization component was used in one course (at Hobart and William Smith Colleges) with positive feedback.

**Interpreter component.** We have also developed an interpreter component for interpreting (rather than compiling) Bantam Java programs and an accompanying assignment, which enables students to build their own interpreter. This assignment could be used in addition to the other compiler projects or it could replace the code generation assignment in time-pressed courses, since the code generation assignment is much longer than the interpreter assignment.

**Ant-based build process.** We have added an alternate way of building the Bantam Java toolset. Previously, the toolset was built via Make [3]. Now it can alternatively be built using the standard Java build tool, Ant [4]. This makes it possible to install and run the Bantam Java compiler on non-Unix platforms such as Windows (e.g., it has been installed and used on Windows XP). It may also make it easier for some students to install the toolset on Unix systems. With this addition, the toolset can now be installed and used on all current major platforms, namely machines running the Windows, *nix, or Mac OS X operating systems.
FUTURE WORK

We will pursue future work in several directions. First, we plan to conduct a study exploring quantitatively the importance of a project-based compiler course in the undergraduate computer science curriculum. We hope to gather data from several institutions to measure the impact of a project-based compiler course on students' programming skills and their understanding of compilers and computer systems.

We also plan to add optional advanced assignments, such as a garbage collector written in C. Furthermore, we will design a third version of the Bantam language. As some instructors have already given us feedback that the extended language seems too challenging for their course, we will create a language that sits between the base and extended languages in terms of its feature set. Both the additional language and the additional garbage collection assignment will give instructors even more flexibility in designing their compiler courses.

CONCLUSIONS

In this paper, we describe our experiences using the Bantam Java compiler project in the classroom. Overall, the instructors and students who have used this project have found that it greatly enhanced their compiler course. Finally, we extended the project in several ways to improve its effectiveness. We added an extended language, a JVM target, an Ant-based build process, and optimization and interpreter components and assignments.

ACKNOWLEDGMENTS

Joshua Davis and Lori Pietraszek were supported by student grants from the Provost's Office at Hobart and William Smith Colleges. The authors thank the reviewers for their helpful feedback.

REFERENCES

[1] The javac group. URL: http://openjdk.java.net/groups/compiler/.


SEMANTIC HOOKS USING A MINI-LANGUAGE FOR A

COMPILER PROJECT

John S. Mallozzi
Iona College, New Rochelle NY 10801
914 633–2565
jmallozzi@iona.edu

ABSTRACT

In presenting semantic specifications to a parser generator, numerous
connections and “hooks” must be presented, relating parsing and semantic
code. This paper describes, in the context of a particular toolset developed by
the author, a mini-language that can be used to specify these “hooks” in a way
that allows the user to concentrate on each aspect of compiler generation
separately.

INTRODUCTION

A course in compiler design generally combines treatment of the theory with a term-
long project. Fitting these two together coherently can be a problem. The student must
get the project underway as soon as possible, but the theory underlying the code needed
takes a good deal of time to develop. Most of us are not free to demand an entire
additional course to satisfy our needs, since there are plenty of competitors. This paper
will consider some tools that aid the process within a single three-credit semester.

The easiest way to assign a non-trivial project within a one-semester course that
must cover theory and other aspects of compiler design is to use a parser generator, which
allows the student to concentrate on semantic code-writing instead of the tedious – albeit
easy – details of applying parsing theory to actual code production and, more to the point,
waiting until that theory is covered in class to produce any code. To be useful, this
generator should accept specifications that include “hooks” to semantic code.

Often, one sees a quick, recursive descent compiler for a very simple language, a
brief discussion of the desirability of automation and then a quick switch to bottom-up
tools such as lex-yacc (and their successors), followed by a project using these tools. Some years ago, the author decided to frame the term project using top-down tools, and, eventually, to concentrate on recursive descent in particular. There are many reasons why one might think this approach best. Some of them, and some tools developed to support the top-down approach, were described in [3]. The purpose of this paper is to explicate more recent work, which makes the provision of semantic “hooks” easier, and in particular describes a mini-language to express connections between parsing and semantic processing. Use of this mini-language in compiler specifications appears (anecdotally) to be beneficial in completing term projects. Formal study of this apparent benefit would be appropriate in future course offerings.

DESCRIBING A LANGUAGE

The foremost practical advantage of bottom-up over top-down parsing methods is that the former has access to more “natural” grammars (“LR” grammars) than the latter (“LL” grammars). (There is also a significant theoretical advantage: LR grammars are more inclusive than LL grammars and have nicer properties. However, the focus here is on more practical matters.) The common example of expressions (see [1] and [2], for instance) makes the point nicely. An LR grammar for (simplified) expressions is shown in Figure 1.

```
expression → expression + term
expression → term
term → term * factor
term → factor
factor → identifier | number
```

Figure 1

This description cannot be matched by as clean an LL grammar, because left-recursion is not allowed. One winds up instead with a grammar like the one in Figure 2.

```
expression → term moreTerms
moreTerms → + term moreTerms
moreTerms → ε
term → factor moreFactors
moreFactors → * term moreFactors
moreFactors → ε
factor → identifier | number
```

Figure 2
However, an actual context-free grammar (CFG), is not necessary to describe a language. Instead, Extended-Backus-Naur Form (EBNF) works very well, both intuitively and to lead the development of a recursive-descent compiler. Left recursion must still be avoided, but it is essentially a non-issue, since EBNF allows repetition, avoiding much of the recursion. Thus, we have the EBNF for expressions in Figure 3, in which the Kleene star (*) indicates zero or more repetitions and parentheses are used for grouping.

\[
\begin{align*}
expression & \rightarrow term \ (\ + \ term \ )^* \\
term & \rightarrow factor \ (\ * \ factor \ )^* \\
factor & \rightarrow \text{identifier} \ | \ \text{number}
\end{align*}
\]

Figure 3

This last approach is arguably simplest and most transparent of all, and the theoretical need for a CFG is not present if the project is written from EBNF using recursive descent. The path from EBNF to recursive descent is quite smooth, and can be automated. Certainly there are available tools to do this for all three approaches. Here, a set of tools developed by the author is described, with emphasis on semantic “hooks,” especially as they involve the mini-language, which is also described. The ease with which this mini-language can be used within the EBNF is the reason the author prefers this approach.

A JAVA PACKAGE FOR COMPILER BUILDING

As described in [3], students are provided with a Java package, compiler, which contains a number of interfaces and classes that can be used to develop a compiler. These types are organized into several hierarchies. Through the extension and use of these hierarchies, students develop the semantic part of the compiler in an organized, object-oriented manner, which fits nicely with the recursive-descent approach: the compiler itself is an instance variable of a parser class, and instances of a semantic records class and its subclasses provide the arguments and return values for the parsing and semantic methods.

First of all, there is a class for tokens, along with subclasses. Objects of this type are returned by the generated scanner. The parser requests tokens from the scanner and performs syntax analysis in the usual way, by matching the tokens that appear with those specified by the language productions. Semantic “hooks” placed into the EBNF productions make use of two additional hierarchies. One consists of any kind of object one might want to store in a symbol table. Of greater interest here is a base class, SemanticRecord, for a hierarchy through which inter-method communication can be accomplished. This communication may be between parser and semantic methods, or between semantic methods themselves. Some subclasses are provided in the package, but students may extend this hierarchy as needed.

The student interacts with these tools in two ways. First, he or she must provide a file containing an EBNF specification of the language, including semantic “hooks.” Using this file, the tools generate a skeleton compiler, containing the complete scanner
and parser along with stubs for semantic routines in a separate semantics class, with all interfaces to the parser supplied. Working within a helpful IDE, such as Eclipse, it is easy to note places where the student must insert code to complete the compiler.

A mini-language is employed in writing the specification file. Use of semantic actions, as described for example in [2], is not unusual, except that here the actions are methods which may include parameters and return values. As the student writes the specification, the greatest difficulty encountered is in getting correct the interfaces among several important components:

1. The parser recognizes tokens, using a one-symbol lookahead. One often must capture the currently recognized token, to use it in semantic routines. However, this token must be wrapped in an appropriate semantic record type.

2. Parsing routines, such as the one for an expression, frequently must return a value to be used in semantic processing, which once again must be declared to be of an appropriate type.

3. Semantic routines must use tokens and values from the parser and values from other semantic routines to accomplish their tasks, and may return values to be used in other routines.

The idea behind the tools described here is to allow the student to specify all needed interfaces within the EBNF productions for the language, after which the tools will set up the compiler in such a way that he or she may concentrate on one semantic routine at a time.

MINI-LANGUAGE FOR SEMANTIC SPECIFICATIONS

To specify terminals and nonterminals, EBNF is used. The tools require that all symbols used be declared (rather than, for example, requiring that nonterminals be enclosed within angle brackets). Special symbols such as punctuation may be declared to have a name or else placed between single quotes in the productions. The use of declarations also provides a place to specify any return value for a nonterminal parsing method. This return value must have base type SemanticRecord. The student may use one of the supplied subclasses or design a new one. At present, no parameters are permitted in the specification of nonterminal methods (though allowing them would prove especially useful in processing errors).

A semantic action method may accept parameters and/or return a value, all having base type SemanticRecord. Each action method must be declared – again obviating the need for any special notation while providing a place to declare the parameters and return values. Each action is placed within a production, at an appropriate place for execution (as particular advantage of the top-down approach). Many of the actions are very simple matters of transferring information, making it tedious to design an action method for each such simple transfer. For this reason, a small amount of code may be placed, along with actions, within a production. The actions and all the code are contained within curly brackets, to separate them from the EBNF. This action code constitutes the mini-language of primary concern in this paper. The principal novelty is the concept of an action ID, as described below:
An actionID is one of the following:
1. An identifier previously assigned a value.
2. The symbol $tok$, which stands for the token most recently recognized by the parser.
3. The symbol $val$, which represents the result of the most recently applied nonterminal or action method.

Action code consists of a sequence of action statements between curly brackets to specify connections between the parser and the semantics. The necessary code to connect the two is then generated automatically, with the special symbols defined above translated into appropriately typed variables.

The allowable action statements take several forms:
1. action (args);
   where action has been declared as a semantic method and args are its arguments, which may be empty, but otherwise must be a comma-separated list of actionIDs.
2. identifier = action (args);
   where identifier is a new variable, local to the production in which it occurs, and action (args) is as before, but now returns a value. Here and elsewhere, identifier is not declared; it will automatically be declared with the appropriate type in the generated code, avoiding another common source of student error.
3. identifier = actionID; which is just an assignment, but serves to introduce identifier.
4. $ret identifier; , which results in the value of value of identifier being returned as the result of the nonterminal within whose production this action code appears. This statement must be the last in a given action code sequence.

**EXAMPLE**

Assuming that the nonterminal for an expression has been declared to return a value, and that other symbols have been declared appropriately, Figure 4 contains productions for the assignment statement and expression nonterminals.

```plaintext
assignment -> identifier {lhs = $tok; checkVariable(lhs);} ' :=' expression ';' { assign(lhs, $val); } %
```

```plaintext
expression -> term {x = $val; } (addop {op = $tok;}
    term { x = calculate(op, x, $val); } )% {$ret x;} %
term -> etc.
```

**Figure 4**

In the first production’s action code, once the identifier on the left side of the assignment symbol is recognized, it is saved in a variable, lhs, for later use. In actuality, the current token is wrapped in a semantic record. After this, a semantic method is to be called to check that the variable has been declared. After the expression on the right-hand side has
been parsed, an action method is to be called to generate code for the assignment. It takes as parameters the variable holding the earlier-saved token and the value returned by the most recently called action or nonterminal method, in this case the one that parses expression. (The symbol % is used to indicate the end of a production.) Similarly, the action code for an expression saves the value returned from parsing a term, remembers which addition-type operator was seen, and then uses these two (x and op) and the value of the next appearing term for use in a semantic method that generates code for the actual calculation. After all repetitions, the current value (actually a record for where the value is stored) is returned as the result of parsing the expression. Note that control, such as repetition, is generated automatically.

The parser code generated from the specification of assignments is in Figure 5. Note how details such as declarations and casting have been handled automatically. The student has no need to work with this code, although reading it may be beneficial for understanding the parsing process.

```java
match(identifier_TOKEN);
lhs = current.wrap();
semantics.checkVariable((IDRecord)lhs);
match(ASSIGNMENT_TOKEN);
ValueRecord resultOf_expression1_0 = expression();
match(SEMICOLON_TOKEN);
rhs = resultOf_expression1_0;
semantics.assign(((IDRecord)lhs, (ValueRecord)rhs);

Figure 5
```

The code generated for semantic method assign, with an indication of where to insert code, is in Figure 6. It consists of just a stub, which remains to be coded by the student. However, interfaces and details of type are automated, so that focus can be on that coding task.

```java
//>> BEGIN assign <<//
void assign(IDRecord lhs, ValueRecord rhs)
{
    tracer.print("APPLYING SEMANTIC ACTION assign");
    //******** EDITABLE SECTION -- CODE FOR ACTION ROUTINE **********/
    // TODO: REPLACE TEMPORARY STUB CODE BELOW WITH CODE FOR assign
    if (!tracer.isTracing()) // TEMPORARY
        System.out.println("SEMANTIC ACTION assign"); // TEMPORARY
    //*************** END OF EDITABLE SECTION ***************/
}
//>> END assign <<//

Figure 6
```

In all of this, the student may focus on what must be done at each stage, not on how to fit it in with all the auto-generated and other student-written code. Once the skeleton
compiler is generated, the student will find stubs for each semantic method, which he or
she can then consider separately from all the rest of the compiler, using the parameters
sent and the value to be returned, and any local variables desired. Very few truly global
variables are needed, and these generally accompany less structured statements, e.g., the
location of the current loop if break statements are allowed in the language.

CONCLUSION

If the student uses EBNF with action code to specify a language compiler, properly
written tools generate a particularly effective skeleton compiler, which permits the
student to concentrate separately on specifications and the writing of semantic code. This
is done within a simple framework – EBNF and Java inheritance – not calling upon much
of the theory of compiler design or the use of parse stacks, semantic stacks, etc. As this
theory is developed, and while the compiler is being written, much can be said about how
the tools are written, using the theory, thus satisfying the curiosity of students and tying
all together by the time the course ends. However, the project itself can be started early,
with a minimum of theoretical preparation. Tools described in this article may be
obtained via anonymous FTP from the author, at

REFERENCES

[1] Aho, Alfred, Lam, Monica, Sethi, Ravi, and Ullman, Jeffrey, Compilers:

[2] Fischer, Charles, Cytron, Ron, and LeBlanc, Richard, Crafting a Compiler,
Addison-Wesley, Boston, MA: 2010.

LEVERAGING HARDWARE DESCRIPTION LANGUAGES
AND SPIRAL LEARNING IN AN INTRODUCTORY COMPUTER
ARCHITECTURE COURSE

John Robinson, Ganesh R. Baliga
Computer Science Department
Rowan University, Glassboro, NJ
(856) 256-4805
robinsonj@rowan.edu, baliga@rowan.edu

ABSTRACT
This paper describes our experience with using hardware description languages (HDLs such as VHDL) in the sophomore level Computer Architecture course for Computer Science (CS) majors. In our approach, we leverage the students’ understanding of Object Oriented Programming (OOP) to introduce programmatic specifications of computer components such as multiplexors, adders etc. Students will be exposed to HDLs yet again in more advanced hardware courses such as Digital Design, Computer Architecture and Embedded Systems Programming. In this manner, we plan to harvest the benefits of spiral learning, whose effectiveness in pedagogy is well accepted. The proliferation of Fully Programmable Gate Arrays (FPGAs) and modern development environments makes it feasible for CS majors to experience the creation of functioning hardware using specifications coded in VHDL. We expect our approach will result in improved learning outcomes for CS majors in the hardware field.

INTRODUCTION
As part of a general revitalization of the curricula associated with undergraduate computer architecture, we proposed a project to integrate modern digital hardware design technology and build stronger integration among hardware courses for CS majors that
have been traditionally taught in isolation [2]. The ties between the courses will be based on spiral learning techniques of redundancy and increasing complexity [6]. A hardware description language, in particular Very High Speed Integrated Circuits Hardware Description Language (VHDL), was used to introduce recurring topics iteratively in a constructive way. Iterative reintroduction using VHDL over the entire hardware curriculum will allow the students to leverage the knowledge gained as a CS major and apply it to understanding computer architecture and the software that drives the hardware. We expect that such an iterative approach reinforces understanding since it allows the students to build upon their prior knowledge and, furthermore, to approach the very same concepts from different perspectives and increasing complexity. This iterative method is inspired by the idea of spiral learning [6] which builds on a constructivist view on learning. It assumes that learning is an active process in which learners construct new ideas or concepts based upon their prior knowledge. A curriculum can support this process if it revisits its basic ideas repeatedly.

Traditionally, a computer hardware curriculum teaches theoretical knowledge and employs simulation software to run projects. The revitalization project not only focused on introducing and integrating new digital design technology into the curricula, but also, moving the curricula from a theoretical /simulation based focus to a project based learning environment and providing continuity over the entire hardware course sequence. The impetus for this initiative is to improve, expand, and facilitate student learning about digital design, computer circuits and hardware, while capitalizing on the students’ existing strengths in Object Oriented Programming (OOP) techniques. Our approach will include using the VHSIC Hardware Description Language (VHDL) language, Field Programmable Gate Arrays (FPGAs), and CAD software from Aldec and Xilinx.

Current trends in industry [11] demand that CS graduates have a fluency in modern digital systems design techniques and become agile product development and rapid prototyping team members. There is also a convergence of hardware design methodologies (Software Based) currently used by Computer Scientists and Electrical Engineers. Modern digital design techniques (Software Based) are essential to both CS and Electrical Engineering curricula [14,15,16]. This provides a unique opportunity for CS educators to produce graduates that are comfortable working in a realm traditionally reserved for Electrical Engineers and demands that Electrical Engineering curricula become more software centric [1,3,7,8,9,11,13]. This novel approach to digital systems design will allow the students to apply the knowledge gained as a CS major to the design and understanding of computer architecture.

Leveraging what CS students learn and employing a spiral curriculum will facilitate the common threads pulled through the entire CS hardware curriculum. This paper describes the experience of using VHDL in an introductory computer architecture course.

THE NEW CURRICULUM FOR COMPUTER ORGANIZATION

Traditional CS OOP methodologies, project driven learning, and visualization techniques will be utilized as a novel approach to allow the “natural” exploration of the details of implementation for the “black box” components introduced in computer organization. Projects and lectures in this course allow the students to apply their CS knowledge in understanding computer hardware components and the software that allows
communication between the components. An introduction to VHDL and FPGAs will allow the students to examine the details of implementation and experiment with the “black box” components taught in Computer Organization. A specific mini MIPS processor will be used in this course and will be presented over the entire hardware course sequence. Computer Organization (our sophomore level Computer Architecture course) will introduce the architectural features of this processor.

The novel curriculum for teaching computer architecture is presented to the students using VHDL and concept sets. These concept sets are tools for learning about digital design and modern design techniques. The concept sets aim to help all CS undergraduate students achieve the following targeted learning outcomes:

- Move from a theoretical to a project based learning environment using state-of-the-art techniques for digital design, e.g. HDLs, FPGAs, and computer simulation.
- Design hardware using traditional CS design techniques and modeling.
- Better integration of courses over the entire computer hardware curricula.
- Deeper appreciation of the boundaries between hardware and software.
- Student skills and knowledge that is transferable to subsequent laboratory courses and future careers.

**USING VHDL TO TEACH COMPUTER ORGANIZATION**

In the Fall semester of 2009, a new approach to teaching Computer Organization was piloted. The course moved from a purely theoretical/simulation format to a combination of theoretical/simulation and project based learning utilizing the concept set and VHDL. The concept set is a series of projects that explore the hardware of a microprocessor using VHDL and OOP. New software was also introduced to provide access to modern simulation and development tools.

Traditionally, Computer Organization is taught using microprocessor models and the accompanying instruction sets, simulation software, and assembly programming. The course serves as an introductory experience to the concepts of computer architecture. In this version of the course, a MIPS processor was introduced along with VHDL as a way to illustrate the details of the components that comprised the processor as outlined by our concept sets. The MIPS processor was deconstructed and each component (multiplexers, registers, ALUs, etc..) was described in VHDL and discussed. The processor was reconstructed as a controller and datapath. The instruction set for the processor was also explored and simulated using PCSpim [12] and PathSim [10]. The similarities between OOP techniques, high level languages and VHDL were emphasized and simulations of the VHDL code running on physical hardware were shown using software from Aldec and Xilinx. The final concept set project had the students implement a Fibonacci number generator machine in VHDL using the algorithm, state diagram, and the datapath shown in Figures 1, 2 and 4 [17]. The expected VHDL code, showing the code for the loop driving the datapath, is shown in Figure 3. A balance between theory/simulation and hardware design was sought using the concept sets and VHDL. The effort to revitalize the hardware curriculum seeks to utilize this balance over the entire course sequence.
if (input = 0)
    output = 0;
else if (input = 1)
    output = 1;
else {
    F1 = 0;
    F2 = 1;
    while (input >= 2) {
        output = F1 + F2;
        F1 = F2;
        F2 = output;
        input = input – 1;}
}

Figure 1 – Fibonacci Algorithm

```
architecture behv of FSM is
  -- define enumerated states
  type states is(idle, init, doLoop, done);
  -- declare state signal
  signal nState, cState : states;

  process(start, comp_result, cState)
  begin
    case cState is
      when idle =>
        -- code to drive components in datapath
        done
      when init =>
        -- code to drive components in datapath
      when doLoop =>
        N_Sel <= '0';
        N.ld <= '1';
        F1_Sel <= '1';
        F2_Sel <= '1';
        F1_ld <= '1';
        F2_ld <= '1';
        Out_enb <= '0';
        if comp_result = '1' then
          N_Sel <= '0';
        end
```
N_Id <= '0';
F1_Sel <= '0';
F2_Sel <= '0';
F1_Id <= '0';
F2_Id <= '0';
nState <= done;
else
nState <= doLoop;   
end if;
when done =>
   -- code to drive components in datapath
end case;

Figure 3 – Fibonacci Controller VHDL Code

In this project students were introduced to the complexities of implementing a controller for a small application specific processor. Concepts such as state diagrams, enumerated types, decisions, and switch statements allowed the students to implement the controller by leveraging knowledge acquired in other CS courses.

The use of VHDL allowed the exploration of the components and subsystems. The use of VHDL is “natural” to CS majors because it mimics OOP techniques where the details of implementation are typically ignored and complexity is managed with the use of classes and objects. The complexities of the individual components are hidden or “encapsulated” in the VHDL code allowing larger systems to be built by drawing simple schematics to connect these components. The topics presented in this version of the
course were the first level of the *spiral curriculum* [6] model for the hardware courses. The concepts set will be revisited in each of the subsequent hardware courses (Principles of Digital Computers, Digital Design Lab, Embedded Systems, and Advanced Computer Architecture) of the CS major. Each exploration of the concepts will be at a higher level of complexity than prior courses, reinforcing what was introduced in latter.

**SUMMARY AND CONCLUSIONS**

These days CS majors are taught software design using techniques featuring Object Oriented Programming, code reuse, rapid prototyping, project centered learning, and visualization. In addition, they are taught the theory, components, and design of computer hardware. Traditionally electrical engineers did hardware design, but with the advent of FPGA and HDLs, computer scientists can also design hardware. There is a convergence of hardware design methodologies used by electrical engineers and computer scientists. This provides a unique opportunity for CS educators to produce graduates that are comfortable working in a realm traditionally reserved for electrical engineers. A learning experience featuring *spiral learning* [6] and a methodology that is “natural” to CS majors will allow them to develop a deeper understanding of digital systems. This novel approach to digital systems design will allow the students to apply the knowledge gained as a CS major to the design and understanding of computer architecture. The curriculum described in this paper will enable students to bridge the gap between hardware and software design to produce a unified view that enables the co-development of systems using both hardware and software.

**REFERENCES**


[8] Hamrita, T. K., Potter, W. D., Bishop, B., Robotics, Microcontroller, and Embedded Systems Education Initiatives at the University of Georgia, An


WIDENING THE K-12 PIPELINE AT A CRITICAL JUNCTURE

WITH FLASH™

Michael Sands, James Evans, Glenn David Blank
Lehigh University
Department of Computer Science and Engineering
29 Memorial Drive West
Bethlehem, PA 18015
1-610-758-4085
mjs207@lehigh.edu, jfe208@lehigh.edu, glenn.blank@lehigh.edu

ABSTRACT

Students at a young age acquire many negative stereotypes regarding computer programming and computer science in general. These stereotypes discourage students, especially young women and minorities, from pursuing an education in computer science. Using an innovative curriculum focused on Adobe Flash™ we try to dispel these stereotypes and encourage students to pursue a computer science education. Adobe Flash provides a platform to demonstrate that computer science can be a fun and interesting topic that provides many unique and engaging challenges. This allows us to bring these students back into the pipeline of potential computer scientists.

Categories and Subject Descriptors

K.3.2 [Computer and Information Science Education]: Computer Science Education, Curricula, K-12 Education

General Terms Experimentation, Human Factors

Keywords

Computer science education, introductory courses, misconception, women, minorities

Copyright © 2010 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

Many well known stereotypes about computer science are misconceptions. For example: —Computer science is only for nerds and geeks, —Computer science is only for men, and —Computer science means sitting by yourself in a cubicle writing code. These stereotypes, well known in society, are picked up by students at a young age, and reinforced by the attitudes and actions of peers.

These stereotypes discourage many students from pursuing an education in computer science. The problem is especially acute among young women and students from underrepresented minorities. The number of computer science degrees awarded to women declined from 35.8% to 22% between 1984 and 2002 (Robb 2003). The trend has worsened with overall declines in CS enrollments. Computing Research News reports that —interest in CS among women fell 80 percent between 1998 and 2004, and 93 percent since its peak in 1982 (Vegso 2005). As Camp (1999) observes, —the computing community cannot sit back and assume that as the numbers of students rises, the percentage of women students will automatically rise and that the ___[incredible shrinking pipeline] will take care of itself.‘ We must take direct action to attract and retain more women to computing at all points in the pipeline (i.e., K-12, undergraduate, graduate, faculty and industry). Indeed, the pipeline shrinks even before female students get to high school. One high school in our region reports having just one girl out of 125 students in IT elective courses. Besides the well known and documented problem of decreasing enrollments in the face of increasing demand for IT professionals, the narrow pipeline will likely stifle the development innovative and fresh ideas that could be explored in the field.

We have attempted to overcome these stereotypes with interactive multimedia that explicitly seeks to overcome these misconceptions with video interviews of computing professionals (including women and minorities), animation and interactive exercise (Blank, Hiestand and Wei 2004). Though the results were promising, there is still a need for an approach that incorporates a hands-on problem solving experience that introduces novices to the excitement of programming without bogging them down with syntactic details of difficult concepts. Our goal is to create curricula that dispel these stereotypes at a critical juncture, just as students are entering high school, when they are entering high school and deciding about electives and career goals. Using multimedia and the Adobe Flash programming environment we aim to show students that computer science is a fun and exciting field that involves team work and problem solving. Once we get the students to this point and have broken the stereotypes we have also given the students the tools to form their own opinions about computer science. A version of this curriculum was developed as part of a three week summer program offered to at risk 8th and 9th grade students in the Launch-IT project (www.lehigh.edu/launchit). With promising results, we believed we could move this approach to a high school setting. Building on preliminary results reported in Sands, Moukhine and Blank 2008, this paper chronicles further developments that hold much promise.

LV STEM (Lehigh Valley Science, Technology, Engineering, and Mathematics) is an NSF GK-12 project which recruits graduate students to help develop innovative curricula in local schools (see www.lehigh.edu/stem). LV STEM puts graduate fellows in K-12 classrooms where they can lend a hand to teachers while sharing their cutting
edge knowledge with students who see them as successful role models. Graduate fellows, K-12 teachers, and professors work together to create novel curricula based on state of the art research and real world applications. The Flash outreach team is one of eight in the LV STEM program, two of which focus on computing. Rieksts and Blank (2008) describe a program for middle school students, by developing missions for rovers on a simulated Martian landscape; the remaining six teams are in other STEM fields, including mathematics, biology, chemistry, physics, and mechanical engineering. The lead and second authors of this paper are the graduate fellows for the Flash team and the third is the faculty team leader and Principal Investigator of LV STEM.

2. METHODOLOGY

2.1 Motivation

Adobe Flash is a powerful multimedia authoring tool for creating animation and interactivity in applications and web pages. Flash lets developers integrate efficient vector graphics, animation, sound, and video into —movies that can be played on any computer with the freely and widely available Adobe Flash plug-in. Flash also includes a rich programming language, ActionScript, that supports the development of dynamic and interactive programs. But what makes Flash especially attractive is the ability to create animated movies in a matter of seconds, without any coding in a programming language.

Flash is an appealing language for all age groups, genders and backgrounds. A novice can create vector-based graphics, finding Flash more versatile than something like Microsoft Paint. Animation is then relatively easy, by dragging a graphical object, setting keyframes, then letting Flash automatically interpolate the changes in intervening frames. Graphics and animations are thus the hook that attracts students, who naturally want to learn how to control their movies. Students are thus motivated to learn how to program in ActionScript in order to turn these movies into interactive applications, responding to button clicks, for example to stop and restart movies. Flash also lends itself to classroom use by letting students of differing abilities work on the same projects. This also provides the opportunity for more advanced students to help novice students with the project. This is an enjoyable experience for both students and reinforces the knowledge that they are learning. Students are also impressed that they can learn how to use a technology widely used on the web, including many of their favorite sites. As opposed to other approaches that seek to attract new students, such as microworld programming environments, computer games or robotics, we have found, in both the Launch-IT and LV STEM projects that Flash appeals to girls, boys, and students from underrepresented minorities. Flash is also appealing to high school instructors. The Microsoft Office User Systems (MOUS) course, required for all incoming high school students in Bethlehem, Pennsylvania, has consisted of learning how to use word processors and spreadsheets. Although these are important skills for the workplace and higher education, most students are bored. Since this course is the only required technology education most of these students get in high school, it tends just to reinforce negative stereotypes about the field. When we urged updating this course, instructors and school administrators were receptive to the idea, in part because of complaints from students and parents about the course, as well as their own concerns about declining enrollments in IT courses—in high school. Flash lets instructors demonstrate how to have fun with a powerful real world tool.
Inserting a few weeks of Flash into the MOUS course spurs interest of students in the field, including taking full term courses on Flash and other programming languages.

2.2 Curricula

Students were given projects to learn Flash and the basic concepts of programming. The basic structure of the curricula was the instructor would demonstrate the concepts to be learned for the project on the projector. A handout would then be given explaining what the students had to do for the project. The instructor would then move around the classroom helping the students with issues and providing guidance when needed. This format allowed the students the most time for hands-on experience with Flash. The first projects were aimed at teaching the students the workspace and the basics of drawing and animation. After these concepts were covered the rest of the curricula covered ActionScript and the basics of programming. For one of the projects students were asked to create a Sunrise/Sunset scene that had different control buttons. When one button was clicked it would cause the scene to play the sunrise animation. When the other button was clicked it would play the sunset animation. Through this lesson the students learned the concepts of event driven programming, conditionals, and variables (if it was already sunrise you can’t have sunrise again). In following project students learned user interaction through the keyboard, a project that involved moving a character through a world, and other basic programming concepts.

3. PREVIOUS WORK

The LV STEM project began working towards this goal in earnest during the 2007-2008 academic year. During this period a graduate fellow, Nick Moukhine, worked with two teachers to begin improving their technology education programs. In the first semester, using the trial version freely available from the Adobe site, we inserted one week of Flash in the MOUS course, quickly introducing graphics and animation. The positive results encouraged our team to expand to a two week curriculum in the second semester, giving students more time to be creative with animated projects. Using funds available from the PITA (Pennsylvania Infrastructure Technology Alliance), we partnered with the Bethlehem Area School District to acquire an educational site license of the Adobe Creative Suite for the school. A twoweek Flash course was run in four different sections of the MOUS course. Students were quizzed at the end of the course and asked about their interest in a full length Flash course. Of the 54 that responded, 28 (51.8%) expressed interest.

4. CURRENT WORK

4.1 Three Week Intro Flash Course

Expanding the two week curriculum to three weeks was crucial, in order to take students from animation to ActionScript. While graphics and animations is the hook, it is important that novices see how programming adds more power and interactivity to their movies. We would not want anyone signing up for a full Flash course without understanding that it would involve more ActionScript programming. The three week
curriculum covers the basics of the Flash workspace and timeline, drawing, animation (frame by frame and tweening), and the basics of ActionScript programming.

Each lesson begins with a demonstration of the topic from the instructor. Afterwards, the students are shown what they are going to be making, based on a handout showing the tools and steps to take to learn and apply the topic being taught. For the remainder of the class students work with Flash to create the project described in the handout. Students are encouraged to work with each other to solve problems and issues with Flash. During this time the graduate fellow and teacher walk around to assist students with any problems they might be having.

This course was taught in three sections of the MOUS course at Liberty High School during the fall semester. The results were so promising that it will be taught in every section of the MOUS course at Liberty in the spring semester, now that all teachers have been trained.

4.2 Flash TeacherWorkshop

In order to prepare all technology teachers at Liberty High School for teaching the three week Flash curriculum, the graduate fellow taught Flash in a six week workshop. It was offered as in-service training to Bethlehem teachers, thus fulfilling a district requirement. The workshop was also offered as a LVCSTA (Lehigh Valley Computer Science Teachers Association) event and was open to all teachers who registered for it. Two teachers represented Freedom, the other Bethlehem high school, and a high school teacher from Allentown also participated.

During the workshop the teachers were taught the layout and basics of the Flash workspace. They were taught all the tools used for drawing, how to do frame by frame animation, tweening, using symbols and the library, and ActionScript. The goal of this workshop was to prepare the teachers to not only teach the course but to have them able to handle the problems or issues that their students might encounter.

Note: other instructors and students interested in learning this material are encouraged to try a multimedia lesson available online at [www.cse.lehigh.edu/~cimel/prototype.html](http://www.cse.lehigh.edu/~cimel/prototype.html). Click on —Go To Multimedia Demo, then —The Universal Computer. A multimedia course, implemented in Flash, introducing Computer Science will start up. (This version requires Microsoft Internet Explorer as the browser.) Once in this title, select User Interfaces, then Flash.
tutorial, from the menu on the left. The tutorial demonstrates an earlier version of Flash with animated lessons within the environment and an instructor teaching in either audio or text mode. Visitors are also invited to check out the video interviews on misconceptions about Computer Science, in the first chapter, described in Blank, Wei and Hiestand 2004.

4.3 Full Length Flash Course

Buoyed by the formative results obtained during the 2007-2008 academic year high school teachers and administrators approached our team about developing a full length Flash course. Based on those results and the interest and backing of the teachers in the technology department, the school district approved the creation of two Flash courses to be offered starting in the 2009-2010 academic year at Liberty High School. The school district also approved changes at Freedom High School; an expansion of the MOUS program to incorporate Flash, and the addition of new Flash courses. The courses to be offered will each be half semester courses. A basic Flash programming course will be offered to the students as an 8 week course. This will be followed with an advanced Flash programming course that will also be 8 weeks in length. The basic curriculum for these courses has been laid out but it still a work in progress. We worked with teachers at Liberty to develop a curriculum that seeks to expand interest in computer science. The first new course will be a new introduction into the programming sequence, leading naturally into any of the languages already offered, Visual Basic, C++ or Java.

5. FORMATIVE RESULTS

5.1 Three-Week Flash Course Student Results

The Liberty High School population includes 33% from underrepresented minorities and 43% from low income families. At the end of the three week course students were given a survey. They were asked questions to gauge the amount of knowledge they learned about Flash as well as their interest in the full length Flash course.

The 32 students, 2 Female African-American, 11 Female Caucasian, 5 Female Hispanic, 3 Male African-American, 2 Male African-American/Hispanic, 4 Male Caucasian, 1 Male Caucasian/Hispanic, 2 Male Hispanic, 1 Male Pacific-Islander, 1 Male Other, were asked the following questions: —Before taking the 3-week Flash course how much did you know about Flash? and —After taking the 3-week Flash course how much did you know about Flash? As shown in Figure 1, all demographic groups reported significant learning. Of particular interest to us are the scores of the minority and female groups. For all males in a minority group there were significant gains in self-reported knowledge. Males in minority groups started with an average of 1.44 and reported a mean of 2.89 afterwards, a doubling in the amount of Flash they perceived that they learned. This increase also holds for females in minority groups, with a gain of 0.43, while Caucasian females reported a gain of 1.45. Though the gains are smaller for minority girls, they are still encouraging. These results suggest that high school students in all demographic groups are able to gain significant confidence about what they can learn about Flash animation and ActionScript programming in a short module of lessons and hands-on activities.
Students were also tested on their Flash knowledge. They were asked to provide step by step instructions for drawing and animating a ball across the screen. They had to accomplish the objective by writing the instructions such that someone not familiar with Flash would be able to follow them. The students were graded on 5 areas: Content, Process, Terminology, Organization, Clarity. They were rated on a scale from 0 to 4. 0 = Not Present, 1 = Needs Work, 2 = Satisfactory, 3 = Strong, 4 = Outstanding. Table 1 shows mean scores for all students.

These results show that in all five categories students achieved on average a score between satisfactory and strong. Since we can assume the students had little to no knowledge of Flash prior to this course, as the school does not teach it, these results show that their students gained knowledge and can explain what they know; these results corroborate their self evaluations of learning.

Students were also asked about their likelihood of taking a full length Flash course. Figure 2 shows the results. Especially noteworthy in these results are the promising results for minority groups, including male African-American, male African-American/Hispanic, male Caucasian/Hispanic, and female African-American demographic groups. These groups show a moderate to very high likelihood of taking the full length course. These results confirm the results that we found in the Launch-IT program for girls and students from minority groups and motivated the school and district to ask our team to offer full-length Flash course in the fall of 2009.

Students were also asked for comments for improving the three week curriculum. Some typical comments are quoted below:

“More Frequent”

“Do more stuff on it”

“I think the intro to Flash could be a little longer, but other than that what we did was very helpful and maybe I could use it in the future.”
Overall the comments provided were very positive reported earlier in Sands, Moukhine and Blank 2008.

### 5.2 Flash Teacher Workshop Results

At the end of the six weeks Flash teacher in-service training workshop, the eleven participating teachers were given a survey... They were asked to rate the Flash curriculum based on how it met standards, how interesting and engaging they thought it was, how the students would react to it, and if they planned on implementing it in their classrooms among other things.

As table 2 shows, overall the teachers strongly affirm that the Flash curriculum is a positive step to improving attitudes about computer science and technology and that it would be a fun and engaging way to get the students involved in the discipline.

### 5.3 Flash Course Results

The two Flash courses were introduced at Liberty High School this fall with resounding success. In total, 88 students are enrolled in the Flash courses in the Fall 2009 semester. This is over 1.5 times the number of students enrolled in all the other computer science course offered in the fall combined. Of these 88 students, 17% are girls, which exceeds the percent of girls enrolled in all other computer science courses at this high school.

The results for minorities are also promising. The Bethlehem Area School District does not release ethnicity information about students, so we surveyed the current two classes are being taught. Our sample was 55 students in the two Flash I courses. Of these 55 students, 27 of them identified themselves as belonging to a minority, or about 49.1% of the students surveyed. These results show that novel Flash curricula are indeed attracting the interest of females and minorities and have gotten them into the pipeline of computer science.

### 6. FUTURE WORK

Although our approach suggests a significant contribution to computer science education, there remain challenges for it to succeed more broadly. We have the following goals which we are attempting to complete this coming academic year:

1. Create and teach the curricula for the basic and advanced Flash courses at Liberty High School this fall.
2. Expand the curricula for the introductory MOUS course and two full Flash courses out to the other high schools in the Bethlehem school district. The project is underway by a new LV-STEM graduate fellow.

3. Disseminate all lessons, handouts, and media through the LV-STEM website (vlstem.cse.lehigh.edu). Full course curricula will become available after they are classroom tested in the upcoming months.

7. CONCLUSIONS

To widen the pipeline of new students entering computer science, we must overcome negative stereotypes about computer science that K-12 students learn and reinforce among their peers. Using Flash, with its popularity on the Web, relatively small learning curve, powerful animation and programming capabilities, we hoped to dispel these misconceptions and increase interest in the field. We especially hoped to increase interest among women and students from underrepresented minorities at a critical juncture, when they are entering high school and deciding about electives and career goals. Our results show that we did accomplish our goals for the program. We increased the knowledge about Flash and computer science especially in minorities and women. We also spurred interest in the full length Flash course which will lead to other computer science courses also in minorities and women. Our results show that we are creating student interest in the field of computer science. Our results also show, via the creation of the full length course and teaching of the three week course in all classes, that we have heightened awareness and willingness to modernize the computer programming curricula. Until we arrived, Bethlehem high schools had a sequence of Visual Basic, C++ and Java. This sequence apparently reflects the history of how these languages were introduced in the school, with Basic as the oldest and Java as the newest. But this sequence inhibits students from taking the Java, which is now the language of the College Board Advanced Placement test and the first programming language in many university sequences. Moreover, C# is increasingly important for job opportunities and also teaches the same concepts and skills as Java. We have convinced teachers and administrators to change this sequence to Flash/ActionScript, then any of the other programming languages, possibly adding C#. Bottom line, our new Flash curricula are attracting more and more diverse students into computing classes and at the same time motivates teachers and administrators as an improvement to their current curriculum.

An obvious drawback of Flash for K-12 education is its cost. We have found that once administrators see the advantages of Flash, they are willing to invest resources in site licenses. Partnerships that leverage state or federal funds can make all the difference. It would certainly also be desirable if our community could persuade Adobe to reduce its pricing structure in order to provide more access to students, especially in K-12.

Curricula and publications associated with the LV STEM project are available at lvstem.cse.lehigh.edu.
8. ACKNOWLEDGMENTS

LV STEM and Launch-IT are sponsored by NSF grant numbers 0639664 and 0624553, respectively. The Pennsylvania Infrastructure Technology Alliance (PITA) also provided supplementary funding for this project. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or PITA. The authors also express their appreciation to Liberty High School teachers Rebecca Wetzel and Kevin Jones, as well as to Dr. Henry Odi, Co-Director of the LV STEM and Launch-IT projects, for their invaluable help with this project. The lead author gives thanks to God for inspiration and the opportunity to make a difference.

9. REFERENCES


CONNECTING UNDERGRADUATE PROGRAMS TO HIGH SCHOOL STUDENTS: TEACHER WORKSHOPS ON COMPUTATIONAL THINKING AND COMPUTER SCIENCE

Patricia Morreale  
Department of Computer Science  
Kean University  
Union, NJ 07083  
908/737-3804  
pmorreal@kean.edu

David Joiner  
New Jersey Center for Science, Technology and Math Education  
Kean University  
Union, NJ 07083  
908/737-3427  
djoiner@kean.edu

George Chang  
Department of Computer Science  
Kean University  
Union, NJ 07083  
908/737-3797  
gchang@kean.edu

ABSTRACT

The high school preparation of future computer science undergraduates can be varied and depends strongly on the experiences and subject knowledge of their high school teachers. In an effort to connect undergraduate computer science programs to high schools for future student success in college, a public comprehensive university is hosting high school teacher workshops to help teachers enhance their teaching effectiveness. This has established a pathway from the university to the high school. Pre- and post-workshop surveys show that these teacher workshops have changed the perceptions high school teachers had regarding careers in computer science.

INTRODUCTION
High school students are commonly exposed to traditional math and science (Biology, Chemistry, Earth Science, and Physics) as part of their high school curriculum but exposure to computer science and associated computational thinking is frequently absent from the high school experience in the United States. One theory is that today’s classroom teachers are not aware of the materials available to them for use with students and are not aware of the professional opportunities available to their students upon graduation. While using computers may be part of the learning experience for students, problem solving and an understanding of what computer scientists, computational scientists, and information technologists do after college is not universally held among the teaching population in high schools [1, 2].

The importance of computer science to the national curriculum has been addressed by some states with teacher certification programs [3]. This paper does not presume to address the large topic of K-12 teacher certification in computer science, but instead provides an experience report on university faculty interaction with high school teachers during which improving student preparation for university study in computing and computer science was the focus. By holding teacher workshops for high school teachers, a public comprehensive university updated and enhanced ideas current high school faculty had regarding applications of computers in the sciences and the utility of computational thinking in high school.

VENUE AND AUDIENCE

The decline in computer science (CS) majors in the United States is known [4, 5]. Between 2000 and 2007, enrollment in Computer Science by undergraduates in the United States declined 50%. More than half of the college students that initially declare a major in computer science change their majors prior to graduation and the majority of students stop studying computer science by the end of freshman year [6]. Over the years, a number of university outreach efforts have been used to attract students to the major. Faculty visits to high school classrooms, high school student visits to university campuses, and interaction with faculty, are often thought to be the ideal recruiting vehicle. In reality, for all but the most selective institutions, getting students excited about computing may encourage them to major in computer science or information technology – but not necessarily in the visiting faculty member’s department. The high school students may (probably) go elsewhere. A correspondence between university and college faculty visits to high school classrooms and increasing enrollment in the major at the home institution is non-existent. Furthermore, students move on and university faculty visits to the high school must be annual or bi-annual at the least to keep student awareness of computer science high.

An alternative paradigm was considered: what if the emphasis of university and college faculty moved from engaging high school students to engaging their teachers? High school teachers have the potential for tremendous impact on their students, and do not leave a high school with the 4-year predictability that students do. The importance of high school teachers in recruiting undergraduate computer science students has been recognized by others, and well-regarded multi-day regional residential workshops, such as those held by the CS4HS group over the past five years [7], demonstrate the utility of teacher workshops in enhancing teacher effectiveness in high school computer science.
education. However, the one-day effort presented here is more inclusive, as math, science, and computer science teachers from high schools are welcome, not just computer science teachers, and the one-day workshop has the potential to impact a larger group of teachers. Additionally, this one-day workshop allowed teachers who would otherwise not been able to travel away from home for a week of professional development to attend. Another benefit is that one-day workshops can be and are presented more than once a year, with new workshop talks and presentations added to subsequent offerings.

As a result of this thinking, and in coordination with the College of Education, a one-day workshop for high school math and science teachers was developed, offering teachers continuing education units (CEUs) if they attended. By holding the workshop on the university campus, university students and facilities would be available for demonstration and discussion. This would offer the greatest opportunity for high school teachers to ask questions and exchange ideas in a manner which might not be possible in their home school and district. With the support of the Computer Science Teachers Association (CSTA) [8] and the National Center for Women and Information Technology (NCWIT) [9], a workshop proposal was developed and vetted by local high school faculty and CSTA chapter members. Once consensus was reached, the event was promoted and registration via a website [10] was conducted.

ENVIRONMENT

Nationally, discussion regarding CS preparation at the high school level usually considers AP Computer Science test preparation and results as the strongest indicator of student success at the college level in computer science. While this remains true, there are many students who are not exposed to computing or computational thinking in high school despite personal interest and aptitude, and unless they are quickly located on a university or college campus may not be able to master the skills needed in the time provided for university success in computer science. Due to these factors, by the time students get to post-secondary education, it may already be too late for future success in computing.

Therefore, the goal of this workshop was to address the educational ramp from high school to university – the ramp which is the approach for students considering computer science or any discipline involving computational thinking. In light of NSF’s planned “10,000 school project” [11] which seeks to develop a new pre-AP secondary school curriculum in fundamental concepts of computing and computational thinking, with the goal of having this new curriculum taught in 10,000 schools, a professional development pipeline to current math, science, and computer science teachers is vital.

With the professional development model in mind, workshop notices were circulated to potential workshop participants through local CSTA chapter e-mails to members, graduate education classes, and the superintendent offices of regional school district offices. Additionally, an AP Computer Science list server received a posting from a CSTA member, which also reached another community of potential workshop participants. These approaches worked well, with the CSTA chapter e-mails being the most effective. The workshop notice was directed at high school math, science, and computer science teachers and the workshop participants came strongly from those areas,
with many teachers having primary responsibilities which included teaching computer science or mathematics. Physics, chemistry and biology teachers also attended.

**Workshop Presentations**

The workshop day started with a welcome. Two 90-minute sessions were offered in the morning. After lunch, three 90-minute sessions were offered. On occasion, popular morning offerings were repeated in the afternoon, so the participating teachers did not have scheduling conflicts. Usually, at least five topics were provided on the workshop registration site, in order to offer a selection. A sample of recent workshop presentation topics is provided in Table 1.

<table>
<thead>
<tr>
<th>Workshop Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational Concepts with Alice</td>
</tr>
<tr>
<td>Modeling Tools for Science and Mathematics</td>
</tr>
<tr>
<td>3-D Visualization</td>
</tr>
<tr>
<td>CS, IT: Computational Science and Your Students: Why Not?</td>
</tr>
<tr>
<td>CS Unplugged: Computing without a Computer</td>
</tr>
<tr>
<td>Preparing Students for University Success in Computing</td>
</tr>
<tr>
<td>Project Interactivate-toolkit for high school mathematics</td>
</tr>
<tr>
<td>Easy Java Simulations</td>
</tr>
</tbody>
</table>

Table 1. Representative Workshop Presentations

Future workshops will include sessions on peer-led team learning (PLTL), pair programming, and hosting and preparing students for student programming contests. Discussions between workshop sessions and over lunch have been very productive, with university faculty hearing about activities in the high schools, and high school faculty viewing university student research posters and demonstrations.

**Survey Results**

Pre-workshop surveys administered identified that the teachers attending the workshops usually have at least five years of teaching experience and many have more than fifteen years of experience. The majority are from public, suburban districts; a few are from private schools. Some C++ programming is taught to students, but the Java programming is taught by more teachers. Most participants had never attended a computer science or computational thinking workshop before. The high school teachers considered themselves good at programming, and spreadsheets, with lesser skill reported in database management and computer-based modeling. Their expectations for the workshop included networking opportunities, learning about computational thinking, and understanding more about professional opportunities in computer science for their students.

After attending the workshop, post-workshop surveys were completed. All teachers indicated that they felt much more comfortable advancing the use of computing and computational thinking in their classes – which ranged from computer science, to high school mathematics and science topics. The potential for infusion of computational science throughout their curriculum was a real success story from the day, as far as the
university faculty were concerned. Problem-solving techniques using computational tools, take-back curriculum materials, career opportunities for students, and preparing their students for success in computing at the university level were all additional positive outcomes from the workshop.

The presenting university faculty had been particularly concerned that the workshop participants have materials, including web sites and exercises, which could be used in the high school classrooms. This is a distinction of this effort and this emphasis shined through, as workshop participants were very grateful that demonstrations and lectures they had attended resulted in not only verbal ‘take-away’, but also classroom-ready hands-on materials. This was a factor in rating this event as “one of the most outstanding professional development events I’ve attended” as one participant stated. The modeling and visualization tools demonstrated seemed most suited for the computer science and mathematics teachers, although chemistry teachers indicated that they planned to use the modeling and visualization tools also.

Overall, the workshops have been very successful, with participants asking about future workshop dates and suggesting topics which might be added in the future. The importance of computational thinking and computer science to their students’ future was very clear to the participants, as well as the resources which were available to them in the form of local university faculty. Comments are listed in Table 2.

<table>
<thead>
<tr>
<th>I am more aware of advances in this field and career paths available to those with degrees in Computer Science and Information Technology.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I will encourage more students to consider computer science as a major.</td>
</tr>
<tr>
<td>Became aware of the term “computational thinking” and plan to emphasize it in my classes.</td>
</tr>
<tr>
<td>I learned that computing may be done on all levels and with minimal prerequisites.</td>
</tr>
</tbody>
</table>

Table 2. A Sample of Workshop Attendee Comments on the Post-Survey

The strongest endorsement has come from the repeated workshop participants. By varying the program with each workshop, prior participants are able to find new sessions to attend, and have indeed returned to succeeding workshops after their initial experience. Table 3 shows the most important part of the workshop – the change in perception of computer science as a major and the host university as a choice for their students.
Table 3. Comparison of high-school teacher attitudes pre- and post-workshop

<table>
<thead>
<tr>
<th>I would recommend Computer Science, Computational Science, or Information Technology to my students as a career.</th>
<th>Pre-Workshop</th>
<th>Post-Workshop</th>
<th>Percent Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>45%</td>
<td>72%</td>
<td>27% increase</td>
<td></td>
</tr>
<tr>
<td>I would recommend my students consider attending the Workshop Host University for college.</td>
<td>33%</td>
<td>63%</td>
<td>30% increase</td>
</tr>
</tbody>
</table>

LESSONS LEARNED

Schedule the workshop early in the school year. High school teachers are busy during late May or June, with advanced placement examinations, final exams, moving-up ceremonies, proms, and graduation. Our first workshop was held in June, which resulted in a smaller turnout than expected. Subsequent workshops have been held during the fall, with greater attendance.

Work with your prospective audience when developing the workshop agenda. Local CSTA chapters are of invaluable assistance in reviewing drafts.

Consider whom you’d like to invite—and make sure you do. Targeted outreach to magnet schools and alumni can be very effective. E-mail specific professional communities you would like to include.

Think off-campus. Consider inviting speakers from outside the university who may be collaborators or employers of your students. This will provide interesting perspectives, and real-world stories, which are a strong complement to the workshop agenda. Our efforts have included several national and international speakers, which have been very well received by our participants, who appreciated the expertise which was shared with them, both from the ‘locals’ and the ‘out-of-town’ people.

Do a post-mortem. Follow-up with your participants and within the workshop team to see what additional ideas and insight may have developed. The team meeting should be held shortly after the workshop concludes. Plan on following up with your workshop participants by building a mailing list of math, science, and computer science high school teachers in your region which can be used to further develop post-event networking and communication.

SUMMARY

The workshop has become a regular campus event, with both high school teachers and university faculty looking forward to it and thinking about what might be useful to include in the workshop agenda. Current university computer science and information technology majors enjoy volunteering at the event and answering questions about their own preparation prior to university and their experience once on campus. This has served to personalize the ‘computer science student’ to the high school teachers in such a way
that they can share with their students, when they return to their high school, what a computer science major really works on and how he or she got to that point.

The effectiveness of the workshop in the high school community is rated highly as increasing awareness of opportunities in computing and information technology at the local university. Teachers openly indicate that they are now suggesting to their students that they consider majoring in computer science or information technology and consider the local university. With first-hand knowledge of the faculty, resources, and facilities, this recommendation is of the highest priority. A tracking project is underway to identify students recruited to the major as a result of their teacher’s workshop experience as well as to determine how frequently high school teachers are able to work with the materials shared with them during the workshop.

REFERENCES


[7] [www.cs.cmu.edu/cs4hs/](http://www.cs.cmu.edu/cs4hs/) _Explorations in Computer Science for High School Teachers_

[8] [www.csta.org](http://www.csta.org) _Computer Science Teachers Association_

[9] [www.ncwit.org](http://www.ncwit.org) _National Center for Women and Information Technology_

[10] [www.kean.edu/~cssc](http://www.kean.edu/~cssc)

RECRUITMENT OF CS MAJORS THROUGH A
NON-PROGRAMMER’S PROGRAMMING CONTEST

Mary Anne L. Egan
Computer Science Department
Siena College
515 Loudon Rd.
Loudonville, NY 12211
518 782-6546
maegan@siena.edu

ABSTRACT
Declining enrollments in computer science are a cause of great concern. There
has been a 30% decline in enrollments in computer science bachelor
programs over this decade and more than a 50% decline in the enrollment of
women in computer science [1]. This paper describes a unique way of
attracting excellent students to the major by inviting students and their teachers
to a day of exploration and competition hosted by our college. The program
has demonstrated an increased awareness of computer science among the
students and faculty who attend the day-long event. By including teachers and
guidance counselors in the day’s events, we hope to give them a better
understanding of computer science curricula and careers to bring back to their
high schools.

INTRODUCTION
Student IMPACT (Students Interested in Mathematics and Problem-solving
unAware of Computing Talent) is a one day program in career exploration and
competition for Computer Science and Information Systems (CS/IS). The program is
designed to provide high school students with demonstrated academic talent, especially
in mathematics and problem-solving, an opportunity to explore some of the ways college
students and professionals think and work in CS/IS. This opportunity is made available

* Copyright © 2010 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.
for students who have not yet experienced a computer science course and are undecided about their academic and career goals. It gives participating students the chance to investigate areas in which they have potential talent.

**Purpose**

The program grew out of the need to recruit more computer science majors. Recent U.S. Department of Labor surveys indicate that the areas set to experience the most growth from now until 2016 include computer technology, health services and social services. The computer related industry is expected to experience rapid growth, adding 489,000 jobs between 2006 and 2016 [2]. These statistics coupled with the declining enrollments put our country in a precarious technological position.

Furthermore many students indicate that they never considered a computer science major because they were unsure what computer scientists did for a living. Or if they did have an idea, it typically wasn’t a very appealing image. Additionally, exposure to computer science during high school is a deciding factor in 33% of girls’ decision to major in computer science [3]. Programming contests exist for those students who already are experienced in computer science. This alternative contest gives the same competitive spirit and exposure to the exploration of various areas of the computer science field.

**Participants**

The program was originally designed for high school juniors and seniors enrolled in or having completed Pre-Calculus, who have taken no computer programming courses. We have since targeted the program for high school juniors as it seems that seniors may have already decided on their course of study for college. We also want to encourage students to take CS courses at their high school if they are available.

Additionally, the high schools are instructed that prospective participants in IMPACT should be strong math students/problem solvers, should be “people persons” and possess good communication skills. The teams are required to include at least two female students in their team of four.

We invite all high schools from the four counties surrounding Siena College. The invitations are addressed to a math teacher at each school. Through previous outreach programs, our secondary mathematics education program and personal contacts, we are able to send personalized invitations to almost every school. We believe this was an important factor in running a well attended inaugural program. Subsequent programs have also been well attended based on the success of the previous programs.

**SESSIONS**

The program consists of a set of short (25 minute) instructional sessions where students learn about CS/IS topics. These sessions are followed by slightly longer (40 to 45 minute) activity periods where students work on some task(s) or solve problems related to the material of the instructional sessions. Possible topics are drawn from areas
such as computer graphics, software development, geographic information systems, finite automata, security and database systems. Almost all activities require students to work together. Once the activity time is up, the points are added for each team and they proceed to their next instructional session.

The program for students also includes advice from faculty members about academic and career opportunities in CS/IS. In addition, there are CS/IS professionals who describe their careers and the type of academic preparation needed for success in these careers. College students are also available throughout the program to provide general help to the groups and to sign off on the achieved objectives.

The following sections will briefly describe some of the instructional/activity sessions we have run in past years.

**Programming Activity**

This is one of the first mini-lessons that the students encounter. It explains how programming is just one aspect of computer science and details why it can be an important and useful skill. Finally, a short (15 minute) tutorial is given in Alice. After the lesson, the students go to the lab to create their own Alice animations.

Some of the incremental goals that the students are given, and scored on, include:

- Creating a world with specific objects
- Adding a specified number of objects to the scene
- Adding some evasive moves to the story. For example, add a spin move where either one of the characters spins to avoid a jab or kick.

We have run this session at every competition and it is always well received. The students enjoy the challenge, but also appreciate the creative aspect of the problems.

**Automata Activity**

At first appearance, automata would not seem to be an attainable concept in a 25 minute lesson, but after a short lesson students are asked to describe languages recognized by given finite automata state diagrams. They are also asked to construct automata to recognize specified languages.

Examples of activities for this session include:

- Construct a DFA for $M_3$ where $L(M_3) = \{w \mid w$ contains an even number of 0s\}.
- Describe the language recognized by the finite state machine in figure 1.
Students work hard in this session and are successful in achieving most, if not all, of the problems.

**Graphics Activity**

A short lesson on graphics enables students to complete an activity using Game Maker software. One year, the students created a Pac-man game similar to figure 2. They were given incremental tasks which served the dual purpose of scoring and guiding them to the final basic game. An example of one of the incremental tasks was to make the stars disappear and fun sounds play when Pac-man collides with them. When Pacman collides with the ice cream cone, which marks the end of the maze, play a fun sound and make Pac-man stop moving.

**Software Engineering Activity**

For this activity, three team members from each school are involved, as well as a pile of Tinker Toys. The first student enters a classroom and is allowed a short time to examine a structure made out of the Tinker Toys. That student then describes how to build the structure to another teammate waiting in the hallway. This information is then relayed to the third team member located in another classroom, who constructs the object described to the best of their abilities.

This lesson in communication and teamwork is used to represent the importance of exactly fulfilling customer needs in computer science. It is always an eye-opening experience watching the high school students try to describe what they see and determine what questions to ask.
Problem Solving Activity

The final activity of the day is a mathematical problem solving session where students work as a team on a set of discrete math problems. For the previously described activity sessions, the teams are divided into groups of two or three so that each school can participate in all activities. This problem solving activity requires the work of all members and is held immediately following lunch.

One example of the type of questions that the students work on is as follows:

Your school’s mathematics club has 8 members. You need to take a picture of the club for the yearbook. If it takes an average of 15 seconds to take a picture and you take a picture of all the ways you can line up the club members in a row, how long will it take to do this? Give your answer using the largest units of time possible. For example, if it will take 10,921 seconds you must record the answer as 3 hours, 2 minutes and 1 second.

DETAILS

Some of the details of the program are included so that our model can be adopted by as many schools as possible. The event was held on a school day during school hours. We choose to hold it during school hours to maximize participation among faculty and students. The feedback we have received indicates that this arrangement works well. The only constraint is that the schedule must be tight as some schools need the students to return to their high school before dismissal.

The number of schools participating have varied from 9 (during a snowstorm) to 16 bringing an average number of 55 students and 25 faculty to campus for the day. It is important that the faculty attend the instruction and activity sessions so that they may learn about computer science and bring this information back with them to the classroom and guidance office.

Schedule

A sample schedule is provided to demonstrate the orchestration of the day. The students and faculty are divided into two groups (to fit into classroom and lab space). Each of the 50 minute instructional sessions cover two topics. To facilitate the switching of topics, and minimize downtime, the presenting faculty switch rooms instead of the guests. This works well as we don’t need to wait for 30 teenagers to settle into new rooms again. This is indicated in the schedule as E ➔ V and V ➔ E meaning that Drs. V and E switch rooms to give the same instructional lesson to a different group of students.

At the end of each instructional session, the four team members decide how to divide and attack the problems. For example, in the Alice and automata sessions, two team members work on each activity session. Both subteams earn points for their school on their respective tasks.
While the students work during the first activity session, the faculty are free to observe the students at work. Many of the faculty expressed an interest in having their own teams and competing too! During the second activity session, faculty meet with a diverse group of alums. We try to get a mix of gender, careers, grad students and government vs. private sector jobs to represent the different possibilities of a major in computer science. The students listen to this same group of alums during lunch.

<table>
<thead>
<tr>
<th>Time</th>
<th>Student Group A</th>
<th>Student Group B</th>
<th>Teachers/Guidance A</th>
<th>Teachers/Guidance B</th>
</tr>
</thead>
<tbody>
<tr>
<td>8:30am</td>
<td>Welcome Session</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8:45 – 9:35am</td>
<td>Instructional Session I</td>
<td>Instructional Session I</td>
<td>Instructional Session I</td>
<td>Instructional Session I</td>
</tr>
<tr>
<td></td>
<td>$E \rightarrow V$</td>
<td>$V \rightarrow E$</td>
<td>$E \rightarrow V$</td>
<td>$V \rightarrow E$</td>
</tr>
<tr>
<td>9:45 – 10:30am</td>
<td>Activity Session I</td>
<td>Activity Session I</td>
<td>Free to observe</td>
<td>Free to observe</td>
</tr>
<tr>
<td>10:40 – 11:30am</td>
<td>Instructional Session II</td>
<td>Instructional Session II</td>
<td>Instructional Session II</td>
<td>Instructional Session II</td>
</tr>
<tr>
<td></td>
<td>$D \rightarrow Y$</td>
<td>$Y \rightarrow D$</td>
<td>$D \rightarrow Y$</td>
<td>$Y \rightarrow D$</td>
</tr>
<tr>
<td>11:40 – 12:20pm</td>
<td>Activity Session II</td>
<td>Activity Session II</td>
<td>Alumni Panel/CS Info Session</td>
<td>Alumni Panel/CS Info Session</td>
</tr>
<tr>
<td>12:30 - 1:30pm</td>
<td>Lunch with Guest Speakers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:30 - 2:10pm</td>
<td>Final Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:20pm</td>
<td>Contest Awards</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3: Sample schedule for the day's events**

**Budget**

For the size of event that we run, the cost averages out to be approximately $2000. The cost is shared between the Computer Science department and the school’s Admissions department. The Computer Science department covers the costs of general supplies, such as folders, CS department notepads, mailings and CDs (for Alice and Game Maker software). Admissions pays the food costs and provides gifts for the faculty to thank them for bringing their students. We obtain donations to give away as prizes to the top three teams.

**CONCLUSIONS**

The program has been well received by the students and faculty attending an IMPACT day. Several methods have been used to collect data about the students and faculty perceptions about computer science before and after the contest.

**Feedback from the students**

The students complete a survey at the beginning of the day to collect information about their future academic plans. Sample questions from this survey include questions about their favorite academic subject, their plans for college and their intended major.
From these surveys, we have learned that all students plan on attending college/university after high school, but very few intend to major in computer science.

We follow up by sending a subsequent survey to their residence several months after the event to see if there has been any change in plans and/or interests. Sample questions from this survey include questions about plans to take a computer science course and intended major (if a senior).

Through these surveys, we have seen that the day has changed their perceptions of computer science and several students have decided to take computer science courses.

One of the best comments from one of the students was “I just signed up to get out of school, but I had a great time! I thought we would be sitting in front of a computer all day, but this was much different than what I thought computer science was all about.”

**Feedback from the teachers/guidance counselors:**

A post-event survey is given to the adults who attend the event. It requests feedback on the individual programs and activities, insights they may have about student’s perceptions before and after the event, and their own perceptions before and after the event. One of the surprising results of this survey was that originally, many of these math teachers would not recommend computer science as a career option for their more talented math students. After completing the days’ events, they were “most likely” to recommend computer science as a career option. Just think of how many students this teacher’s new perspective may affect!

**Future Plans**

This coming January will be the fourth IMPACT program at our college. After this event, we plan to do a detailed analysis of the surveys that have been submitted by the students and the faculty to determine the effectiveness of the program.

We would love for this type of program to be run at other schools to convey to talented high school students, and their teachers, the types of things that they can do in computer science.

**REFERENCES**


WRITING INTENSIVE AND WRITING EXTENSIVE: A CONTINUUM FOR ADVANCING WRITING IN COMPUTER SCIENCE EDUCATION

PANEL DISCUSSION

Yana Kortsarts (Moderator)
Computer Science Department, Widener University, Chester, PA 19013
ykortsarts@mail.widener.edu

Timothy Dansdill
English Department, Quinnipiac University, Hamden, CT 06518
timothy.dansdill@quinnipiac.edu

Adam Fischbach
Computer Science Department, Widener University, Chester, PA 19013
jafischbach@mail.widener.edu

Mark E. Hoffman
Department of Computer Science and Interactive Digital Design, Quinnipiac University, Hamden, CT 06518
mark.hoffman@quinnipiac.edu

Janine Utell
English Department, Widener University, Chester, PA 19013
jmutell@mail.widener.edu

SUMMARY

The importance of writing in computing education is widely agreed upon. In addition to the individual efforts [4], national curricular guidelines for both Computer Science [1] and Computer Engineering [2] further underscore the importance of writing. The former insists that, “computer science students must sharpen their…writing skills in a variety of contexts—both inside and outside of computer science courses” [1]. The latter is less specific, but no less emphatic about writing’s importance: “Graduates should be able to communicate…ideas effectively in written form….” “This is necessary to ensure that students have the ability to communicate discipline-specific content.” [2]. Furthermore, national professional accrediting bodies such as ABET have emphasized the importance of integrating the teaching of effective written communication into CS curricula. ABET requires that these skills be developed through computer science
clearly the ability to write well is a priority for the profession, yet fulfilling
these standards does become very challenging for individual teachers, programs, and
institutions, taking into account the technical nature of the discipline.

One major challenge is that despite the acknowledgment of the importance of
writing (94% agreement in a recent survey [3]), there is not necessarily agreement among
these individual educators, administrators, or programs as to what constitutes effective
teaching of writing. There is often resistance to adapting writing to CS courses; CS
faculty do not feel properly trained or skilled to manage and grade writing. Students
often think that writing is not part of a computer science course and do not put much
effort into the writing process. It takes a lot of time to explain to students the importance
of the development of writing skills. One way to overcome some of these issues is to
collaborate with faculty who specialize in writing—either within a given curriculum or
across various disciplines. In this panel we present two different models of
collaboration.

One model (Widener University) has emerged from a university requirement calling
for students to take a certain number of writing-enriched courses. Here, Computer
Science (CS) and English faculty members coordinate on course design and assessment
to incorporate writing as a means of teaching effective communication. The other model
(Quinnipiac University) has emerged from a university commitment to training all faculty
through a grant-funded Writing Across the Curriculum (WAC) program. In this context,
the collaboration between the CS and the English faculty members focuses on faculty
development and research on writing practices in CS. As a result of this collaboration,
and the Quinnipiac University faculty member's participation in the wider, WAC
program, writing as a pedagogical component is now used in all CS classes.

Active audience participation is encouraged. The panel will provide an opportunity for
attendees to share their views and to exchange knowledge during a question-and-answer
period that will follow individual presentations.

1. UTELL, KORTSARTS AND FISCHBACH

We will present a model for faculty collaboration in writing in the disciplines (WID)
pedagogy that provides opportunities for active learning and the strengthening of written
and oral communication skills in CS. Instructors who participate in the faculty-driven
WID program at our regional comprehensive university seek out opportunities to
collaborate through workshops, shared course design and classroom teaching, and
assessment strategies; however, much of this collaboration occurs independently of any
formal structure or support.

In this context, we have designed an interdisciplinary collaboration over three years,
focusing on developing written and oral communication skills in two courses: CS II and
Introduction to Bioinformatics. In both courses, students were required to self-teach
content material, present their work to the class, and assess their fellow students’
learning. In constructing and implementing this component, an English/Writing faculty
member (Utell) taught lessons on effective oral presentation design, which were
integrated into the course work. She then joined the class to observe and assess the
students’ presentations, providing global written feedback for the students’ use, and meeting with the CS faculty member for discussion of student performance.[5,6,8]

In addition to presenting the results of this collaboration, the collaboration will be discussed in the context of writing and communicating in the disciplines, and specifically the implications of WID for the teaching of written and oral communication in the field of computer science. Our collaboration uses WID strategies to foster a more writing intensive classroom for active student learning, and to achieve the combined learning outcomes of mastering CS content and strengthening communication skills in preparation for professional life.

2. DANSDILL AND HOFFMAN

We will present a model of faculty collaboration where writing is an integral pedagogical component of all classes including computer science classes that promotes student engagement and understanding, and supports the curricular and accreditation goals of effective written and oral communication. Quinnipiac University’s grant-funded Writing Across the Curriculum (WAC) program is committed to training all faculty in writing to learn (WTL) and writing in the discipline (WID) strategies. Our collaboration emerges from a mutual interest in pervasive writing strategies, and from our joint experience as workshop trainers in our WAC program and in other venues such as SIGCSE 2009. Our collaboration centers on understanding how and why particular types of writing are valued in computer science, and to develop strategies to promote wider adoption of writing across the WTL-to-WID continuum. Toward that end, we have jointly authored papers on writing in computing education [4] and a survey of computing faculty [3]. We have also designed and conducted workshops adapting writing strategies developed by the QU WAC program for computer science and other technical faculty. We are conducting a joint sabbatical project in spring 2010 to further investigate the use of writing in the CS curriculum and to understand the impact of various strategies to promote writing CS education.

Computer Science faculty (Hoffman) uses writing across the WTL-to-WID continuum in all classes. WTL strategies employ writing such as a reading response journal to develop and informally assess student understanding, as a basis for in-class discussion, and to promote student engagement. We have found that properly-constructed low-stakes WTL assignments help students develop conceptual understanding by focusing their critical thinking and requiring them to articulate their understanding in writing. English faculty (Dansdill) has studied and presented on faculty resistance to using writing in their courses [7], and has found that low-stakes WTL assignments are less intimidating for students and faculty making them easier to adapt to any class, while adding a writing “intensive” element to courses that formerly did not employ a spectrum of writing. Low-stakes WTL writing may be linked or bridged to formal WID assignments where students communicate and demonstrate their understanding in appropriate forms for the intended audience.

In our computer science curriculum, and across the University, writing is becoming a pervasive pedagogical strategy to develop, to communicate, and to assess learning. Our collaboration emerged from and continues to develop in this context. In this sense it is extensive; extending beyond individual assignments or courses.
3. BIOGRAPHIES

Yana Kortsarts is an Associate Professor of Computer Science at Widener University. She has been actively involved in integration of writing and oral enhanced components into undergraduate computer science curriculum since 2005. She has been successfully collaborating with English colleague Janine Utell since 2007. She is a co-author of several papers on development of written and oral communication skills in undergraduate computer science curriculum.

Mark Hoffman, a Professor of Computer Science at Quinnipiac University, has been an active member of the highly successful writing across the curriculum (WAC) program at Quinnipiac University since 2003. Mark has been trained as WAC trainer and has conducted WAC writing workshops at Quinnipiac University and other institutions. He presented papers co-authored with Timothy Dansdill at SIGCSE and CCSCNE. Mark participated in a panel at the 9th International WAC Conference, May 2008, Austin, TX.

Timothy Dansdill, an Associate Professor of English at Quinnipiac University, is the coordinator of its First Year Seminar course: "The Individual in Community." He has been actively associated with QU's Writing Across the Curriculum program since 2002 - both as a faculty trainer and as the chief organizer of its bi-annual conference (2006; 2008). He is a long time co-investigator with colleague, Mark Hoffman, on the history of writing practices and programs in Computer Science education.

Janine Utell is an Associate Professor of English and Writing Center Faculty member at Widener University. She has been active in the university's writing-in-the-disciplines program since she joined the faculty in 2003, collaborating with colleagues in Psychology, History, Chemistry, and Computer Science on teaching and research as well as facilitating faculty development opportunities. She has presented at numerous national and international conferences on WID pedagogy and its relationship to general education and critical thinking.

Adam Fischbach is an Assistant Professor of Computer Science at Widener University and has been actively involved in the department’s efforts to integrate oral and written communication skills into the computer science curriculum. He recently co-authored the paper "Developing Oral and Written Communication Skills in Undergraduate Computer Science and Information Systems Curriculum" with Yana Kortsarts and Janine Utell.

REFERENCES

[1] Association for Computing Machinery and Institute of Electrical and Electronics Engineers Computer Science Joint Task Force, Computing Curricula 2001

[2] Association for Computing Machinery and Institute of Electrical and Electronics Engineers; Joint Task Force on Computer Engineering Curricula, Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering, 2004


TEACHING VIRTUAL REALITY TO UNDERGRADUATES

TUTORIAL PRESENTATION

Daniel Cliburn
Department of Computer Science
The University of the Pacific
Stockton, California 95211
(209) 946-2093
dcliburn@pacific.edu

ABSTRACT

Virtual Reality (VR) is a contemporary area of research with many exciting applications in simulation and training. The National Academy of Engineering recently named “Enhance Virtual Reality” as one of fourteen “Grand Challenges for Engineering” (http://www.engineeringchallenges.org/) and many students find the subject to be exciting and engaging. However, VR receives little or no attention in most undergraduate Computer Science programs. This tutorial will have two objectives. The first will be to introduce attendees to the field of VR through several freely available online lesson modules, and attendees will be shown how the modules are actually being integrated into a variety of existing courses. The second tutorial objective will be to describe two immersive stereo display systems: a Head-Mounted Display (HMD) unit and a stereo display wall.

* Copyright is held by the author/owner.
ACCESSIBLE ROBOTICS AND INTELLIGENT SYSTEMS FOR
SOCIAL SCIENCE UNDERGRADUATES

POSTER SESSION

Tom Armstrong
Department of Mathematics and Computer Science
Wheaton College
Norton, MA 02766
508.286.5658
tarmstro@wheatoncollege.edu

In 2001, Wheaton implemented an innovative curriculum called Connections [5] where students enroll in pairs of linked courses that span disciplinary boundaries. We developed a course offering entitled Intelligent Systems, which will be linked with several Psychology courses (e.g., Consciousness, Comparative Animal Behavior) and a Sociology course. Intelligent Systems provides a rigorous introduction to robotics and surveys selected topics in artificial intelligence. Unlike the upper-level Computer Science courses in this area, this course is intended for non-major students.

We explicitly created this course to increase the enrollment of social science students in our course offerings, and to foster scholarly collaborations between faculty in those disciplines with Computer Science. The wide-reaching appeal and interdisciplinarity of robotics and artificial intelligence drove its selection as the course subjects. To further increase the viability of the course, it also satisfies one of the College's curriculum distribution requirements in quantitative analysis. To make the course accessible, no programming experience is expected and no mathematical background is required. In the initial offering of the course, students from the Psychobiology, Psychology, Music, and Economics majors enrolled along with students from Biology, Bioinformatics, and our study-abroad program.

Popular science and press coverage of robotics and artificial intelligence indicate broad appeal, and students outside of the Computer Science major seem both curious and eager to learn more about the topics. Enticing students with hands-on entry-level components is now, more that ever before, particularly affordable and accessible [1]. The popularity drives coursework inside the major with, for example, robotics as the unifying theme for the standard AI class [4]. Similarly, a significant number of intelligent systems-infused offerings are directed at students outside of Computer Science. Several

* Copyright is held by the author/owner.
approaches focus on specific pairings of disciplines like Computer Science and Art [10] or Computer Science and Mechanical Engineering in specialty courses [6, 7]. However, most contact with non-major students happens in a CS0-style course and often some portion of the content is devoted to AI [2]. Or, the course has content tailored for a specific other discipline [3]. Particularly at small colleges, there are few opportunities to offer courses for non-majors. *Intelligent Systems* is distinct in form and purpose from these other offerings for two reasons: 1) the course content is a rigorous technical introduction to robotics and artificial intelligence without prerequisites; 2) it serves as a Computational Thinking (CT) [9] course targeted at students in the social sciences with, we argue, a compelling set of topics with greater appeal than a traditional or targeted CS0 offering.

The course is designed in a way that it is accessible to students with no prior technical background. The first half of the course is a rigorous introduction to robotics beginning with fundamental paradigms and then adding components to build larger systems. We discuss and experiment with effectors and actuators, evaluate kinematics and methods of locomotion, and explore sensors. Students gain hands-on learning experience using iRobot's Create, an inexpensive programmable robotics platform, and examining other robotic systems. Students are tasked with some programming tasks (e.g., wall following) and are expected to program the robots using a modified version of Alice [8] and complete written assignments (e.g., designing and analyzing a robot for a specific task).

The second half of the course builds on students' knowledge of sensing and acting in autonomous robotic systems to focus on planning. We begin with discussions of knowledge representation and single- and multiplayer games. We spend several weeks on approaches to natural language processing and machine translation. Students use their own second language skills and commercially available software to test and evaluate Turing test contestants and machine translation systems. We cover emergent behaviors of complex systems in the context of robot swarms and artificial life. The semester is rounded out with a discussion of robotic ethics and the use of autonomous vehicles in military and humanitarian domains. Each topic is accompanied with relevant literature and significant class discussion.

During the second half of the semester, student groups propose and implement a project. Students have access to a variety of robotics platforms (e.g., LEGO NXT, iRobot Create, and Arduino kits), other computing resources outside of class, and Computer Science majors with programming experience. One group built a decision tree learning system for a game of “20 Questions” about Wheaton's campus. Another group programmed a swarm of square dancing iRobot Creates and took extended-exposure photographs of the swarm. Finally, a group built an Arduino-powered version of Braitenberg's third vehicle, *Love*.

REFERENCES


ROLE OF THE PROGRAMMING ASSIGNMENT SUMMARY AS A QUALITATIVE ASSESSMENT OF INDIVIDUAL STUDENT PERFORMANCE

POSTER SESSION

Karina V. Assiter
Department Of Computer Science
Wentworth Institute of Technology
Boston, MA 02155
assiterk@wit.edu

ABSTRACT

Homework assignments in a first or second year undergraduate programming course (CS1 - CS3) are essential for skills development; instructors who assign homework in CS1 - CS3 would like submitted solutions to both accurately reflect student performance, and to indicate the level of student understanding of course topics. Unfortunately, these are hard to gage when students increasingly have access to solutions on the internet and are frequently encouraged (for sound pedagogical reasons) to work on assignments in groups or pairs. Questions that naturally arise include: a. how much of a students' work is original, in other words not acquired from a friend or an open-source repository, and b. how much did an individual student, working in a group or pair, contribute to the final solution. To address these assessment concerns, we incorporated a summary into the list of programming assignment requirements. Though it serves many roles, including as a vehicle for analysis of course topics, it's most important role is as qualitative measure of an individual students' performance, which can then indirectly inform their final quantitative assessment. This poster describes this assignment summary: including its original purpose, example analysis and summary questions, textual analysis of student responses, and, finally, correlations between responses and student assignment scores.
INTRODUCTION

In the CS1 - CS3 foundation courses, students have to be able to develop basic software development skills through programming based homework. As instructors, we would like submitted solutions to reflect student performance, as well as to indicate the level of student understanding of course topics. This is difficult to gauge when students have access to software repositories on the internet or when working together in groups or pairs; graders who assess submitted work have to evaluate: a. how much of it is original, and b. how much did an individual student contribute to the final team solution.

Plagiarism

For every programming assignment there are likely to be student solutions (or sub-solutions) that were acquired either from an electronic source, or from another person (friend or paid programmer); as the author states in [1] "the store of solution code out on the internet keeps getting better. The search engines are getting better at finding it". Numerous authors have highlighted both the methods in which students cheat, as well electronic and traditional methods of detecting plagiarism [2, 3, 4]. One author suggests that, since students are able to subvert detection strategies, academics should also raise awareness among students about academic integrity [5].

Fuzzy Collaboration

A more complex problem is assessing individual performance when students are encouraged to collaborate on assignments (in pairs or teams) [6]. There is increasing evidence that students benefit from peer-based collaboration [7, 8]. Unfortunately, there is a fine line between collaboration, where all students contribute to the final assignment solution, and cheating, where either a) one (or more) student(s) don't contribute fully to the pair/team solution or where, b) students alternate assignments ("I will do assignment 1 and you do assignment 2"). As there is only one final solution, this type of cheating, which we term "fuzzy collaboration", is hard to detect (let alone prove); we were only able to discover it in cases where either a student admitted to it verbally, or when one of the students (in a pair) had little (or no) understanding of the problem concepts.

ASSIGNMENT SUMMARY

To address our assessment concerns in CS2 and CS3, we incorporated a summary into the programming assignment requirements. Though we were initially concerned with evaluating student performance, we discovered that we could also use the summary to:

• Require Analysis of the topic that the assignment covered
• Confirm that students gained an understanding of assignment concepts.
• Verify that collaborating students each contributed to the assignment
• Determine the complexity level (perceived by students)
• Gathering feedback about assignment improvements.
The questions on the summary varied depending on the course and the assignment, but most include some form of the following:

1. Describe how you approached and solved the problem. If you worked in pairs, how did you make sure that each student contributed?
2. Where did you have trouble? How did you move forward? What topics still confuse you?
3. What did you learn from this assignment? (be specific)

**Student responses**

In this section of the poster, we include: questions from students' assignments, student responses from selected assignment/course combinations, observations about how the grader interpreted these responses and the score that the student received on the associated work.

**CONCLUSIONS**

In this section of the poster, we summarize correlations between student responses and assignment scores. As this is a qualitative assessment, we caution against using responses as anything more than a road map for subsequent assignment evaluation.

**REFERENCES**


ABSTRACT

This poster presentation summarizes the major pedagogical innovations of the TeachScheme!, ReachJava approach, reports on the results of our past faculty workshops (particularly the adoption of our approach by respected colleges, universities, and high schools), and invites visitors to a free (NSF-funded) faculty workshop in Summer 2010. Several textbooks and other instructional materials using this approach, e.g. (Bloch, 2010), (Felleisen F. F., 2008), (Felleisen F. F., 2010), (Felleisen F. F., 2001), (Sperber, 2009) will be available for examination.

A first course in computer programming should not be about the current “hot” language in industry – which may be obsolete by the time today’s freshmen graduate – but rather about lasting, transferable concepts and practices of good programming. Yet beginning programming students spend much of their time wrestling with the language, and often mistake that as the subject of the course. The programming language distracts from the course material; on the other hand, students need a real language to write real programs that really run on real computers.

We resolve this dilemma by starting in a language with simple, consistent syntax and semantics, currently a subset of Scheme (omitting I/O, assignment, sequence, higher-order functions, and local definitions). Our pedagogically-oriented IDE enforces this subset, and gives error messages appropriate to the current subset, but allows students as they outgrow each subset to advance to a larger one with a few mouse clicks. Students become comfortable with fundamental programming concepts — variables, function
composition, function definition, parameter passing, data types, design for reuse and modifiability, conditionals, fields, polymorphism, self-reference and recursion, functional abstraction, event-driven programming, model/view separation, etc. — in this sheltered environment before encountering the same concepts in the more bewildering world of Java, C++, etc.

Simultaneously, students are trained in a step-by-step design recipe for software development: a series of concrete questions, with concrete products at each stage:

1) Identify the **purpose, inputs, and outputs** of the program (function, method, whatever) to be written;
2) Identify (and, if necessary, define) **data types** relevant to the problem at hand;
3) Write **examples or test cases** of how the program will be invoked, in legal syntax and accompanied by expected results, using the data types from step 2 as a guide;
4) Write a **program skeleton**, the syntax to define a function with the name and parameters chosen above;
5) Write an **inventory** of available and likely-to-be-needed expressions, based on parameter names and their data types;
6) Choose and combine items from the inventory to form a complete program **body** (the hardest part, but in practice step 5 often does most of the work);
7) **Test** the program by running it on the examples from step 3.

We emphasize **data types** throughout, not only as a fundamental concept, but as an invaluable tool in coding: to every data type correspond both a natural **coding pattern**, which provides a rough draft of the code and helps students avoid “blank page syndrome”, and a natural **testing pattern**, which provides guidance in building test suites. In particular, recursion is introduced as simply the application of already-learned coding patterns to a self-referential data type. The concrete methodology also provides a handy grading rubric that shows students that **every** step matters, not only the coding.

For non-majors, we aim to convey important programming concepts and methodology in one language. For CS majors, the course switches from Scheme to Java late in the first semester or between first and second semesters. The Java stage is not independent, but builds on and reinforces the programming concepts and methodology already learned in Scheme, with explicit discussion of similarities, differences, and the continued applicability of the concepts and methodology. Students learn to apply the same test-driven, step-by-step design recipe in Java that they’ve been using in Scheme. The result is a student who, after a year of coursework, can approach programming problems in a principled manner (not “hack it until it works”), with understanding and perspective.

Although we currently use Scheme as a first language and Java as a second, the approach is applicable to other languages. Whatever the language, however, we believe the first exposure to programming should be **functional** rather than imperative/sequential/procedural: not only do functional programs have simpler semantics, relying on the familiar model of algebraic expression evaluation rather than a load/store machine model, but it’s enormously easier to write test cases for functional programs than for stateful ones. Stateful testing, along with stateful programming, can be introduced late in the first semester after students have thoroughly internalized functional techniques.
Our approach has been adopted at a number of colleges and universities, including Rice, Northeastern, the University of Chicago, Northwestern, the University of Utah, Cal. Poly San Luis Obispo, Vassar College, and the University of Delaware, as well as dozens or hundreds of high schools. We’ll be offering free (NSF-funded) one-week workshops in Summer 2010 at four locations around the U.S.

REFERENCES


ABSTRACT

Ontology of all of the computing disciplines is available for use in a number of applications. It describes various disciplines, topics, subtopics that belong to the domain of computing, including computer science, computer engineering, information systems, information technology, software engineering, and the many very specialized areas that are emerging.

BACKGROUND AND DESCRIPTION

The computing ontology [1] is a comprehensive collection of topics, put in order by experts in the various subfields. All the fields and subfields are organized as classes and objects in a hierarchical conceptual structure with parent, child or sibling relationships among them. Apart from these relations the fields/classes also have the properties Isa, Uses, Used By, Equals, Is Part of, Has Part etc. Protégé, an open source ontology editor is used to build, organize and maintain these properties; inter-dependencies between these properties are maintained by symmetric, inverse and transitive relationships within the properties eg. Uses -> Used By are inverse to each other. Protégé generates a special type of XML file called OWL (Web Ontology Language). OWL is an ontology modeling language having a XML surface syntax.

Within the domain of computing education, the computing ontology provides a mechanism to identify overlapping areas in and to take advantage of these overlaps to create new courses and make curriculum recommendations. The ontology provides an
objective representation of the entirety of the computing domain so that curriculum and course developers can make conscious choices about what to include and what to exclude from any particular effort.

**KEYWORDS**
Ontology, education, relationship, overlaps.

**Project Goals**

An important task of the current work is to provide a usable, visible interface to the Ontology. This visualization provides a graphical representation of the ontology that is essential for effective understanding of bridges between domains. The poster will display the current state of the visualization effort and seek input from conference attendees about the most useful and meaningful representations.

The Computing Ontology also provides a controlled vocabulary to be used as an indexing tool in collections of resources for computing education. Examples in the CITIDEL project, implemented in DSpace, and the Ensemble project, built with Drupal, will be illustrated on the poster.

The top level nodes for the Computing Ontology are these:

- AlgorithmsAndTheory
- ComputerHardwareOrganization
- ComputingAndNetworkSystems
- ComputingEducation
- ComputerGraphics
- DiscreteStructures
- EthicalAndSocialConcepts
- HistoryComputing
- InformationTopics
- IntelligentSystems
- MathematicalConnections
- ProgrammingFundamentals
- ProgrammingLanguages
- SecurityTopics
- SystemsDevelopment
- SystemsAndProjectManagement
- UserInterface

**CONCLUSIONS**

The content of the ontology comes from a comprehensive survey of curriculum documents, the ACM Computing Classification System, and other taxonomies of computing topics. These were reviewed, edited, and merged by people with expertise in the individual topic areas. Further input is welcome and revisions will be an ongoing process because the field itself changes. The poster will expand several of these areas,
show the visualization efforts, and illustrate the use of the ontology for curriculum review and development.

REFERENCES

[1] The Ontology project can be tracked at http://what.csc.villanova.edu/twiki/bin/view/Main/OntologyProject
ENSEMBLE: ENRICHING COMMUNITIES AND COLLECTIONS TO SUPPORT EDUCATION IN COMPUTING

POSTER SESSION

Dr. Lillian (Boots) Cassel
Professor
Villanova University
610-519-7341
lillian.cassel@villanova.edu

Dr. Ed Fox
Professor
Virginia Tech
540-552-8667
fox@vt.edu

Dr. Frank Shipman
Professor
TAMU
979-862-3216
shipman@cse.tamu.edu

Dr. Weiguo Fax
Associate Professor
Virginia Tech
540-231-6588
wfan@vt.edu

Dr. Dan Garcia
Lecturer
Univ of Cal, Berkley
510-642-9595
ddgarcia@cs.berkeley.edu

Dr. Richard Furuta
Professor
TAMU
979-845-3839
furuta@cs.tamu.edu

Dr. Lois Delcambre
Professor
Portland State Univ
503-725-2405
lmd@cs.pdx.edu

Dr. Peter Brusilovsky
Associate Professor
University of Pittsburg
412-624-9404
hislop@drexel.edu

Dr. Greg Hislop
Associate Professor
Drexel University
215-895-2179
peterb@pitt.edu

Sridhara Potluri
Graduate Student
Villanova University
610-519-5597
sridhara.potluri@villanova.edu

ABSTRACT

Ensemble is a new NSF NSDL Pathways project working to establish a national, distributed digital library for computing education. Ensemble is building a distributed portal providing access to a broad range of existing educational resources for computing while preserving the collections and their associated curation processes. CITIDEL and CSTA are two of the major educational resources that are part of this expanding...
collection. We want to encourage contribution, use, reuse, review and evaluation of educational materials at multiple levels of granularity and we seek to support the full range of computing education communities including computer science, computer engineering, software engineering, information science, information systems and information technology as well as other areas often called "computing + X" or "X informatics".

The various aspects of the project include: (1) developing a distributed portal, (2) defining appropriate metadata and methods for indexing computing resources - including using an ontology to describe computing topics, (3) instrumenting the portal so that we can track use and reuse of resources (so that faculty can easily get metrics analogous to citation counts for the educational resources that they create), (4) integrating social software into the portal, (5) developing mechanisms to handle use of resources at multiple levels of granularity, (6) articulation of the various topics and how they overlap in the various computing disciplines, (8) user development and dissemination, (9) information finding and collection development, and (10) evaluation.

Ensemble provides content, communities, and tools for computing educators and students. The content consists of freely available computing education resources stored within Ensemble or at other locations. Ensemble provides federated search, indexing, annotation, reviews, and other services to make these resources accessible, visible, and more useful to the community. Ensemble also provides other sources of information of interest to computing educators. This content includes information streams such as news, event notices, and blogs.

Ensemble communities support interaction among computing educators via facilities such as discussion forums, posting of working papers, and connections to venues such as Twitter and Facebook. These services support open collaborations such as a CS1 community site and also hosts closed working spaces for groups like the ACM Education Board and the Future of Computing Education Summit working groups. Ensemble tools provide access to more advanced facilities to help instructors and students' access and organize materials relevant to computing education. An example is Visual Knowledge Builder, which provides a workspace for collecting and organizing computing education resources. Ensemble also includes an alternate interface accessed via the Ensemble pavilion in Second Life.

Ensemble supports the full range of computing disciplines and also programs that blend computing with other STEM areas (e.g., X-informatics and Computing + X). Ensemble is funded by the National Science Foundation via the NSDL Pathways program.

REFERENCES


CSTA SOURCE: A WEB REPOSITORY OF K-12 COMPUTER SCIENCE TEACHING AND LEARNING MATERIALS

POSTER SESSION

Dr. Lillian (Boots) Cassel  
Professor  
Department of Computing Sciences  
Villanova University  
610-519-7341  
lillian.cassel@villanova.edu

Sowmya Moturi  
Graduate Student  
Department of Computing Sciences  
Villanova University  
408-621 4865  
sowmya.moturi@villanova.edu

ABSTRACT

The Computer Science Teachers Association is a membership organization that supports and promotes the teaching of computer science and other computing disciplines. The project is about designing and maintaining CSTA portal which is a web repository of K-12 Computer Science Teaching and Learning Materials. It is a collection of materials developed specifically for computer science and information technology education.

CATEGORIES AND SUBJECT DESCRIPTORS

K.3.2 Computer and Information Science Education

GENERAL TERMS

Design

KEYWORDS

Digital library, computing education

* Copyright is held by the author/owner.
INTRODUCTION

CSTA Source is a Web Repository of K-12 Computer Science Teaching and Learning Materials which is built to reach the people across the full range of audiences for computing education. It contains materials developed specifically for computer science and information technology education. The repository is kept updated with the recent materials which are reviewed by the administrators before adding them to insure that they are comprehensive, relevant, age-appropriate and pedagogically sound.

The repository content is organized as Communities and sub communities. Within each community are a number sub communities and collections. Each collection has a number of items relating to the collection, which can be in any format. The repository provides the flexibility to accommodate differing needs of communities by allowing them to decide on policies such as: who contributes content, whether there will be a review process, who will have access, workflow - reviewing, editing, metadata, management of collections. The CSTA web repository is customized to include new features to address the needs of K-12 teachers and students. Some of the recent enhancements include the ability of users to comment and give ratings to the resources in the repository, adding a counter which counts the number of visits to the K-12 repository, adding a counter which counts the number of downloads of individual resources in the repository, dividing users into groups and providing different access privileges to people in the different groups.

Project Goals

- Building a web portal for CSTA to provide access to a broad range of existing educational resources for computing while preserving the collections and their associated curation processes.
- Expansion of CSTA Source web repository with curriculum and learning materials specifically related to K-12 computer science.
- Customizing the web repository to make it more intuitive for its users.
- Encouraging contribution, use, reuse, review and evaluation of educational materials at multiple levels of granularity.
- Creating and disseminating materials to support the implementation of the curriculum. Facilitate sharing of projects/ideas among members.
- Building a strong community of computing educators who share their knowledge.
- Providing teachers with opportunities for high quality professional development.

POSTER CONTENT

The poster will provide an overview of the CSTA project work with a particular focus on ways to build a strong community of computer science educators who share their knowledge. The various aspects of our project include: (1) making various enhancements to the web repository by adding new features, (2) defining appropriate metadata and methods for indexing computing resources - including using an ontology to describe computing topics, (3) instrumenting the portal so that we can track use and reuse of resources (so that faculty can easily get metrics analogous to citation counts for the
educational resources that they create), (4) integrating social software into the portal, (5) providing access to resources at multiple levels of granularity.

ACKNOWLEDGEMENTS
CSTA project is funded by ACM and NSF (National Science Foundation).

REFERENCES
[1] Computer Science Teachers Association
http://www.csta.acm.org/About/sub/PurposeAndGoals.htm
The education of a technical student in the area of web design is often 'not a pretty sight'. They adapt to the skills but do not acquire the 'design' aspects necessary to make a website both functional AND attractive.

Why not take a talented student who has mastered design and add the skills of Web Development to their repertoire?

The Bachelor's degree in Web Design and Multimedia was designed as a completion degree and focuses on attracting students with Associates’ degrees in graphic design. These students have proven their ability to function well in their respective discipline and also have achieved academic recognition by already completing one degree.

This student may be a recent graduate or may be already out 'working in the field'. The degree delivery is done 'online' allowing for both situations. Many graphic artists are finding that the print medium is no longer sufficient for their career goals. Additionally they may need a Bachelor's Degree to progress within their organization. This degree program achieves both goals.

Students will be trained to combine artistic vision with technical skill in order to design and computerize environments where different sources of information (text, graphics, moving images and sound) work together. The 2-year completion degree will compliment the student’s associate degree without duplicating the area they studied.

Three of the courses are based on the CIW- Certified Internet Webmaster industry certification. Students may choose to sit for the 2 exams necessary to fulfill the Associate Design Specialist certification at the end of their first year of study. Vendor-neutral certification can be a differentiating factor for a professional who is looking to advance in the IT job world. To a hiring employer, not only does a certification provide evidence of an individual's familiarity with a particular technology or practice. It also demonstrates
initiative on the part of the applicant because he or she has invested the extra time and
effort to become certified. In turn, employers often pay certified employees more than
they pay non-certified employees.

The curriculum also includes client and server side scripting as well as device
development. Students will gain project management skills as they work on real projects
throughout their degree coursework which will culminate with a capstone Senior Project
experience.

According to Diana Middleton, author of a recent article published in the Wall Street
Journal, "Career experts say the key to securing jobs in growing fields will be coupling
an in-demand degree with expertise in emerging trends. For example, communications
pros will have to master social media and the analytics that come with it; nursing students
will have to learn about risk management and electronic records; and techies will need
to keep up with the latest in Web marketing, user-experience design and other Web-
related skills." She also states, "More than two million new technology-related jobs are
expected to be created by 2018, according to the BLS. Jobs that are expected to grow
faster than average include computer-network administrators, data-communications
analysts and Web developers."1

software engineers are expected to grow rapidly from 2008 to 2018. Expanding Internet
technologies have spurred demand for these workers, who can develop Internet, intranet,
and Web applications. Demand for Web administrators and Web developers will also be
strong. More of these workers will be needed to accommodate the increasing amount of
data sent over the Internet, as well as the growing number of Internet users. In addition,
as the number of services provided over the Internet expands, Web administrators and
developers will continue to see employment increases."2

In the area of graphic design, the Bureau of Labor Statistics states, "employment
is expected grow about as fast as average. Keen competition for jobs is expected;
individuals with Web site design and animation experience will have the best
SB10001424052748703278604574624392641425278.html?
is expected to grow 13 percent, as fast as the average for all occupations from 2008 to
2018, as demand for graphic design continues to increase from advertisers and computer
design firms. Moreover, graphic designers with Web site design and animation
experience will especially be needed as demand increases for design projects for
interactive media—Web sites, mobile phones, and other technology. Demand for graphic
designers also will increase as advertising firms create print and Web marketing and
promotional materials for a growing number of products and services. Growth in Internet
advertising, in particular, is expected to increase the number of designers. However,
growth may be tempered by reduced demand in the print publishing, where many graphic designers are employed.\textsuperscript{3}

The above statistics further validate the need for combining the skill sets. Colleges are under pressure to design innovative curriculum offerings that will prepare students for the new jobs and promising career paths that are expected to emerge in the next few years. Web and interactive media graduates should continue to enjoy excellent job prospects as the expanding integration of Internet and interactive media technologies results in a growing need for specialists who can develop and support Internet, intranet, and interactive media applications.

\textsuperscript{3}http://www.bls.gov/oco/ocos090.htm
COMPUTER GRAPHICS TAUGHT BY BUILDING A
RENDERING ENGINE

POSTER SESSION

Paul E. Dickson
Hampshire College
School of Cognitive Science
893 West St.
Amherst, MA 01002, USA
(413) 559-5861
pdickson@hampshire.edu

ABSTRACT

We have created a curriculum for a computer graphics course that teaches computer graphics through the implementation of a ray-tracing rendering engine. Students begin the semester by building a 2D ray-tracer and then shifting it into 3D. By the end of the semester they have implemented a rendering engine capable of handling 3D objects with point and ambient light sources.

The premise of this course is that students gain a greater understanding of the concepts behind computer graphics by implementing them than by simply using them in a commonly available graphics language such as OpenGL. In this course students must apply the information taught about how ray-tracing works in order to create each week’s assignment. Each assignment builds on the previous assignments and leads to the creation of the total rendering environment.

Students build the engine in C++ and use it to implement an object-oriented design. The overall structure of the engine is defined by library header files given with each assignment. Specifying the basic structure of the code makes it possible to build a test suite of images that can be used to test each student’s engine. Building the renderer also helps students to see how different components of a ray-tracer interact and to better understand graphics.

Students are required to work in pairs for this course. The reason for this is twofold: first, the assignments are too large for a single individual to handle easily; second, it is good practice for them to learn to work with another person while coding.

* Copyright is held by the author/owner.
The curriculum has been used once so far and the results are positive. One-third of the student completed all assignments for the course successfully and two-thirds made significant progress on the material. All students gained a good working knowledge of ray-tracing.

One of the real successes of this course was how much better students programmed after the semester was completed. In classes taught since the semester of this course, all of the students from this course have shown and expressed greater confidence in their programming and ability to handle large assignments. The course greatly improved their ability to debug code. Students found this course to be difficult but did not get discouraged.

Anyone interested in teaching this course or one similar can find the material pertaining to it at http://helios.hampshire.edu/~pedcs/classes/cs209Spring09/index.html.
THE BENEFITS OF USING SCRATCH TO INTRODUCE BASIC
PROGRAMMING CONCEPTS IN THE ELEMENTARY
CLASSROOM

POSTER SESSION

Patrice Gans
The Fraser-Woods School
173 South Main Street
Newtown, CT 06470
203-426-3390
pgans@fraserwoods.com

Computer programming is traditionally reserved for the high school classroom. The majority of students often take their first programming course as a high school junior or senior as many high level mathematical concepts are prerequisites for the class. Consequently class sizes are shrinking as high school students find themselves overwhelmed with course options and intimidated by the rigor of the computer programming curriculum.

Recognizing that the key to attracting students is to grab their attention at an earlier age, the Lifelong Kindergarten group at the MIT Media Lab in collaboration with UCLA Graduate School of Education and Information Studies developed Scratch. Scratch, a new graphical programming language, was created specifically to attract a nontraditional student population, those students who might not have ever imagined themselves as programmers. With this goal in mind, the developers sought to construct a new approach to programming. By taking advantage of recent advances in computing power and interface design the designers of Scratch were able to create a programming language that is more engaging and accessible for children and teens. The program enables students, as young as six, to create interactive stories, games, and animations which they can then share with one another over the Internet through the MIT Scratch website (http://scratch.mit.edu).

Over the last year and a half, I have been teaching Scratch in my computer classes to students, ages seven to thirteen. My experience with this age group is that the overwhelming majorities of children quickly adapt to the Scratch environment and rise to the challenge of creating their own programs. The students find the Scratch language

* Copyright is held by the author/owner.
is a perfect match to their learning style, providing hands-on interactive learning. Experimentation and exploration comes naturally to this age group, which makes it the ideal time for them to be learning to program. The Scratch programming language is comprised of a series of command “blocks” (e.g., statements, loops, variables) which “snap” together to create a script. Using blocks reduces the barriers that can be created by traditional programming languages’ abstract syntactic and semantic rules. These blocks only snap together if they are syntactically appropriate. Without any of the obscure punctuation and syntax of traditional programming languages, Scratch has successfully brought programming into the elementary classroom. The students find the commands to be extremely intuitive, and as a result, they are able to create their first program quickly and painlessly.

Unlike traditional programming languages, students receive instant feedback while working with Scratch. In a matter of seconds they can determine if the program is behaving as intended. This also provides them with the opportunity to learn about the design process which entails starting with an idea, creating a working prototype, then experimenting with it, debugging it when things go wrong, receiving feedback from others, then revising and redesigning as needed.

Students set their own design goals when working with Scratch. They are empowered by the flexibility that Scratch provides them to create their own versions of games and animations. In the classroom, a single objective can yield a large variety of different Scratch creations, thus allowing each student to express their individuality.

Computer programming is an important skill for the 21st century. Learning to program offers benefits for everyone: it enables students to express themselves more fully and creatively, helps them develop as logical thinkers, and helps them understand the workings of the new technologies that they encounter everywhere in their everyday lives. Scratch is an ideal way for students of all ages to experience the benefits that programming has to offer.

The goal of Scratch is to help children and teens become fluent with digital media, empowering them to express themselves creatively and make connections to powerful ideas. By using Scratch in the elementary classroom, students became creators of digital media instead of passive observers.
CREATING THE TECHNOLOGICALLY SAVVY K-12 TEACHER

POSTER SESSION

Susan P. Imberman, Roberta Klibaner
College of Staten Island
2800 Victory Blvd.
Staten Island, NY 10314
susan.imberman@csi.cuny.edu, roberta.klibaner@csi.cuny.edu
718-982-2850 contact
718-982-2856 fax

The ubiquitous nature of computer science has become a dominant form of cultural pedagogy that both youth and, to a lesser degree, adults are currently engaged. Computer applications, even relatively widely available ones like YouTube, have, to a large degree, changed how modern society learns. Each day, we see computers being used in ways that creatively enhance educational opportunities. For example, social networking sites are helping students connect with their peers to complete homework. Many educational institutions routinely use intranet sites and classroom management systems to share course materials. Internet tutorial videos covering nearly all high school subjects flourish on YouTube and Google Video. Innovative and creative educational computer applications abound. Yet, educational approaches harnessing these resources are relatively limited. A technologic disconnect exists between these applications and teachers who routinely do not use them. To address these issues we created a course within our Teachers on Sabbatical program that give teachers the tools and skills they need to integrate computer science into their teaching and curricula.

The Teachers On Sabbatical program at the College of Staten Island is a new and unique program tailored toward professional development of in-service teachers during their sabbatical year. Program focus has been to increase teacher awareness of new and relevant educational practices. To this end, teachers were exposed to a breadth of educational techniques and issues, including a focused approach to computer science in the classroom. Community building was encouraged using team projects to foster relationships that extended beyond the scope of the program. A core course in this program was focused on giving teachers a breadth of experience in education related computer applications, internet skills, and programming using games and robotics. The basis for the course curriculum was to introduce teachers to computer software, hardware,
and techniques that could aid in classroom management, enhance curriculum development, and help introduce computer science to their students.

Even though anecdotal evidence showed that most teachers registered for our program used word processing software to create exams and curricula materials, the comfort level that would enable these teachers to use new and/or unfamiliar software was not very high. By extending the teacher's word processing skills we first were able to mitigate the teacher's "fear of the computer". Once comfortable, teachers were shown how to use excel to maintain a gradebook, and program using Scratch, Alice and NXT robots. They were also encouraged to create curriculum using this software. Assistive technologies were addressed using software packages such as Dragon Naturally Speaking and Inspiration.

The Department of Education of the City of New York has installed interactive Smart Boards in many classrooms. Teachers are expected to use this technology with little or no training. To overcome their reluctance to embrace these, we made both the boards, and software, available in a nonthreatening environment. Teachers were encouraged to view Smart Board online tutorials and incorporate the board's use in curriculum development.

The focus of this poster will be to show the different projects and curriculum used in this course to create a technologically savvy educator.
INTEGRATING MERKLE-HELLMAN KNAPSACK CRYPTOSYSTEM INTO UNDERGRADUATE COMPUTER SCIENCE CURRICULUM

POSTER SESSION

Yana Kortsarts
Computer Science Department, Widener University, Chester, PA
Tel: 610-499-4367
URL: http://www.cs.widener.edu/~yanako
ykortsarts@mail.widener.edu

Yulia Kempner
Computer Science Department, Holon Institute of Technology, Holon, Israel
Tel: 972 - 3-5026739
URL: http://www.hit.ac.il/departments/computers/staff/yuliak/yuliak.htm
yuliak@hit.ac.il

ABSTRACT

In classical symmetric or private-key cryptosystems the encryption and decryption keys are either the same or can be easily found from each other. A new type of cryptosystem, call a public-key cryptosystem was invented in the 1970s. In a public key cryptosystem the fact that one knows how to encrypt the message does not mean that it can be easily decrypted. One of the earliest public-key cryptosystem was proposed by R.C. Merkle and M.E. Hellman in 1978. The Merkle-Hellman cryptosystem is based on the subset sum problem, a special case of the knapsack problem.

There have been many variants of knapsack cryptosystems and the history of their development and the history of the development of their cryptanalysis are very important. We present a way to integrate Merkle-Hellman knapsack cryptosystem into introductory cryptography course and into introductory core computer science courses. Ideas for the undergraduate student projects are proposed and discussed.

Poster will present a short history of the development of public-key cryptology. Poster will focus on three types of knapsack cryptosystems: Additive Knapsack Cryptosystem; Multiplicative Knapsack Cryptosystem; and Multiply-Iterated Knapsack

* Copyright is held by the author/owner.
Cryptosystem. Each cryptosystem will be illustrated by the examples and poster will discuss ways to integrate these knapsack cryptosystems into the following undergraduate computer science courses: Introduction to Computer Science, Cryptology, and Analysis of Algorithms. Poster will also discuss ideas for undergraduate student projects.
medPing: USING DATA STRUCTURES WITH EMBEDDED MEDICAL DEVICES

POSTER SESSION

Mark D. LeBlanc
Computer Science
Wheaton College
Norton, MA 02766
508.286.3970
mleblanc@wheatoncollege.edu

The future of monitoring our health is about to radically change. Embedded devices providing real-time monitoring of one’s health (e.g., blood pressure, glucose level, heart rate, DNA mutation) are an active area of current research and development and will be increasingly commonplace. Today, individuals with diabetes wearing a particular glucose meter who drive a Lincoln sedan can have their meter warn the driver of a negative slope in glucose levels, thereby warning the driver in real time of a potential blood sugar “low” while driving [3]. This poster presents an example that exposes students in their initial year to a set of lab and programming assignments featuring these state-of-the-art changes in medicine in order to motivate them to participate in the exciting innovations occurring in the boundary of medicine and software.

medPing is a collection of C++ classes that provide an Application Programming Interface (API) for building applications that interface between (currently emulated) embedded medical chips and an iPhone. Has your heart rate been too high for too long? Your cell gets a “ping”. Is your blood pressure falling too low while you sleep? “Ping.” Is your DNA sequence in a skin cell gene significantly different from a previous sequence? “Ping” (your oncologist too). The use of medPing is part of a larger effort at Wheaton College (Norton, MA) to repeatedly expose computer science students to the exciting and necessary role of computing in biology [2]. The emergence of bioinformatics, health IT, and research and development at “big pharma” are generating sustainable options for employment and graduate programs for computer science students who can apply their computational skills to the challenges of health research and practice.

Used in the Data Structures course on the Mac OS X platform in Spring 2009 and again in Spring 2010, a number of medPing labs and assignments introduce the classes that emulate an embedded medical chip under the skin of a person. All output from the

* Copyright is held by the author/owner.
medPing API appears on an iPhone console screen. Originally, students practice reading the API and experimenting with medPing methods. Two labs and two assignments lead students to (1) investigate the relation between a method’s behavior and its implementation details, (2) create apps that monitor and analyze a patient’s vital signs, and (3) enhance the class functionality to store a patient’s medical history over time. While medPing was especially successful as a teaching model early in the course, the course does not focus on medPing throughout the semester. The Data Structures’ syllabus (our second course in the major sequence taught using C++) is full of critically important topics (e.g., exposure to multiple data structures, recursion, pointers, and the C++ Standard Template Library), thus we have not at this point attempted to force medPing further than its originally intended purpose of providing a motivating introduction to objects and their associated APIs [1]. That said, it has proved successful in providing a set of objects for reference purposes, e.g., “Recall the child classes of the medPing object….” or “why did we need destructors in our queue of medPing objects?”

MedPing methods offer levels of abstraction that buffer students from the iPhone’s Objective C code that handles input and output from the iPhone console. Vital sign emulators provide bell curve random number generators for realistic values, for example, a range of (normal) systolic blood pressure values between 90 and 140 mmHg. A non-Mac version is provided for those students who cannot access the campus Mac labs in off hours; in this case all output appears on a standard output console.

REFERENCES


Pedagogical research has already established that couching lessons within the use or design of computer games can be an effective method of instruction. Text adventure games are a particularly excellent vehicle for exploring many Data Structures topics. In these games, the interface is console-based and involves a simple input-response cycle that is easy for novice programmers to master. The game's player uses typed commands to move around a series of locales, each of which is described upon entering. There are physical objects in each location that can be inspected, picked up and carried, dropped, and sometimes used in some way. Meeting the requirements of a global objective causes the game to end. Beyond this, student creativity can add layers of plot complexity and puzzles, humor, and originality without requiring advanced programming expertise.

Students can build this project in multiple phases stretching over half of a semester. The game concept naturally lends itself to the incorporation of multiple object class design tasks at the beginning of the semester, requiring one object class to represent the locations in the game world, one class to represent the physical objects that can be manipulated, one to represent the player, and one to represent the game world as a whole.

Students next come to recognize the usefulness of class inheritance when they see that since the player class carries objects and the location class also contains objects - and the process of accessing and manipulating those objects is the same in each case - they need to create a superclass for both players and locations. All object-containing and -manipulating code is encapsulated there, while the location class extends this to include such things as a description and exit and the player object class contains implementations of hunger, points acquired, etc.

The classic linked-list implementation project also folds into this game as the object-container superclass is implemented as a linked list, with methods for traversal (list all of the objects in the room / in the player's inventory), search (find a specific object by name in order to inspect it), insertion (pick up an object and add it to the inventory, or
drop an object and add it to the room), and removal (remove an object from the room or drop it from the inventory).

A final class-wide competition in which games are played by friends and classmates and judged on complexity, creativity, and ease of play provides a capstone which doubles as a motivator to make the game world and plot original and exciting, while providing the larger community with exposure to Computer Science projects as an expression of creativity.
RUNNING A COMPUTER SECURITY COURSE:

CHALLENGES, TOOLS, AND PROJECTS

POSTER SESSION

Vladimir V. Riabov and Bryan J. Higgs
Department of Mathematics and Computer Science
Rivier College
Nashua, NH, 03060
603 897-8613
vriabov@rivier.edu and bryanhiggs@gmail.com

Challenges and effective ways of instruction in computer security classes (security tools, technology overviews, research projects, virtual labs, and Web resources) are discussed with examples of lecture notes, OPNET™ lab assignments, homework study cases, and projects available on the instructors’ Websites. The project-based approach motivates students in exploring computer-security techniques, writing technology overviews, and conducting research, and provides them with knowledge, instructions, and hands-on experience.

Rivier College offers courses on computer security at both undergraduate and graduate levels in Computer Science programs. These courses are introductions to the methods, algorithms, and tools of computer system security. Topics cover both the theoretical and practical aspects of security including cryptography, protocols, standards, and security implementation. An important part of the courses are surveys of actual techniques used by hackers to attack systems.

The authors’ websites are the gateway to courses, publications, and numerous resources on the Internet (through the World Wide Web, Secure Shell, and Secure FTP). Each course has a portal to syllabi, assignments, lecture slides and notes, tools, software installation instructions, tutorials, lab manuals, examples of project papers, research reports, Internet links, lists of recommended readings, etc.

The classes cover security concepts; history of cryptography; theory of sets, permutations, combinations, and probability; number theory and modular arithmetic; classical cryptosystems; symmetric block ciphers; public key cryptography; an overview of message authentication codes, hashes, and message digests; principles of authentication; Web security and privacy for users; tunneling and virtual private networks (VPNs); and malware. The instructors discuss with students secure ways of sharing the

* Copyright is held by the author/owner.
network resources, issues of confidentiality, medical and personal information security on the Internet, and protection from electronic spam. This overview helps in introducing encryption algorithms such as the RSA Public-Key encryption algorithm.

A student can try to solve the problems by a simple experimentation with the Java Applets Tools especially designed for these courses. Students use these tools to create and decipher simple shift substitution ciphertexts, MonoAlphabetic substitution cipher, the Playfair and Vigenère ciphers, as well as to explore modular arithmetic and message digests. The tools also are used in reviewing the concepts of probabilities and combinatorics.

The course assignments include three homeworks, one lab, midterm and final exams, and a project paper that covers in depth one of the computer security technologies. Every class starts with a brief discussion of a topic that is related to the homework exercises. After this "warm-up" introduction, the instructor offers a discussion on the main topic and asks students for a feedback on lecture materials and their arguments on selecting a competitive strategy for the problem analysis and development. These discussions help students to focus on the main point of the class session and stay active in class. After cracking a couple of simple short ciphers, students are asked to explore how cryptographers might actually crack classic ciphers. The students are encouraged to use various components of the Java applet while working on this assignment. They start by exploring a MonoAlphabetic Substitution Cipher (e.g., the oldest Caesar cipher) that maps individual plaintext letters to individual ciphertext letters, on a 1-to-1 unique basis. To encipher a message, students simply take each letter in the plaintext, find that letter in the Plaintext row, and substitute the corresponding letter immediately below it, in the Ciphertext row.

Finally, students examine the Letter Frequency Analysis approach based on assumptions that the plaintext consists of characters written in some known natural language (e.g., English), and the frequency of letters in a typical piece of text in that language is known. After the concept review and exploration with the Java Applets tool, students are asked to study the two ciphertexts: Ciphertext-1 (3 pages, 620 words, 2,685 characters, and 128 lines), where the original word spacing, punctuation, and style have been retained; and Ciphertext-2 (46 pages, 25,955 words, 103,818 characters, and 2,596 lines), where word spacing and punctuation have removed, and the text has been organized in groups of four letters. This makes it more difficult to decipher the ciphertext using the context that those clues (word spacing and punctuation) provide. Usually it takes more than 6 hours for a student to decipher these ciphertexts, using a variety of techniques and tools (e.g., one student wrote some custom UNIX scripts and a standard UNIX dictionary to help with the mechanics of the solution).

The last assignment gives students an opportunity to review the theory of probabilities that plays an important part in many areas of security. It covers four topics: “CIA Hiring”; “Brobdingnag Battles”; “Delta Force”; and “Ethnic Dispute”. In an attempt to overcome the all too common “Math-phobia” of students, some standard statistical/probability problems were re-cast using scenarios that were more “security-related”.

Several classes were designed as computer labs that help students in exploring the network-security study cases and finding ways of solving them. The OPNET IT Guru™
Academic software package was used for studying firewalls and virtual private networks. Using this knowledge and skills, students develop their own lab projects and include virtual lab techniques into their research projects related to various network security protocols, such as the Diffie-Hellman asymmetric key agreement protocol and RADIUS protocol.

Students are encouraged to conduct research and write project papers on modern computer-security technologies. They select projects that would be beneficial for their careers and valuable for companies and the community. Usually, students demonstrate their project portfolios during job interviews. Such demonstration of their actual professional skills in computer security helps students in finding a job immediately after the graduation. Many projects are implemented in local companies and the community, e.g., “Secure Wi-Fi Technologies for Enterprise LAN Network”, “Steganography and Steganalysis”, “Intrusion Prevention System”, “Security and SQL Injections”, “Virtual Private Networks”, “Firewalls Overview”, “RADIUS Protocol”, and “Secured Communication in Java”. Students are encouraged to submit summaries of their research projects to professional journals and magazines.

The authors believe that this project-based, tool-exploration, and virtual-lab approach can be effectively applied to future courses of a similar nature in academia.
ABSTRACT

When I was young and learning to fish, an old angler once told me something that was, on one hand, trivial. On the other hand, it was almost prophetic. He said, “If you want to catch fish, you have to go where the fish are”. For many years I have used LISTSERVs with all my classes to enable open conversation among students and me. Over the years, it has become more and more difficult to manage LISTSERVs for me at my college. Also, although still very useful, they are archaic as communication means. My reason for using LISTSERVs has always been to maximize communication among participants in a course.

Today, on the Internet, there are many types of social networking sites. Students have Facebook [1], Twitter [2], and MySpace [3] accounts galore. “If you want to catch fish, go where the fish are”. As an experiment during the Fall 2009 semester, I used Twitter as our way of intra-class networking. It was a dreadful failure. I have always stayed away from sites like Facebook because I have not wanted to intrude upon students’ personal lives. In the past, accounts tended to be so open that to be a ‘friend’ meant that a person saw all or most of whatever was being posted by friends. This is no longer entirely true. There is a Facebook tool called a ‘Group’ that allows people to join with no requirement to be friends, thus providing the firewall between college and private life.

The only thing that a person needs is a Facebook account, which is free and easy to set up. During the Spring 2010 semester, I will run an experiment with Facebook groups to enable active intra-class discussion. I have created three Facebook groups named CSCI 100, CSCI 246, and CSCI 344. As part of Homework 1, I will require students to join their course group. None of the students’ private information will be accessible to any group member. But, all group members can contribute to the group through posts to its
wall, participate in any discussions, view scheduled events such as study sessions and exams, and even post images or videos. I will be the administrator of each group. By the time CCSCNE 2010 takes place, I will have good, empirical information about using Facebook groups to enhance communication among participants in classes. I will present this information via a poster during the Faculty Poster session.

REFERENCES
A GENERAL EDUCATION COURSE - “INTRODUCTION TO
ALGORITHMIC THINKING” - USING VISUAL LOGIC©

POSTER SESSION

Namita Sarawagi
Dept. of Mathematics and Computer Science
Rhode Island College
600 Mt. Pleasant Avenue
Providence, RI 02908
(401) 456-8038
nsarawagi@ric.edu

“Introduction to Algorithmic Thinking” is a new, General Education, Computer Science course at Rhode Island College, taught for the first time in Fall 2009. This 100-Level course [1] provides an alternative, to students interested in a CS course, beyond the usual computer literacy course - “Introduction to Computers”. It introduces, algorithmic thinking, the study of step-by-step procedures for solving problems, with simple applications from various disciplines. The major topics covered are: algorithms, flowcharts, top-down design, selection, repetition, and modularization [1].

Visual Logic© PGS Systems (www.visuallogic.net) is an interactive, graphical flowchart software to learn programming fundamentals [5]. (I was introduced to this software in a tutorial session at CCSCNE 2009 [4]). It can be used to create the logic of a problem’s solution, as well as execute it. It also includes a LOGO type Graphics package. This software provides the perfect tool for this Gen. Ed. course as it is extremely user-friendly, has minimal syntax and is simple for the student clientele of this course. The book, “A Guide to Working with Visual Logic”, by Thad Crews and Chip Murphy [2], can be bundled with the Visual Logic© software [5]. Thad Crews is one of the developers of this software.

The challenges, in teaching “Introduction to Algorithmic Thinking” as a Gen. Ed. course, are to present each topic (classroom examples) using real-world and interesting problems and to find a wide variety of problems (programming assignments) for students to apply the concepts to, because of the varied student clientele that this course is targeted towards. Also, these problems and examples need to be simple enough for this introductory level course. The book for Visual Logic [2], mentioned above, and the book “Just Enough Programming Logic and Design” by Joyce Farrell [3], are great resources.

* Copyright is held by the author/owner.

250
for such examples and problems. Both these books are excellent choices as textbooks for this course. Most of the examples/problems presented in this course, were taken from these books [2, 3]. Some of these problems, the concepts involved in their solutions/algorithm, and the discipline they fall under are listed in the following table.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Concepts in Algorithm</th>
<th>Discipline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mad Libs *</td>
<td>Input and Output, Variables</td>
<td>Language/Game</td>
</tr>
<tr>
<td>Guessing Game (secret number)</td>
<td>Random, Conditionals</td>
<td>Mathematics/Game</td>
</tr>
<tr>
<td>PayRoll (Calculate Pay / Overtime Pay)</td>
<td>Conditionals, Arithmetic Expressions</td>
<td>Business/Mathematics</td>
</tr>
<tr>
<td>Winning Prizes – Lottery *</td>
<td>Random, Conditionals, Boolean Expressions</td>
<td>Game</td>
</tr>
<tr>
<td>Multiplication Tables</td>
<td>Loops</td>
<td>Mathematics</td>
</tr>
<tr>
<td>Grocery CheckOut Bill</td>
<td>Sentinel Controlled Loops</td>
<td>Business/Mathematics</td>
</tr>
<tr>
<td>Average, Largest, Smallest Values in a List</td>
<td>Loops, Accumulator, Conditionals</td>
<td>Computer Science / Mathematics</td>
</tr>
<tr>
<td>Line Drawings</td>
<td>Nested Loops</td>
<td>Art</td>
</tr>
<tr>
<td>Logging In – (matching usernames/passwords)</td>
<td>Arrays, Reading Input from a File</td>
<td>Computer Science / Databases</td>
</tr>
<tr>
<td>Baseball Statistics</td>
<td>Arrays, Reading Input from a File</td>
<td>Sports</td>
</tr>
<tr>
<td>Drawing Regular Polygons/Rotating Shapes</td>
<td>Graphics Commands, Procedures</td>
<td>Art/Drawing/Geometry</td>
</tr>
</tbody>
</table>

*The Mad Libs and Lottery problem are from Joyce Farrell’s book ([3]). The others are from the Visual Logic book ([2]).

The logic and solutions of many of the problems above, and others, will be on display in this poster. The poster will attempt to provide a visual perspective of the concepts covered in the entire course through problems and their visual solutions created using Visual Logic©. One such example Rotating Flags Problem [2] (on Page106), is a graphics problem which illustrates the concepts of procedures/modularization and top-down design, using the Visual Logic© Graphics Commands

**Rotating Flags Problem:**

Draw 8 rotating flags symmetrically in a circle as follows

![Snowflake Image]
REFERENCES


A DIJKSTRA'S ALGORITHM SHORTEST PATH ASSIGNMENT

USING THE GOOGLE MAPS API

POSTER SESSION

James D. Teresco
Visiting Associate Professor of Computer Science
Mount Holyoke College
South Hadley, MA 01075
(413) 538-3253
jteresco@mtholyoke.edu

An assignment involving a hands-on implementation of Dijkstra's Algorithm for computing single-source shortest paths is a useful supplement to a class discussion and readings about graph structures and algorithms in a data structures course. Ideally, the graph data used needs to be small enough to be manageable, but large enough to be interesting. This might consist of a small road system, airline schedules, or even the layout of a campus or building. This poster presents a variation on a Dijkstra's Algorithm assignment that uses the Google Maps API [2] and highway routing data from the Clinched Highway Mapping (CHM) Project [3] that allows students to compute and display, in Google Maps, real routings between locations.

The CHM Project allows road and travel enthusiasts to track the highways on which they have travelled. Project collaborators have gathered and continue to gather data about a variety of highway systems in North America and Europe for use in the project. The data consists of a set of “waypoints” for each highway included in the project. A waypoint consists of a place name and its coordinates (latitude/longitude). Most waypoints are at major intersections, but others are included primarily to help define the shape of the route more accurately. The granularity is not sufficient to provide a perfect representation of a highway's route, but is fine enough to be reasonably accurate and coarse enough, leaving out route details and insignificant intersections, to avoid overwhelming the tools and the users. The CHM data is used by a Highway Browser that allows project participants to view the highway's route so they can create lists of the highway segments they have travelled. These lists are then used to generate statistics and maps of the participants' travels.

For the purposes of this assignment, the mapping data is treated in its raw form: the individual highway data files (lists of waypoints) that make up a highway system. The
highway data files are used as input to the support programs developed by the author to generate a set of composite highway data for a given highway system. The composite highway data is stored in a format convenient for use in this assignment, and is readily processed by student programs to construct a graph structure for a highway system. During the Fall 2009 semester, the highway systems made available to students included small systems that are useful during development and debugging such as the Interstate highway system in Hawaii (47 waypoints/vertices, 48 connections/edges) and the Yukon Provincial highway system (351 waypoints, 354 connections), and larger systems such as the set of all Interstate, U.S., and state highways in New York state (7265 waypoints, 8416 connections). Several other highway systems are available from the CHM Project, with many more under construction.

The CHM data provides real and appropriately-sized input for an academic study of Dijkstra's Algorithm. The Google Maps API provides a fun and useful way to visualize the data and the results. The Google Maps API is free to use, but a domain-specific API key must be obtained and specified in the plotting program. Points and routes can be plotted in a browser window, much like that done by Google Maps and other mapping sites. The students' Java programs produce output files consisting of points to be plotted, with connections optionally drawn between successive points. Students upload their output files to the course web server, where they can visualize their results in map form. Students are encouraged to examine the Javascript program that generates the maps, but they need not modify it.

Students are provided with a starter program with a few methods left as stubs to be filled in. Students are required to implement methods that construct an appropriate graph structure, print the graph data to the screen or into files suitable for display using the mapping scripts, and compute shortest paths using Dijkstra's Algorithm and print them to the screen or into mapping data files. Students base their implementation of Dijkstra's Algorithm on the one provided as an example by Bailey's free textbook, *Java Structures* [1], the primary text for the course. Students are also provided with input data format descriptions, output format requirements, and instructions to upload files and view results in the Google Maps environment.

Student reaction to the assignment was very positive. They were excited to be able to work on a program that would produce results that they could display with the mapping scripts. Students were allowed to work in groups of 2 or 3 for this assignment, and were given about a week and a half to complete the assignment. Students formed 4 groups, 3 of which made submissions that were complete and correct or nearly so.

This assignment will continue to be improved for use in future offerings of data structures. Additional development will improve the pre-processing tools and further automate the process of displaying mapping data. Improved processing of the CHM data will result in more user friendly highway data. For example, the junction of New York state routes 29 and 30 would currently receive the waypoint name @NY29;NY30@NY30;NY29, a simpler name such as NY29/NY30 could be generated automatically and would be easier to work with. Waypoints that are not at important intersections but are included only for route shaping (typically prefixed by + in the CHM data) could be folded into adjacent edges. Improvements to the map display will allow more control over the points to be plotted and will further automate the visualization.
process. Ideally, a student will be able to upload results through a web form rather than separate steps to copy to the server and to enter an appropriate URL, as is currently required.

Support materials, handouts, and sample solutions are available on request from the author.

REFERENCES


OUR EXPERIENCES INCORPORATING ROBOTICS INTO OUR SERVICE COURSE

POSTER SESSION

David Toth
Computer Science Department
Merrimack College
315 Turnpike Street
North Andover, MA 01845
978-258-1208
david.toth@merrimack.edu

ABSTRACT

A number of colleges and universities are using robots as a focus in introductory computer science courses to try to stimulate students' interest in computer science. Several colleges have integrated robots into their CS1 courses or even built an entire course around robots [1, 2, 3, 4]. Other colleges have integrated robots into CS0 courses [5, 6]. In this poster, we present the experiences we had when incorporating robotics into our service course, which is designed to be neither a CS0 nor a CS1 course.

Our course, Introduction to Information Technology, aims to introduce students to the concepts in information technology and their uses in the workplace. Because of breadth of this subject, we are only able to devote 1-1.5 weeks to any particular topic, including introducing students to computer programming. The course is almost entirely populated by students who will not major in computer science, although on rare occasions, a student taking this course will continue on to CS1 and the computer science major. Previously, we introduced programming with Scratch and also had students play a game called LightBot, which requires students to “program” a virtual robot to light up tiles [7, 8]. Many of the students had a lot of success with LightBot and enjoyed it. Conversely, a large portion of the students did not appear to be engaged with Scratch.

This year, following the success of using LightBot, we considered alternate methods to introduce programming and chose to use robots in the course to introduce students to computer programming. We used the Scribbler robots from Parallax, Inc. in three of our four sections of Introduction to Information Technology this fall [9]. The Scribbler robot comes with a graphical programming environment that can be used by students instead of using the Basic Stamp editor that also comes with the robot. Students were taught how
to use the graphical programming environment to make the Scribblers move, make
decisions using if/else statements, repeat tasks with a while loop, and detect obstacles
using the robot’s sensors. The students were encouraged to play with the robots for the
50 minute class period to get comfortable with them. They were told that during the
following class, which was 100 minutes, their assignment would be to program the robots
to navigate a simple U-shaped maze. The students were told that their robot should be
able to be placed at either entrance of a maze by the professor and be able to enter the
maze and exit out the other side.

We had mixed results and encountered several unexpected challenges when
conducting the lab. The success rate of the students was low and few students were able
to complete the exercise the way we had hoped they would. This may have been due to
some of the many problems we encountered while conducting this lab. However, the
reason for changing the vehicle for introducing programming was to try to engage more
students and a greater proportion of the students appeared to be engaged in trying to make
the robot traverse the maze than had been engaged by Scratch last year. We also noticed
that several students who had not seemed to be particularly engaged during the course
were working hard on the lab and appeared to be enjoying themselves. Given that we
were primarily looking to engage the students more successfully than last year, the lab
may have worked better than the success rate indicated.

REFERENCES

[1] Institute for Personal Robots in Education, Institute for personal robots in
education, http://www.roboteducation.org/schools.html, retrieved October 26,
2009.

[2] Union College, Computer Science 2009-2010,
http://www.union.edu/applications/catalog/deptView.php?mode=catalog&year=2

[3] Lauwers, T., Nourbakhsh, I, and Hamner, E., CSbots: Design and Deployment of
a Robot Designed for the CS1 Classroom, Proc. 40th SIGCSE 2009, 428-432,
2009.


[5] Connecticut College, Connecticut College: course catalog: computer science,
http://aspen.conncoll.edu/camelweb/?fuseaction=registrar&circuit=cc&function=
course&action=computerscience, retrieved October 26, 2009.

[6] Cliburn, C., CS0 Course for the Liberal Arts, Proc. 37th SIGCSE 2006, 77 – 81,
2006.

October 26, 2009.

October 26, 2009.
ON SLIDER PUZZLE PROJECTS WITH .NET COLLECTION CLASSES

POSTER SESSION

Linda Wilkens
Department of Mathematics and Computer Science
Providence College
Providence, RI 02918
401 965-1896
lwilkens@providence.edu

ABSTRACT

This poster presentation reports classroom experience using Sam Loyd’s Sliding Tile Puzzle for student programming projects which use .NET collection classes both to store game tree nodes, and to support user and computer generated solutions to the puzzle. This project offers the students the opportunity to develop a non-trivial application, to design classes, to reuse data structures developed during the semester, and to leverage existing data structures including several of the .NET collection classes.

The sliding tile puzzle has been used extensively for programming projects. A description of the puzzle and a discussion of its history can be found on several websites[1, 2]. The advantages of the puzzle for student projects is that the computational requirements of a module to find a computer-generated solution to a particular puzzle can be controlled by designating the difficulty of the puzzle; i.e., by creating a solvable puzzle by making random valid backwards moves from the original solution state.

The students can first implement an application that allows users to solve the puzzle, and then try to develop their own heuristics for generating computer solutions. Then finding a computer solution can be guided by a directed search, such as a best-first search through the generated tree of game states, where the root is the initial tile configuration, and the fringes are the unexplored reachable states. This search requires the students to contain the generated node states in a structure that can be searched efficiently for a node with a particular state and that also allows for quickly finding the most promising state from among the unexplored fringe states. The students are given an outline of the A* search algorithm, and are led through a set of labs which result in a working but minimal version of the program. From this point they need to implement the module for computer solution, exploring various alternatives for storing both the open and closed lists and the

* Copyright is held by the author/owner.
implicit tree that is generated as the algorithm progresses. Naïve data structure choices lead to implementations that can solve only simple puzzles, whereas using more sophisticated alternatives, if implemented correctly, give better results.

Because the description of A* search discusses an open list, a closed list, and a game tree, students frequently start with a very literal interpretation of the algorithm and choose a linear container such as the .NET Collection class ArrayList or the templatized List<T>. Most students quickly realize that if they keep their lists in sorted order they can use a binary search rather than a linear search for a particular element, thus speeding up solution execution. A major goal of the project is to have the students then replace their linear list with either a hashtable or with .NET’s Dictionary class which takes two template parameters, one for the type of the item, and one for the type of the key. Using either a hashtable or an instance of the Dictionary class is best supported by implementing the IEquatable interface, in order to allow two different nodes in the game tree, which may have different costs and different parents in the tree but which represent the same tile configuration, to be considered equal. Another goal of the project is to have students use a priority queue to support quickly finding the most promising fringe node to explore next. A related challenge for the students is to link the priority queue with the other data structures, thus leveraging the use of reference variables and exploiting the advantages of both types of structures.

Additional issues addressed in the project include file I/O, code instrumentation, using profiling for performance analysis, comparing algorithm analysis with timing results, and user interface design.

The choice to develop in .NET lead to a language decision: the projects discussed here were developed in C#. However, the author has used this same project in previous semesters, combining Visual C++ with Standard C++ Library container classes. The poster will compare the two approaches, and will list advantages and disadvantages of each. The website www.codingquest.com lists resources used during the course and links to html-ized versions of the students’ slider puzzle project presentations.

REFERENCES


# INDEX OF AUTHORS

<table>
<thead>
<tr>
<th>Author</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abelson, H</td>
<td>8</td>
</tr>
<tr>
<td>Allen, F</td>
<td>5</td>
</tr>
<tr>
<td>Allen, M</td>
<td>100</td>
</tr>
<tr>
<td>Alnajjar, H</td>
<td>1</td>
</tr>
<tr>
<td>Anewalt, K</td>
<td>144</td>
</tr>
<tr>
<td>Armstrong, T</td>
<td>211</td>
</tr>
<tr>
<td>Assiter, K</td>
<td>214</td>
</tr>
<tr>
<td>Bailie, F</td>
<td>15</td>
</tr>
<tr>
<td>Baird, B</td>
<td>47, 100</td>
</tr>
<tr>
<td>Baliga, G</td>
<td>174</td>
</tr>
<tr>
<td>Beidler, J</td>
<td>155</td>
</tr>
<tr>
<td>Bi, Y</td>
<td>155</td>
</tr>
<tr>
<td>Blank, G</td>
<td>181</td>
</tr>
<tr>
<td>Bloch, S</td>
<td>218</td>
</tr>
<tr>
<td>Brusilovsky, P</td>
<td>74, 224</td>
</tr>
<tr>
<td>Cassel, L</td>
<td>74, 221, 224, 227</td>
</tr>
<tr>
<td>Chang, G</td>
<td>191</td>
</tr>
<tr>
<td>Chang, M</td>
<td>8</td>
</tr>
<tr>
<td>Chung, C</td>
<td>47, 100</td>
</tr>
<tr>
<td>Cliburn, D</td>
<td>210</td>
</tr>
<tr>
<td>Connelly, R</td>
<td>34</td>
</tr>
<tr>
<td>Corliss, M</td>
<td>159</td>
</tr>
<tr>
<td>Cotler, J</td>
<td>40</td>
</tr>
<tr>
<td>Coulter, P</td>
<td>230</td>
</tr>
<tr>
<td>Danner, N</td>
<td>100, 152</td>
</tr>
<tr>
<td>Dansdill, T</td>
<td>205</td>
</tr>
<tr>
<td>Davis, J</td>
<td>159</td>
</tr>
<tr>
<td>Delcambre, L</td>
<td>74, 224</td>
</tr>
<tr>
<td>Dickson, P</td>
<td>104, 233</td>
</tr>
<tr>
<td>Egan, M</td>
<td>198</td>
</tr>
<tr>
<td>Ellis, H</td>
<td>152</td>
</tr>
<tr>
<td>Evans, J</td>
<td>181</td>
</tr>
<tr>
<td>Fax, W</td>
<td>224</td>
</tr>
<tr>
<td>Fischbach, A</td>
<td>205</td>
</tr>
<tr>
<td>Fox, E</td>
<td>74, 224</td>
</tr>
<tr>
<td>Furcy, D</td>
<td>159</td>
</tr>
<tr>
<td>Furuta, R</td>
<td>74, 224</td>
</tr>
<tr>
<td>Gans, P</td>
<td>235</td>
</tr>
<tr>
<td>Garcia, D</td>
<td>74, 224</td>
</tr>
<tr>
<td>Goldschmidt, D</td>
<td>121</td>
</tr>
<tr>
<td>Grissom, S</td>
<td>14, 102</td>
</tr>
<tr>
<td>Hart, D</td>
<td>76, 136</td>
</tr>
<tr>
<td>Heines, J</td>
<td>66</td>
</tr>
<tr>
<td>Higgs, B</td>
<td>245</td>
</tr>
<tr>
<td>Hildreth, E</td>
<td>60</td>
</tr>
<tr>
<td>Hislop, G</td>
<td>74, 152, 224</td>
</tr>
<tr>
<td>Hoffman, M</td>
<td>205</td>
</tr>
<tr>
<td>Imberman, S</td>
<td>237</td>
</tr>
<tr>
<td>Inguva, S</td>
<td>221</td>
</tr>
<tr>
<td>Joiner, D</td>
<td>191</td>
</tr>
<tr>
<td>Kempner, Y</td>
<td>239</td>
</tr>
<tr>
<td>Klibaner, R</td>
<td>237</td>
</tr>
<tr>
<td>Kortsarts, Y</td>
<td>205, 239</td>
</tr>
<tr>
<td>LeBlanc, M</td>
<td>241</td>
</tr>
<tr>
<td>Lederman, T</td>
<td>129</td>
</tr>
<tr>
<td>Lee, S</td>
<td>60</td>
</tr>
<tr>
<td>Leete, D</td>
<td>230</td>
</tr>
<tr>
<td>MacDonald, I</td>
<td>121</td>
</tr>
<tr>
<td>Malan, D</td>
<td>111</td>
</tr>
<tr>
<td>Mallozzi, J</td>
<td>167</td>
</tr>
<tr>
<td>Marion, B</td>
<td>15</td>
</tr>
<tr>
<td>Mastropalo, K</td>
<td>78</td>
</tr>
<tr>
<td>Meinke, J</td>
<td>xi</td>
</tr>
<tr>
<td>Menzin, M</td>
<td>66</td>
</tr>
<tr>
<td>Meyer, M</td>
<td>78</td>
</tr>
<tr>
<td>Michaud, L</td>
<td>243</td>
</tr>
<tr>
<td>Mitra, S</td>
<td>93</td>
</tr>
<tr>
<td>Morelli, R</td>
<td>152</td>
</tr>
<tr>
<td>Morreale, P</td>
<td>191</td>
</tr>
<tr>
<td>Moturi, S</td>
<td>227</td>
</tr>
<tr>
<td>Mustafaraj, E</td>
<td>8</td>
</tr>
<tr>
<td>O’Rourke, J</td>
<td>121</td>
</tr>
<tr>
<td>Pietraszek, L</td>
<td>159</td>
</tr>
<tr>
<td>Polack-Wahl, J</td>
<td>144</td>
</tr>
<tr>
<td>Potluri, S</td>
<td>224</td>
</tr>
<tr>
<td>Proulx, V</td>
<td>70</td>
</tr>
<tr>
<td>Rao, T</td>
<td>93</td>
</tr>
<tr>
<td>Riabov, V</td>
<td>245</td>
</tr>
<tr>
<td>Rizzo, J</td>
<td>40</td>
</tr>
<tr>
<td>Robinson, J</td>
<td>174</td>
</tr>
<tr>
<td>Russell, I</td>
<td>1</td>
</tr>
<tr>
<td>Ryder, J</td>
<td>248</td>
</tr>
<tr>
<td>Sabin, M</td>
<td>26</td>
</tr>
<tr>
<td>Sands, M</td>
<td>181</td>
</tr>
<tr>
<td>Sarawagi, N</td>
<td>250</td>
</tr>
<tr>
<td>Shipman, F</td>
<td>224</td>
</tr>
<tr>
<td>Spezialetti, M</td>
<td>11, 100</td>
</tr>
<tr>
<td>Sturm, D</td>
<td>55</td>
</tr>
<tr>
<td>Name</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Teresco, J</td>
<td>253</td>
</tr>
<tr>
<td>Toth, D</td>
<td>73, 256</td>
</tr>
<tr>
<td>Turbak, F</td>
<td>8</td>
</tr>
<tr>
<td>Utell, J</td>
<td>205</td>
</tr>
<tr>
<td>van Delden, S</td>
<td>85</td>
</tr>
<tr>
<td>Welty, C</td>
<td>66</td>
</tr>
<tr>
<td>Whitfield, D</td>
<td>15</td>
</tr>
<tr>
<td>Wilkens, L</td>
<td>259</td>
</tr>
<tr>
<td>Young, A</td>
<td>6</td>
</tr>
<tr>
<td>Zelikovitz, S</td>
<td>55</td>
</tr>
</tbody>
</table>