The Journal of Computing Sciences in Colleges

Papers of the Sixteenth Annual CCSC Northeastern Conference
April 15-16, 2011
Western New England College
Springfield, Massachusetts

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 26, Number 6
June 2011
The Journal of Computing Sciences in Colleges (ISSN 1937-4771 print, 1937-4763 digital) is published at least six times per year and constitutes the refereed papers of regional conferences sponsored by the Consortium for Computing Sciences in Colleges. Printed in the USA. POSTMASTER: Send address changes to Jim Aman, CCSC Membership Chair, Saint Xavier University, Chicago, IL 60655.

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

THE CONSORTIUM FOR COMPUTING SCIENCES IN COLLEGES
BOARD OF DIRECTORS .................................................... ix
CCSC NATIONAL PARTNERS ........................................ x
FOREWORD ........................................................ x
  John Meinke, UMUC Europe
WELCOME TO CCSCNE 2011 FROM THE CHAIRS ....................... 1
  Stoney Jackson, Western New England College, Karl R. Wurst, Worcester State University
BOARD MEMBERS — CCSC NORTHEASTERN REGION ............... 2
2011 CCSC NORTHEASTERN REGIONAL CONFERENCE COMMITTEE . 2
REVIEWERS – 2011 CCSC NORTHEASTERN CONFERENCE ............ 3
WHAT CAN WE DO? RAISING THE NUMBERS OF WOMEN IN COMPUTING
— FRIDAY PLENARY SESSION ..................................... 5
  Jane Chu Prey, Microsoft Research
COMMUNITIES: OPEN SOURCE AND EDUCATION WORKING TOGETHER —
SATURDAY PLENARY SESSION .................................. 6
  Stormy Peters, Mozilla
GREENFOOT: INTRODUCING JAVA WITH GAMES AND SIMULATIONS —
PRE-CONFERENCE WORKSHOP / TUTORIAL PRESENTATION ...... 7
  Adrienne Decker, University at Buffalo, SUNY, Frances P. Trees, Drew University
AN OPEN SOURCE FIELD TRIP FOR FACULTY — PRE-CONFERENCE
WORKSHOP ........................................................... 10
  Gregory W. Hislop, Drexel University, Clif Kussmaul, Muhlenberg College, Mel Chua, Red Hat, Inc., Sebastian Dziallas, Olin College
<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>GETTING READY FOR A UNIT IN <em>ALICE 2.2</em>: WEEK 1</td>
<td>Eileen M. Peluso, Lycoming College</td>
</tr>
<tr>
<td>EVENT-PROCESSING IN <em>ALICE 2.2</em> — PRE-CONFERENCE WORKSHOP</td>
<td></td>
</tr>
<tr>
<td>TUTORIAL PRESENTATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>INTRODUCING CASE STUDY BASED TEACHING IN COMPUTING CURRICULA</td>
<td>Massood Towhidnejad, Salamah Salamah, Thomas B. Hilburn, Embry-Riddle Aeronautical University</td>
</tr>
<tr>
<td>— PRE-CONFERENCE WORKSHOP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPROVING THE STUDENT SUCCESS AND RETENTION OF UNDER ACHIEVER STUDENTS IN INTRODUCTORY COMPUTER SCIENCE</td>
<td>Carolee Stewart-Gardiner, Kean University</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>USING VISUAL LOGIC©: THREE DIFFERENT APPROACHES IN DIFFERENT COURSES</td>
<td>Dee Gudmundsen, The College of Saint Rose, Lisa Olivieri, Chestnut Hill College, Namita Sarawagi, Rhode Island College</td>
</tr>
<tr>
<td>- GENERAL EDUCATION, CS0, and CS1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ALIGNING GENERATIONS TO IMPROVE RETENTION IN INTRODUCTORY</td>
<td>Christian Roberson, Plymouth State University</td>
</tr>
<tr>
<td>COMPUTING COURSES</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPROVING STUDENT PERFORMANCE WITH AN EXTRA-CURRICULAR ROBOTICS CLUB</td>
<td>Charles N. Stevenson, Salem State University</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>INTRODUCTION OF VIRTUALIZATION IN THE TEACHING OF OPERATING SYSTEMS</td>
<td>Kevin Grammer, Jack Stolerman, and Beifang Yi, Salem State University</td>
</tr>
<tr>
<td>FOR CS UNDERGRADUATE PROGRAM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ROBOTICS ACROSS THE CURRICULUM WITH dLife — DEMONSTRATION</td>
<td>Grant Braught, Dickinson College</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>JAVA VISUALIZATION OF STUDENT-IMPLEMENTED OS ALGORITHMS —</td>
<td>Adam Fischbach, Widener University</td>
</tr>
<tr>
<td>DEMONSTRATION</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>iJava - AN ONLINE INTERACTIVE TEXTBOOK FOR ELEMENTARY JAVA INSTRUCTION — DEMONSTRATION</td>
<td>Robert Moll, University of Massachusetts</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>USING THE GOOGLE MAPS API WITH HIGHWAY MAPPING DATA AS A PEDAGOGICAL TOOL — DEMONSTRATION</td>
<td>James D. Teresco, Siena College</td>
</tr>
</tbody>
</table>
INTRODUCING BIOINFORMATICS INTO THE COMPUTER SCIENCE CURRICULUM—PANEL DISCUSSION .......................... 130
Mark LeBlanc, Wheaton College, David Rilett, John Russo, Michael Werner, Hongsheng Wu, Wentworth Institute of Technology

UNDERSTANDING NSF FUNDING OPPORTUNITIES — TUTORIAL PRESENTATION .............................................. 133
Sue Fitzgerald, National Science Foundation

PROCESS-ORIENTED GUIDED INQUIRY LEARNING (POGIL) IN COMPUTER SCIENCE — TUTORIAL PRESENTATION ................. 135
Clif Kussmaul, Muhlenberg College

MOTIVATING STUDENTS TAKING CS1 BY USING IMAGE MANIPULATION IN C AND C++ ................................................ 136
Paul E. Dickson, Hampshire College

THE I-PHONE/I-PAD COURSE: A SMALL COLLEGE PERSPECTIVE ...... 142
Lubomir Ivanov, Iona College

SOUNDLIB: A MUSIC LIBRARY FOR A NOVICE JAVA PROGRAMMER . . 149
Viera K. Proulx, Northeastern University

UNTANGLING THE MAZE OF PEDAGOGICAL STRATEGIES FOR INTRODUCTORY PROGRAMMING — PANEL DISCUSSION ........ 156
Frances Bailie, Iona College, Adel Abunawass, University of West Georgia, Smiljana Petrovic, Iona College, Deborah Whitfield, Slippery Rock University

SHARING YOUR INSTRUCTIONAL MATERIALS VIA ENSEMBLE — TUTORIAL PRESENTATION .............................................. 160
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

SHORT MOBILE GAME DEVELOPMENT PROJECTS FOR CS1/2 — TUTORIAL PRESENTATION .............................................. 163
Stan Kurkovsky, Central Connecticut State University, Delvin Defoe, Rose-Hulman Institute of Technology

MOTION CAPTURE IN A CS CURRICULUM — FACULTY POSTER ...... 165
Bridget Baird and Ozgur Izmirli, Connecticut College
THE CONSORTIUM FOR COMPUTING SCIENCES IN COLLEGES
BOARD OF DIRECTORS

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

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

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

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

**Bill Myers**, Treasurer (2011), Dept. of Computer Studies, Belmont Abbey College, 100 Belmont-Mount Holly Road, Belmont, NC 28012-1802, 704-461-6823 (O), 704-461-5051, (fax), myers@crusader.bac.edu.

**John Meinke**, Publications Chair (2012), Collegiate Associate Professor, UMUC Europe, US Post: CMR 420, Box 3668, APO AE 09063; Werderstr 8, D-68723 Oftersheim, Germany, 011-49-6202-5 77 79 16 (H), meinkej@acm.org.

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

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

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

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

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

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

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

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

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

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

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

**Will Mitchell**, Conference Coordinator, 1455 S 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; Werderstr 8, D-68723 Oftersheim, Germany. 011-49-6202-5 77 82 14, (H) sdean@faculty.ed.umuc.edu.

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

**Paul D. Wiedemeier**, National Partners Chair, The University of Louisiana at Monroe, Computer Science and CIS Department, 700 University Avenue, Administration Building, Room 2-37, Monroe, LA 71209, 318-342-1856 (O), 318-342-1101 (fax), wiedemeier@ulm.edu.

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

**Myles McNally**, Webmaster, Professor of Computer Science, Alma College, 614 W, Superior St., Alma, MI 48801, 989-463-7163 (O), 989-463-7079 (fax), mcnally@alma.edu.
CCSC NATIONAL PARTNERS

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

Microsoft Corporation
Turings Craft
National Science Foundation
Panasonic Solutions Company

FOREWORD

Once again we come to the final conference of the 2010-2011 academic year, and once again Northeastern has outdone itself. When I look at the program I simply salivate, wanting to be there for the presentations – but another item is networking with folks! I see so many names of colleagues that I really would like to be there and chat with – it’s interesting that as I look at the names of the presenters at the Consortium conferences I see a number of names that I was in University with – and have not seen since – but they remain professional colleagues, and enjoy communicating with them as we try to get the conference proceedings in final form.

The Consortium is now over a quarter of a century old, and is still experiencing growing pains! We are all volunteers, and that means that when one of our volunteers is over-extended something has to slip. I apologize personally to all of you for those slips. Our mailings in the last year have been behind schedule, and it’s due to a number of items in the domino effect. We (the Board) are working diligently at trying to resolve such problems. As a Board we try to deliver the best service that we can within constraints, and growing pains adds an additional constraint.

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 a quarter century ago to now having ten regional conferences. George handles the coordination with our printer, Montrose Publishing, and that has been most helpful to me (as I’m located in Germany, a bit of a geographical displacement.) 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. Both George and Susan have been wonderful to work with for over the last couple decades. They continue to be loyal troopers, supporting the Consortium – and I’m most appreciative of their supprt

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 must also express my 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.

Thanks also are go to our printer, Montrose Publishing, and the staff there! As I look at the printing this Spring, and this issue is the final issue, they have supported us through multiple versions of the Journal – as the manuscript went out to the regions minor problems were discovered, and each one resulted in minor changes in the final manuscript, but the patience of the folks at Montrose Publishing must be recognized. My sincere thanks to both Jerry Golis and Tammy Bonnice who have become real friends over the many years that we’ve worked together, and they’ve provided support and pataience to the Consortium.

Attend a regional conference (or two or three) as well as help execute the conference of your choice. Check out the web site – ccsc.org –, the list of regions and the upcoming conferences, as well as how to contact the region and indicate how you would like to assist. 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
Welcome to the 16th Annual Conference of the Northeast region of the Consortium for Computing Sciences in Colleges. The conference is held in cooperation with the ACM SIGCSE and Upsilon Pi Epsilon Honor Society and is hosted by Western New England College in Springfield, MA.

The program committee has worked hard to bring you a thought provoking conference featuring a range of topics, presented in a variety of formats. During the plenary sessions, Stormy Peters will present *Communities: Open Source and Education Working Together*, and Dr. Jane Chu Prey will present *What Can We Do? Raising the Numbers of Women in Computing*. We accepted 52% of the papers submitted. These will be presented in six sessions. We have seven tutorial sessions, a demo session, and two panel sessions.

In response to last year's record attendance, we have added an additional Friday morning, pre-conference workshop, bringing the total to four workshops. We will have a poster session for works in progress. And for our future educators, researchers, and practitioners, we have a student poster contest and a student programming contest.

We thank the program committee for their dogged dedication, without whom none of this would be possible. We thank the CCSCNE Board for their guidance and support. We thank our talented reviewers, who provided prompt and thorough reviews of submissions. We thank Western New England College for hosting the conference. Most importantly, we thank those who participate and attend the conference, making CCSCNE such a dynamic and provocative community.

We hope that you enjoy your visit to Springfield, MA and Western New England College, and that you leave feeling inspired and fulfilled.

*Stoney Jackson*, Conference Co-Chair  
Western New England College

*Karl R. Wurst*, Conference Co-Chair  
Worcester State University
BOARD MEMBERS — CCSC NORTHEASTERN REGION

Lawrence D'Antonio, Chair .......................... Ramapo College of New Jersey, NJ
Richard Wyatt, Secretary, 2010-2013 .................. West Chester University, PA
Adrian Ionescu, Treasurer 2008-2011 .................. Wagner College, NY
Michael Gousie, Editor, 2010-2013 ..................... Wheaton College, MA
Paul Tymann, Membership Chair, 2010-2012 ................
...................................................................... Rochester Institute of Technology, Rochester, NY
Paul Tymann, Webmaster, 2008-2011 .....................
...................................................................... Rochester Institute of Technology, Rochester, NY
Frank Ford, Registrar, 2008-2011 ..................... Providence College, RI
Lonnie Fairchild .................................. SUNY Plattsburgh, NY
Timothy Fossum ..................................... SUNY Potsdam, NY
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

2011 CCSC NORTHEASTERN REGIONAL CONFERENCE

COMMITTEE

Stoney Jackson, Conference Co-Chair .............. Western New England College, MA
Karl Wurst, Conference Co-Chair ..................... Worcester State University, MA
Mark Bailey, Papers Co-Chair .......................... Hamilton College, NY
Aaron Cass, Papers Co-Chair ........................... Union College, NY
Darren Lim, Papers Co-Chair ........................... Siena College, NY
Lisa Hansen, Panels Co-Chair ......................... Western New England College, MA
Susan Imberman, Panels Co-Chair ..................... College of Staten Island, NY
Mark Hoffman, Demos Co-Chair ........................ Quinnipiac University, CT
Yana Kortsarts, Demos Co-Chair ......................... Widener University, PA
Ingrid Russell, Speaker Co-Chair ........................ University of Hartford, CT
Adrian Ionescu, Speaker Co-Chair ........................ Wagner College, NY
Sandeep Mitra, Posters Co-Chair ............................ The College at Brockport State University of New York, NY
Stanislav Kurkovsky, Posters Co-Chair ........................ Central Connecticut State University, CT
Lonnie Fairchild, Posters Co-Chair ........................ SUNY Plattsburgh, NY
John Willemain, Tutorials and Workshops Co-Chair ................................. Western New England College, MA
Bonnie MacKellar, Tutorials and Workshops Co-Chair ........................ St. John's University, NY
Jonathan Hill, Tutorials and Workshops Co-Chair ........................ University of Hartford, CT
Heidi Ellis, Registration Co-Chair ............................. Western New England College, MA
Lori Postner, Registration Co-Chair ........................... Nassau Community College, NY
Darci Burdge, Registration Co-Chair ........................... Nassau Community College, NY
Mihaela Sabin, Vendors Co-Chair .............................. University of New Hampshire at Manchester, NH
Adam Fischbach, Vendors Co-Chair ............................ Widener University, PA
Matthew Hertz, Publicity Co-Chair .............................. Canisius College, NY
Nadine Hanebutte, Publicity Co-Chair ........................ St. John Fisher College, NY
Frank Ford, Programming Contest Co-Chair ........................ Providence College, RI
Leh-Sheng Tang, Programming Contest Co-Chair ........................ Western New England College, MA
Delbert Hart, Programming Contest Co-Chair ........................ SUNY Plattsburgh, NY
Marc Corliss, Programming Contest Co-Chair ............................. Hobart and William Smith Colleges, NY

REVIEWERS – 2011 CCSC NORTHEASTERN CONFERENCE

Ziya Arnavut ........................................ SUNY Fredonia
Barbara Bracken ........................................... Wilkes University
Warren Carithers ........................................ Rochest er Institute of Technology
Elizabeth Chang ........................................ Hood College
Yan Chen ........................................ Borough of Manhattan Community College/CUNY
Mary Courtney ........................................ Pace University
Lawrence D'Antonio ....................................... Ramapo College of New Jersey
Bob Dugan ........................................ Department of Computer Science, Stonehill College
Michael Eckmann ........................................ Skidmore College
Emanuel Emanouilidis ..................................... Kean University
Lonnie Fairchild ........................................ SUNY Plattsburgh
Chris Fernandes ........................................ Union College
Adam Fischbach ........................................ Widener University
Jill Gerhardt ........................................ Stockton College
Michael Gousie .......................................... Wheaton College
WHAT CAN WE DO? RAISING THE NUMBERS OF WOMEN IN COMPUTING

FRIDAY PLENARY SESSION

Jane Chu Prey, PhD
Microsoft Research
co-sponsored by ACM/SIGCSE

Did you ever wonder why there are only 18% women studying computing in the US? Where else are there women in the classroom and do they continue equal representation in the workforce? From the work in the United States; the current studies begun in Brazil; and brief overview of other countries, why are certain places successful in certain levels but not at others? What are the lessons we can learn from our neighbors?

This interactive session will provide a brief overview of the gender diversity situation including directions and potential best practices with ideas for strategies that can be used by conference attendees.

It is hoped that the audience will provide information about their particular communities and ask provocative questions to the audience at large.

SHORT BIOGRAPHY

Dr. Prey received her BS degree from the University of Illinois Urbana in biology and her PhD from the University of Virginia in Instructional Technology. She was a faculty member in the Computer Science Department at the University of Virginia for 11 years before joining Microsoft Research in January, 2004 where she led the Tablet Technologies in Higher Education initiative. Jane is currently responsible for the development and implementation of the Gender Diversity and Pipeline Strategy for Microsoft Research. Dr. Prey also spent two years as the Computer Science Program Manager at the National Science Foundation in the Division of Undergraduate Education. She is a member of the IEEE CS Board of Governors, the ACM Education Board, and the CRA Board as well as a number of university advisory boards.

Jane and her husband of 35+ years have 3 children [all in computing careers]. She loves to travel and read cookbooks and murder mysteries.

* Copyright is held by the author/owner.
COMMUNITIES: OPEN SOURCE AND EDUCATION WORKING TOGETHER

SATURDAY PLENARY SESSION

Stormy Peters
Head of Developer Engagement at Mozilla

Open source software is a relatively new development model that employs critical computer science skills like working in virtual teams, code reading and communication skills. Open source software projects get many of their new developers as students. While programs like the Google Summer of Code internships help funnel students to open source software projects, they also work with universities in a number of ways. Many professors are teaching their students open source software skills by teaching them to work on real projects. This requires collaboration between universities, professors and open source software projects. Come learn and discuss how open source software communities and educational communities are working together.

SHORT BIOGRAPHY

Stormy Peters is the Head of Developer Engagement at Mozilla where she is passionate about open web technologies and the developers that use them. Previously she worked as Executive Director of the GNOME Foundation. Stormy has experience with open source communities and business. She set up OpenLogic Expert Community linking the community to businesses that use their technology. She also worked at Hewlett-Packard where she founded and managed the Open Source Program Office that is responsible for HP's open source strategy, policy and business practices. She received a Bachelors degree in Computer Science from Rice University. Stormy Peters is a passionate and frequent keynote speaker on free software related topics.

* Copyright is held by the author/owner.
GREENFOOT: INTRODUCING JAVA WITH GAMES AND SIMULATIONS

PRE-CONFERENCE WORKSHOP / TUTORIAL PRESENTATION

Adrienne Decker  
Department of Computer Science & Engineering  
University at Buffalo, SUNY  
201 Bell Hall  
Buffalo, NY 14260  
(716) 645-3184  
adrienne@buffalo.edu

Frances P. Trees  
Mathematics and Computer Science Department  
Drew University  
36 Madison Avenue  
Madison, NJ 07940  
(973)408-3552  
ftrees@drew.edu

ABSTRACT

Greenfoot is an interactive Java development environment created by the developers of BlueJ for use in introductory programming classrooms. Greenfoot provides a graphical animated environment in which to develop programs while students only need to use basic Java syntax to control the actors in their scenarios. Neither students nor educators need to focus on the graphical elements of the programs. They simply focus on the behaviors of the actors and the system provides the graphics. This hands-on workshop is intended to give educators (middle school, high school, and college) an introduction to the Greenfoot environment, a demonstration of how it can be used to introduce object-oriented programming to students, and a guided approach to developing Greenfoot experiences that can be integrated into existing curricula. The resources that are available to instructors using the environment will also be discussed.

INTENDED AUDIENCE

This workshop is intended for those instructors (middle school, high school and college) who have never worked with the Greenfoot environment before. The workshop will introduce the environment and how it can be used in the teaching of introductory programming. Knowledge of Java is preferred, but knowledge of another programming

* Copyright is held by the author/owner.
language is sufficient. The workshop is not a workshop about Java, so Java-specific syntax issues will not be explained, but Java is the language used in Greenfoot and in the hands-on exercises introduced in this workshop.

PARTICIPANT REQUIREMENTS

If the workshop is not held in a computer laboratory, participants will need to have a laptop in order to fully participate in the workshop. Prior to the workshop, it participants should download and install the Java SDK and Greenfoot (both are free). Instructions and information for all workshop participants are available at http://www.greenfoot.org/hubs/newyork/events/20110415-workshop.html.

MATERIALS PROVIDED

Each participant will receive hard copies of

• the presentation slides

WORKSHOP AGENDA

[10 minutes] Introduction
[20 minutes] Greenfoot Demonstration
[60 minutes] First Scenario (Hands-on exercise)
  • Movement
  • Keyboard control
  • Random behaviors
[60 minutes] Moving Beyond the first scenario: Advanced Features (Hands-on)
  • Image control/advanced animations
  • Sound
  • Collision Detection
[15 minutes] Publishing Scenarios and Overview of Resources Available
[15 minutes] Questions, Discussion, and Wrap-Up

WORKSHOP LEADERS

Adrienne Decker is a teaching assistant professor in the department of Computer Science & Engineering at the University at Buffalo, SUNY and the leader of the New York Greenfoot Hub. She has been teaching introductory object-oriented programming using Java for ten years to both majors and non-majors and has been teaching using Greenfoot for two years. Adrienne is a past organizer of five Killer Examples Workshops at OOPSLA 2005-2009. She currently serves as secretary for the Western New York CSTA and has given several presentations to the local chapter about various computing concepts. The University at Buffalo, under Adrienne's initiative was named one of the first Greenfoot Hubs in the United States. Greenfoot Hubs are local representatives who have been chosen to educate others about using Greenfoot in the classroom.
Fran Trees teaches Mathematics and Computer Science at Drew University in Madison, NJ. She primarily teaches introductory programming courses and has integrated Greenfoot into these courses. Fran is actively involved with the AP CS program and has conducted AP CS teacher workshops and institutes for many years. As CSTA Chapter Liaison, Fran promotes CS education and professional development to various local CS communities. Drew University, under Fran's initiative, has also been designated as a Greenfoot Hub.
AN OPEN SOURCE FIELD TRIP FOR FACULTY*

PRE-CONFERENCE WORKSHOP

Gregory W. Hislop  
Drexel University  
hislop@drexel.edu

Mel Chua  
Red Hat, Inc.  
mel@redhat.com

Clif Kussmaul  
Muhlenberg College  
kussmaul@muhlenberg.edu

Sebastian Dziallas  
Olin College  
Dziallas@students.olin.edu

ABSTRACT

Student participation in Free and Open Source Software (FOSS) projects and communities offers excellent opportunities for learning and for developing student interest in computing. This workshop will be a virtual field trip for faculty wanting to know more about FOSS communities, practices, and tools so they can take advantage of this opportunity. The workshop will provide an overview of FOSS from a development perspective, hands-on activities to introduce FOSS tools, and examples of student participation in FOSS projects. Participants will also be introduced to a variety of resources that support faculty and student participation in FOSS communities. The intended audience is computing educators at the college or high school level.

AGENDA

This workshop builds on prior workshops related to student participation in FOSS. These include an NSF-funded workshop on involving students in the development of Humanitarian FOSS, and the Red Hat Professor's Open Source Summer Experience (POSSE). To the extent possible given the venue, the workshop will involve hands on work. The workshop will include discussion among participants. Topics will include:

1. The Landscape of Open Source Software
   • The FOSS movement/concept, history, principles and economics of FOSS
   • Educational potential of student participation in FOSS communities
2. Virtual Field Trip: FOSS forges and project sites
   • Common features
   • Evaluating a FOSS project for possible student participation

* Copyright is held by the author/owner.
3. Virtual Field Trip: FOSS tools and techniques
   • Collaboration and knowledge sharing tools
   • Project control tools
4. Examples of FOSS in Education
   • Approaches to incorporating FOSS participation
   • Examples for students with strong computing background
   • Examples for non-majors and students with less computing background
   • FOSS for social good.
5. Roadblocks and Resources
   • Potential roadblocks and resources related to student FOSS participation
   • Getting started
6. Wrap-up
   • Discussion about potential for FOSS in computing education.
   • Discussion about collaborating on projects.

PRESENTER BIOGRAPHIES:

Mel Chua is part of the Red Hat Community Architecture team. She has worked on a variety of open source projects including Fedora, Sugar Labs, and One Laptop Per Child. She is currently organizing the Red Hat Professor's Open Source Summer Experience (POSSE) to help faculty gain FOSS skills and knowledge.

Sebastian Dziallas is the engineering manager for Sugar on a Stick (SoaS), a Fedora-based Linux distribution for the Sugar Learning Platform, originally deployed by the One Laptop Per Child (OLPC) project. Sebastian is also a Fedora packager and the founder of Fedora's Education SIG. In his free time, he is a student at Olin College of Engineering in Needham, MA.

Gregory Hislop holds faculty appointments in Information Science and Technology and Computer Science at Drexel University. He PI on the HumIT project and co-PI on SoftHum project (see xcitegroup.org). Dr. Hislop has 15 years experience leading development of curricula in SE, IS, and IT, and spent almost 20 years as an IT professional.

Clif Kussmaul is Associate Professor of Computer Science at Muhlenberg College and formerly Chief Technology Officer, Elegance Technologies, Inc. and has 15 years of experience in software development, FOSS, and computing education He has contributed to a variety of FOSS projects including Drupal, Moodle, Twiki and Trac. He has also guided student participation in FOSS.
GETTING READY FOR A UNIT IN ALICE 2.2: WEEK 1 / EVENT-PROCESSING IN ALICE 2.2

PRE-CONFERENCE WORKSHOP / TUTORIAL PRESENTATION

Eileen M. Peluso, PhD
Mathematical Sciences Department
Lycoming College
700 College Place
Williamsport, PA 17701
Telephone: (570) 321-4135
Fax: (570) 321-4371
Email: pelusoem@lycoming.edu

ABSTRACT: Carnegie Mellon's Alice is a freely downloadable teaching tool that uses 3D graphics to create a fun and engaging first programming experience. Incorporating a unit using Alice in an applications, graphical art, or web design course will introduce students to fundamental programming concepts and hopefully interest a diverse group of students in taking additional coursework in computing. Participants in this hands-on workshop/tutorial sequence who have no Alice experience will learn the Alice 2.2 environment, and all participants will leave with materials for a 2-week instructional unit in Alice 2.2. As participants work through the unit materials, pedagogical considerations will be presented and discussed.

The primary audience for the workshop and tutorial sequence is middle school and high school technology teachers and college professors interested in using Alice in a course for non-majors. With that said, anyone simply wanting to learn Alice will enjoy the workshop and tutorial. No previous experience with Alice is needed; however, participants with knowledge of Alice will find value in the unit materials and other pedagogical aspects of the workshop. The workshop stands on its own; tutorial participants should either attend the workshop or have some familiarity with Alice.

Each participant will receive a hard copy and the link to a soft copy of lesson plans, handouts, and rubrics for four structured activities and a lesson plan and rubric for a fifth, open-ended activity, all using Alice. Additionally, one general information handout and two tips-and-tricks handouts will be included. The soft copy will also include completed activities and additional sample Alice worlds/programs.

* Copyright is held by the author/owner.
INTRODUCING CASE STUDY BASED TEACHING IN

COMPUTING CURRICULA*

PRE-CONFERENCE WORKSHOP

Massood Towhidnejad, Salamah Salamah, Thomas B. Hilburn
Embry-Riddle Aeronautical University
600 S. Clyde Morris Blvd.
Daytona Beach, Fl, 32114
towhid@erau.edu, salamahs@erau.edu, hilburn@erau.edu

ABSTRACT

The use of case studies is an effective method for introducing real-world professional practices into the classroom. Case studies have become a proven and pervasive method of teaching about professional practice in such fields as business, law, and medicine. The term “case study” is used in a variety of ways. In its most naive form, it simply refers to a realistic example used to illustrate a concept or technique. Although the use of case studies in education has shown success in the above mentioned disciplines, it is yet to be adopted in any significant way in the computer science education. One of the reasons for the lack of use of the case-study approach is the lack of sufficient material for this purpose. The main aim of the workshop is fourfold. First, the participants will be introduced to a comprehensive set of case study (the DigitalHome case study) that can be used throughout a computing curriculum, emphasizing full software development life cycle. Second, the workshop will include an interactive discussion of how to use the case study artifacts and the associated teaching support materials to complement their teaching artifacts throughout their computing curriculum. Third, the attendees will be asked to participate in the assessment of case study material and make recommendations for improving both the material and the delivery. Fourth, we intend to follow up with the attendees who are interested to participate in our project, by incorporating some of the case study material as part of their course work during the fall 2011 semester.

* Copyright is held by the author/owner.
INTRODUCTION

Many computing programs have a software engineering course that involves a software development project in which students are grouped into teams to work on a semester or year-long software development project. Unfortunately, this is too often isolated from the rest of the curriculum and does not form a real-world basis for the entire curriculum. As a result, these programs produce graduates who are familiar with the basic theoretical concepts in software development, but lack the skills to apply these concepts in real-world development environments. Therefore, it is imperative that computing curricula introduce professional and real-world education into the academic programs. The use of case studies is one widely-used method for introducing real-world professional practices into the classroom.

Although many computing text books include the use of case studies to explain ideas, these cases seem to serve a specific purpose (e.g., discussing programming constructs, software planning or requirements analysis, design, risk analysis, construction or testing issues by using simple examples that are quasi-realistic), they often lack the following:

- Realistic artifacts (often space does not allow providing a complete requirements or design document)
- Completeness (covers only a portion of the life cycle, and not an end to end), with a focus on design and implementation
- Ability to decouple from the text and apply in ways not intended by the author
- Techniques for integration into course activities or into the curriculum as a whole
- A scenario format that would motivate students to get engaged in problem identification and solution.
- Guidance to the instructor on how to use the case study to teach a course topic or concept

The DigitalHome case study [1, 2] is intended to address these shortcomings by providing a complete set of artifacts associated with software development as well as providing case modules that can be used by faculty in teaching different subjects in a computer science curriculum. Each case module represents a mini case study and is associated with a specific teaching subject and learning objectives (e.g. requirements analysis, object oriented design, testing, team building...). Case modules also include an exercise booklet and a set of guidelines to assist the instructor in teaching the session.

WORKSHOP GOALS

The aim of this workshop is to advance the understanding of case study based teaching and to introduce the participants to a comprehensive case study in software development. The goals of the session are:

1. Describe the background and effectiveness of case study techniques in professional education.
2. Introduce the DigitalHome software development case study material.
3. Initiate the use of case study material (artifacts and case modules) by faculty in a variety of computing programs and courses.
4. Assess the quality and utility of the case study material by the participating faculty.
5. Assist interested participants in developing plans to incorporate case study material as part of their course offering during fall 2011 and beyond.

WORKSHOP DETAILED AGENDA

The workshop’s detailed agenda is as follows:

1. **(20 minutes)**: Facilitators and participants introductions and backgrounds
2. **(15 minutes)**: Introduction and background to case study teaching, previous uses, and advantages.
3. **(30 minutes)**: Description of the DigitalHome case study, its origins, artifacts, and case modules.
4. **(50 minutes)**: Group exercise on the use of the DigitalHome case modules. Participants will be broken into groups of 4-5, based on their major area of teaching (i.e., programming, data structure algorithm, software engineering, etc). Each group will be provided an appropriate case module to discuss and evaluate. The participants will be asked to discuss the utility of the case module as well as suggest necessary changes to improve both the particular case module as well as the DigitalHome case study as a whole.
5. **(40 minutes)**: Group discussion; groups report back on their activity in # 4 above.
6. **(25 minutes)**: Workshop summary, final thoughts and discussions.
7. **(60 minutes, after the completion of workshop)**: Development of initial plan for fall 2011 for interested parties.

ACKNOWLEDGMENT

Initial work on the DigitalHome case study was funded as part of the NSF project: “The Network Community for Software Engineering Education” (SWENET) (NSF 0080502). In addition, the current work on the case study is funded through NSF’s (DUE-0941768) “Curriculum-wide Software Development Case Study”.

REFERENCES


ABSTRACT
In many universities, student success and retention in computer science are an ongoing concern to faculty. Finding best practices to improve the success and retention rate of less than the best students, has been a particular challenge. This study, conducted at a public teaching university, has focused on enhancements to introductory computer science courses to aid in the retention of students in STEM (Science, Technology, Engineering, and Mathematics) majors. The majority of these students had no prior programming or computer science experience. All computer science majors and some other STEM majors are required to take the first course in programming (CS0). The PLTL (Peer Led Team Learning) methodology, successful with Chemistry majors elsewhere, was used to improve student success in CS via collaborative workshops. Quantitative grade data and attitudinal surveys are combined to show the success of this study on the student success rates, which contributed to the retention of these students over the past 3 years. Over 80% of the students who attended PLTL workshops, said that future students should attend PLTL workshops in this and follow on CS courses. The progress of the students' success, and the enthusiasm of the students in introductory CS classes shows the contribution PLTL has made in the retention of STEM students.
INTRODUCTION: Prior Work in Peer Led Team Learning

The development of Peer Led Team Learning at CCNY was for improvement of student learning and reduction of attrition in Chemistry. [2] CCNY is a public teaching university with comparable student profile and demographics to Kean University. PLTL at CCNY involved three overarching principles:

- Peer leaders were facilitators of student learning, not instructors, or answer givers, to reduce dependency on instructors, and increase self and group scientific discovery.
- Workshops were to consist of well formed problem solving exercises, highly integrated to the course material, for small group collaborative problem solving.
- Trained by faculty, Peer Leaders and faculty developed the PLTL exercises to aid in peer leader growth, and assurance that exercises were tuned to the course material.

There is significant literature that demonstrates the benefits of adding PLTL workshops to a scientific course like Chemistry. [3] CCNY contributed materials to help us learn the PLTL workshop concepts. Computer Science is relatively new to using this PLTL model. Two successful studies have used PLTL in Computer Science, and are documented. One was a consortium of competitive colleges and universities, which used the Emerging Scholars model and recruited strong minority students.[4] Rutgers University was a member of that group. We started with a set of PLTL workshops from Rutgers, for CS1 classes, and we used a few at the beginning of our process. The other study at Georgia Tech [1] was with students whose mean SAT score was 1340 to 1356 in their 3 year study. Kean is a public teaching university. We wanted to see if PLTL could help ordinary students (with Math SAT of 500-599), and under achiever students (at risk of earning C or D, but had potential to earn B or C). We wanted to improve their understanding of algorithms, and problem solving, and their success in introductory Computer Science. We also wanted to see if PLTL could improve student interest, so they would want to stay in their STEM majors. It is one thing to assist scholar Computer Science majors who are generally doing ok in the course to do better, but it is another level of challenge to involve the under achiever students at risk of earning a "C" or "D", to help them learn enough to want to stay in their STEM major.

PLTL AT KEAN UNIVERSITY

According to Kean University's Mission statement: "Kean offers a wide range of demanding programs that provide the high quality of instruction and academic support services necessary to assure its diverse students the means to reach their full potential, including students from academically disadvantaged backgrounds, students with special needs, and adults returning or entering higher education." So the challenge for us in the sciences is how do we create a program of high quality with students, many of whom are from academically disadvantaged backgrounds? We provide various means of academic support. In the past this has been primarily tutoring. But in the sciences, we tend to find that while students do get help to solve the particular homework they are trying to accomplish, tutoring does not generally produce self reliance. And the tutored students often withdraw, or get "D"s. Also many students who are getting by with a "C" do not go
to tutoring. So in the Fall 2006, we started the PLTL program at Kean, funded by the Epsilon Corps grant from NSF(DUE #431637). It was to provide an alternative to tutoring for math and computer science introductory courses. It would be for enrichment for all students, as a means to encourage students to continue as STEM majors. The math PLTL workshops did not continue, but the computer science PLTL workshops did.

In 2006, and 2007 overall DFW rates (% of students with D, F, or W grades) in introductory CS classes were higher than we were comfortable with. PLTL might be a way to aid in improving student outcomes, lowering DFW rates, while raising the success rates of the entire class. So those were our goals, to lower DFW rates, to improve the grades of all the class in the process, and to improve retention in the STEM majors. Because PLTL focuses on all of the students, not just the weakest or the strongest students, we felt it was a good candidate tool.

Kean is mainly a commuter university where about 2/3 of the students work for pay. About 1/3 of our CS0 students work 15 to 25 hours a week and 1/3 over 25 hours a week. Convincing them to participate in enrichment is difficult. We used surveys to determine what they thought of PLTL. The main reason why they did not attend was they had conflicting times for work or classes, and they had many other commitments. [7]

Participation Was Key

It took 6 semesters of trial and error at Kean until we found a way to improve the voluntary participation by the students. In Fall 2007, the workshops were offered in CS0 (Computing Fundamentals - with Java programming) classes, but almost no one came. The students thought these workshops were tutoring for D and F students, so the students could pass exams, or turn in solutions to the assignments. In January 2008, the PLTL leaders were given 20 minutes in each section, to lead a demonstration workshop problem with the class, but almost no one came to the regular PLTL workshops. The faculty and PLTL leaders spent the early semesters creating and improving workshop problems. In Fall 2008 the department agreed that all faculty of CS0 would give a small amount of extra credit to students who participated in PLTL workshops. Even with extra credit, very few students came. But just before the midterm, 4 students, concerned that they might not do well on the midterm, asked if they could have PLTL sessions on Saturday. We found a Saturday PLTL leader. This was the first time a group of 6 students (4 regulars) began to act like a PLTL group was supposed to. They came on a regular basis, and although 4 of the 6 were in jeopardy of a "D", they got a "C", and the 2 better students got a "B". But those 6 were not common. In Spring 2009 only 2 students out of 80 came to PLTL workshops regularly. So in Fall 2009, in 1 section, the first half hour of the PLTL workshop was scheduled inside the regular class time. That worked! One section had everyone exposed to a PLTL workshop in class time, and the students could elect to stay or go at the end of class time. The other 2 PLTL workshop problems were immediately following the class, for those who chose to stay. Each student in that section got a chance to experience the value, the fun, and the collaborative learning of the PLTL workshop problems, in a peer led environment. About half of that section stayed. And of course at exam weeks, the workshop swelled. So students who have to work 20 to 30 hours a week, took an hour out of their week to stay for PLTL workshop, because they experienced the value in it. The Spring 2010 semester we did it for 2 sections. It was the first time we had
50% participation of 2 sections. Word got around, and about 6 students from another professors' class came regularly. PLTL was finally on the way to real participation.

Peer leaders took attendance at the workshops, and were charged with keeping everyone participating. So we had all the data of who went to workshops, and we had their grades. This enabled us to do some comparative grades between the half of the class who chose to stay, and the half that did not. Of course, the half who did not stay for PLTL did get 1 short workshop problem. And the students who did stay for PLTL were self selected, and could be called the more motivated half of the class. Because most of the students from the other professors' classes were so few, the grades data here for Fall 2009 and Spring 2010 is for my 2 sections of 21 each, totaling 42 students a semester.

**PLTL Workshops Improved Student Ranking**

One challenge of the study was to prove that our students could improve during the semester with PLTL workshops. Since the first half of the course is easier, students who only earn a "C" or a "D" on the midterm, are at risk of failing the final exam. So we looked at how the PLTL workshops in the second half of the course could assist all students, and especially the at risk students, to do their best in the more difficult half of the course. Figure 1 shows that 60% of PLTL students clearly improved their rank from midterm to final, whereas 50% the non-PLTL half declined in their class ranking by the time of the final exam.

The students often come from a background where teachers gave predigested, top down concepts with pre-solved problems, and students were expected to be able to follow the solution and mimic the solution on a similar problem in the lab. Students often said that the word problems on the midterm were too difficult to solve, even if they knew the basic skills of programming. Getting these students to think for themselves is a real challenge. The PLTL leaders and the faculty create leading questions for the workshop exercises that enable the students to discover the solutions. For example "What do you know about the problem?" or "What are the main steps this solution needs to do?"

**PLTL Improved the Success For All Students in Intro Computer Science (Fig 2)**

Algorithmic thinking is a very important part of this course, because CS0 is required for many STEM majors in their curriculum, as well as Computer Science majors. Therefore our emphasis is on algorithmic thinking in PLTL enrichment and problem solving skills for all students. Because of the collaborative methods used in PLTL
workshops, and the fact that there is only one PLTL leader for 6 to 20 students, the better students were encouraged to assist their peers also, in the problem solving efforts. Because PLTL workshop problems are not graded, these efforts were strictly focused on enrichment for all participants. Some students thought it was just "more work" but that is why the PLTL leaders, with the professor, worked every week to try to make the problems both relevant to the material of the week of the course, and also some degree of fun. We used pairs with think/pair/share, or groups of 3 or 4 to work on an algorithm of various games, and frequently had two small groups share their solutions with the others on the board. Then students critique what was different about the two algorithms.

In Fall 2009 the best students were not staying for the PLTL sessions. But by the final exam, the PLTL students caught up to the non-PLTL students and had the same average on a difficult final exam. One way to deal with students who do not do well on exams is to make the final easier, but we were determined to improve the student understanding instead. So in Spring 2010 midterm and final exam grades, the two halves of the class were almost the same at the midterm, but by the time of the final exam, the PLTL attendees had 10 more points average on the final exam.

![Figure 3 DFWRates Decrease and CRate Increases with PLTL](image)

**PLTL Workshops Reduced the DFW Rates**

The best illustration of the progress of our PLTL program through the semesters is the line graph above (figure 3). The lines show that through the semesters, the progress of students without PLTL was non-existent, but the progress of the students with the PLTL was significant. Those who rarely or never attended PLTL had aggregate DFW rates of over 30% each semester, but the F and W rates went to 0% for those attending PLTL, and to 5% to 12% for the D grades. And the C grades went from around 30% each semester without PLTL, to about 45% to 58% for the frequently attending PLTL students. The A's and B's improved for the PLTL students, but were not as significant.

**Student Assessment of PLTL Workshops**

The anonymous student attitude surveys clearly showed a positive assessment about PLTL workshops. Improving the interest of under achievers has a clear link to improved success. In figure 4 over 80% of the PLTL attending students felt that future students in
the same introductory computer science course should attend PLTL, as well as follow on computer science courses should provide PLTL workshops. In our commuter college, 55% felt that PLTL workshops improved a sense of community. And in figure 5, over 70% of the PLTL attending students said PLTL helped them do better on exams, and over 65% said it helped them in the course overall. The long term attitude was that about 65% of the students attending PLTL workshops felt it helped them remain a STEM major, which is one of the original goals of the PLTL introduction. (The only students who strongly disagreed, went to PLTL but were not STEM majors.) Each year that the students attended in stronger numbers, the surveys also improved. The students clearly indicated that the conflict of student commitments of work and other classes, does remain a difficult problem to solve.

PLTL Improved Leadership Of the Leaders

With weekly leader workshops, the PLTL leaders learned how to ask questions, instead of answering them, and become small group facilitators, rather than to give solutions to students. When the students had difficulty creating algorithms, the PLTL leaders learned to guide the students by having them re-read the problem, and discuss the start of a solution among the small group. It is imperative that PLTL leaders help the professor to create/improve the workshop problems. As PLTL leaders work each week to improve the exercises with leading questions, they help direct students to discover the solutions to problems in the process.

![Figure 4: Student's Recommendations for Future Computer Science students by PLTL attendees Spring 2018](image)

![Figure 5: Student's Opinions of Benefits of PLTL by PLTL attendees Spring 2018](image)

The leaders learn to see the problems as a discovery tool, and the leader role to enable the discovery by the students. One of the leaders said in the leader survey that "I have begun to learn what leadership and facilitator really mean. I used to think it meant showing the student the solution."

CONCLUSIONS

Our experience with PLTL in early Computer Science classes was very encouraging. When PLTL is used with typical students, who have many commitments, and less than
excellent backgrounds, the success rates of all students can improve, with lower DFW rates, and improved retention attitudes of STEM majors. There are some requirements.

- PLTL workshops need to be clearly integrated into the course.
- Computer Science focuses on algorithmic thinking & collaborative problem solving.
- PLTL requires the participation of the students! The workshops must be of clear benefit to the students.
- PLTL leaders need to be trained as facilitators, not tutors, and PLTL can grow new leaders from among the ranks of the PLTL students.
- PLTL workshops can be grown, and improved, and shared, so that everyone does not have to invent them from scratch.
- PLTL needs support, because the student leaders need to be paid, and at least one faculty member needs to be a coordinator of the students and workshops.

Creating the workshops is a major task. And tailoring the workshops to encourage algorithmic thinking has been a serious effort of our leader workshops. So, we began with some workshops from Rutgers, and others are available from University Wisconsin, Milwaukee and Indiana University.[5] Our PLTL workshops are available for sharing with other universities who wish to use them. (www.kean.edu/~cstewart/cps1231.html)

REFERENCES


ABSTRACT

One of the main challenges students in introductory programming courses face is learning the syntax of a particular programming language, such as C++ or Java, while they simultaneously try to understand the fundamental logic of programming constructs. Visual Logic© (www.visuallogic.org) is an interactive, graphical, flowchart tool that can be used to address this challenge. In this paper, we present three different courses, developed independently at three different colleges, using Visual Logic with completely different approaches. The first course is a General Education CS course introducing algorithmic thinking. The second is a CS0 course which uses Visual Logic for six weeks and then transitions to Python. The third course is a CS1 course which uses Visual Logic as a tool to teach programming concepts and to brainstorm solutions. Visual Logic is not taught separately, but integrated with the various Java control structures. Once students gain an understanding of basic programming logic concepts, the transition to a programming language such as Python or Java, in the same course or the next course, is often much easier for them. Though these are different approaches in three different courses, using Visual Logic has shown an increase in the interest level of the students and enhanced their learning.
FEATURES AND ADVANTAGES OF VISUAL LOGIC

Visual Logic is an interactive, graphical software tool that enables a novice programmer to develop interactive, executable flowcharts [9]. Visual Logic is extremely user-friendly, requires minimal syntax, and is very simple for a student to learn. Students are introduced to the logical structures found in most programming languages: input and output (console, dialog box and file), arithmetic and logical operators, assignment statements, if/else statements, loops, arrays, functions and user-defined procedures. Students select a statement from a drop-down menu and enter only constants (numeric and string), variable names, arithmetic expressions or Boolean expressions. Visual Logic includes a graphics package containing LOGO style commands which can be used to create colorful graphics programs.

Although other good programs, such as Alice and Scratch have been developed for introducing programming, Visual Logic includes some unique features. The program introduces students to the fundamentals of flowcharting, a traditional method of graphically illustrating a program. The "visualness" of the flowchart symbols and connecting arrows clearly illustrate the logic flow. Students can easily correct the logic of a program, rerun the flowchart, and determine if it produces the desired output.

RATIONALE FOR USING VISUAL LOGIC

The purposeful, graphical flowchart symbols provide a visual perspective to a conceptually abstract discipline. Since most students are visual learners [2], attempting to teach abstract concepts using a traditional programming language, which is very textual, adds to the complexity of the learning experience of many students. The simplicity of the Visual Logic program and its straightforward correlation with the essential structures of most programming languages provides a good visual introduction to programming concepts [3]. The analysis, algorithm development, and documentation skills acquired through flowcharting can be applied to other languages in future courses.

IN A GENERAL EDUCATION COURSE

"Introduction to Algorithmic Thinking" is a new, General Education, Computer Science course at Rhode Island College. It provides an alternative to the traditional Introduction to Computers computer literacy course, for students of all majors. Computational Thinking is emerging as a basic skill that should be developed in K-12 and General Education [6], and this course in Algorithmic Thinking addresses this need at the introductory Computer Science level.
The course 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. Students interested in programming may continue on to take the CS1 course which introduces Object Oriented Programming in Java.

Students taking this General Education course come from varied backgrounds and lack prior programming experience. This gives rise to two major challenges: (i) finding simple, real-world, interesting problems for in-class examples and student assignments and (ii) finding the appropriate programming language/tool for students to develop programs and test their logic. Using Visual Logic solves (ii) very effectively due to the advantages and interesting features described above.

Some examples of interesting problems from various disciplines (e.g. arts, sports, math, science) used in the course, and the underlying algorithmic concepts in their solutions, were on display in a CCSCNE 2010 poster [7]. It also included an example to teach procedures in a simple program using Visual Logic's LOGO capability.

The first programming assignment is a Mad Libs program, shown below, which introduces the concepts of input, output and variables in a fun and familiar game. Visual Logic makes it easy to come up with quick interactive solutions, thus allowing students to try out many more problems than they would have using a traditional programming language.

The first programming assignment is a Mad Libs program, shown below, which introduces the concepts of input, output and variables in a fun and familiar game. Visual Logic makes it easy to come up with quick interactive solutions, thus allowing students to try out many more problems than they would have using a traditional programming language.

Students from various majors as well as undecided students have the opportunity to successfully learn programming fundamentals in a (almost) syntax-free environment, thus focusing on the real content of the course, "algorithmic thinking" applied to problems from various disciplines. The same reason makes this one of the most enjoyable and satisfying courses to teach.
"Introduction to Programming Logic", the CS0 course at Chestnut Hill College is a prerequisite for Computer & Information Science/Technology majors and other interested students who have had no previous programming experience.

The goal of most introductory programming courses is to develop students' problem solving skills and to express algorithm solutions using a computer programming language [5]. As the complexity of the programming languages has increased, students' struggles with the syntax of the languages have made them lose sight of the main purpose of the course - solving problems through algorithm development. This problem is not unique to students at CHC[8].

To introduce prospective computer science and computer technology majors and minors to the essence of computer programming without overwhelming them with tedious syntax, the department revised their CS0 course so that students could focus on algorithm development using Visual Logic in the first part of the semester and then implement these algorithms in Python during the second half of the semester. The rationale for introducing a programming language as a complement to Visual Logic and for choosing Python in particular reflects an effort to meet the needs of the student population, adhere to curriculum requirements, and prepare students for the workplace. Students are often limited in the number of major courses that can be required of them. In order to provide students with exposure to a variety of programming languages, the department decided to introduce students to a "real" programming language, believing that, as Agarwal et al [1] state "the use of a higher-level programming language gives the students a firmer foundation".

Much consideration was given to choosing the language that would complement Visual Logic. Python was chosen for its simple syntax and its growing use in industry. Many of Python's features, such as not requiring variable declarations, coincide with the way that Visual Logic operates. Students are introduced to Python through a simple manual developed by the instructor that emphasizes the relationship between the concepts learned in Visual Logic and reintroduced in Python. As students learn Python, the basic programming structures introduced in Visual Logic are reinforced as they examine them in a different context. Students revisit programming problems they completed in Visual Logic and recode them in Python, helping them see the connection between flowcharting and programming in Python.

In general, students welcome the ease of Python as they begin learning the language-- though some miss the graphical interface of Visual Logic. The following simple Visual Logic flowchart program and its Python program equivalent demonstrate the coding of a FOR loop using a negative step, random number generation, and screen output.

```python
import random
for counter in range(99, -1, -1):
    print(random.randint(0, 99), end= " ")
```

![Visual Logic Flowchart Program](image)
The implementation of a few programming structures in *Visual Logic* differs from their implementation in *Python*. For example, although *Visual Logic* provides a structure to represent arrays, *Python* does not support arrays, but instead supports a more powerful list structure. The list structure cannot be applied in *Visual Logic*, but the exposure to arrays in *Visual Logic* helps students understand the differences between arrays and lists and gives them an appreciation for the power of lists.

By the end of the course students are generally comfortable with the basic structures of programming and are ready for CS1, which focuses on an early introduction to objects in *Java*. Although the use of *Visual Logic* in the first CS course has not solved all the problems of preparing students for *Java*, many students find the transition less daunting.

**IN CS1 (with JAVA)**

"Introduction to Programming", currently taught in *Java*, is the first required course in the Computer Information Systems and Computer Science majors at The College of Saint Rose. This objects-late course is also the required general education course for mathematics majors, all science majors, math adolescence education majors, elementary education math concentrators, and criminal justice forensics majors.

Students enrolled in "Introduction to Programming" have had very little preparation for learning a syntax-driven programming language and often have difficulty progressing from a problem statement to a sequence of logical steps. *Visual Logic* can be used to demonstrate the how a control structure works before teaching the related *Java* syntax. *Visual Logic* does an excellent job of illustrating difficult concepts such as when to use a decision versus a loop and the transfer of execution that happens when you call one method from another. The student sees separate flowcharts and the interaction between the flowcharts. This visual tool makes the concept come to life!

*Visual Logic* can also used as a tool when brainstorming a solution to a problem as a class and also when students do independent homework. Instead of proceeding directly to writing the program in *Java*, students develop a *Visual Logic* flowchart. They are able to incrementally test this solution by "running" the flowchart. Once students have arrived at the correct sequence of steps to produce the desired output, then they can proceed to writing the program, using appropriate *Java* structures/techniques for each step. The very last step is dealing with syntax errors!

To illustrate, one problem that was solved as a class: "Write a program that calculates the amount a person would earn over a period of time if his or her salary is one penny the first day, two pennies the second day, and continues to double each day. The program should display a table showing the salary for each day, and then show the total pay at the end of the period. The output should be displayed as a dollar and cents amount, not the number of pennies. *Input validation: Do not accept a number less than 1 for the number of days worked.*"[3]

This *Visual Logic* flowchart was developed incrementally. The final flowchart shows a do-while input validation loop and a for loop to generate tabular output.

Students in this course struggle a little with *Java* syntax as all students do, but it is at a much later point in the process, only after they have developed the correct logic.
After using *Visual Logic*, they have confidence in the steps and sequence of their solution. Using *Visual Logic* encourages them to categorize *Java* control structures and techniques (Scanner input, JOptionPane output, etc) as building blocks that can be used to create the overall solution of any problem.

**CONCLUSION**

This paper describes three different CS courses using *Visual Logic* with different approaches. In all courses, novice programmers were able to learn basic programming concepts in a visual, (almost) syntax free manner, thus focusing on the logic of the problem's solution. This minimized student frustration and enhanced learning. In student course evaluations, most of the comments were extremely positive. A typical example: "Visual Logic was really helpful when it came to learning the material in class and working on it by myself" speaks of the ease and success of this versatile tool. From the instructor's perspective, using this visual tool generates far more student enthusiasm and understanding than the traditional approach of explaining syntax and fixing errors.

**REFERENCES**


ALIGNING GENERATIONS TO IMPROVE RETENTION IN
INTRODUCTORY COMPUTING COURSES*

Christian Roberson
Computer Science and Technology Department
Plymouth State University
Plymouth, NH 03264
caroberson@plymouth.edu

ABSTRACT
Incoming students at higher education institutions exhibit different core traits than those of students from past generations. The misalignment that can occur between the traditional course approach and the expectations of students is one of the contributing factors to low student retention, especially in Computer Science and Information Technology programs. The author's school observed a clear misalignment of this nature in a first semester introduction to computing course. After researching the core traits of Generation Y and Generation Z, the department underwent a process of redesigning their course to better meet the needs of today's students. This paper provides an overview of the core traits of the current generation of incoming higher education students, describes in detail the major changes involved in the restructuring of the introduction to computing course, and examines the positive results of their implementation during the first offering of the revised course.

INTRODUCTION
Retention of first-year students has been a problem for many higher-education institutions, especially for computer science and information technology programs. At Plymouth State University we have observed a similar trend in the Computer Science and Technology Department. Many students come to the university and sign up to be a Computer Science major or an Information Technology major, but end up changing majors after only one or two semesters. Students tend to refer to our freshmen courses
as "weed-out" courses, implying they are more of a barrier to entry into the program than courses designed to stimulate interest in the field.

One issue that we have observed at Plymouth State is that there is a clear misalignment between the expectations of faculty teaching the introductory courses and the students taking the courses. This issue is compounded by the generation gap between these two groups. Most faculty members are either from the Baby Boomer generation or Generation X while most new students entering college are at the trailing edge of Generation Y or the leading edge of Generation Z [7]. Because the core traits of these two groups are fundamentally different, it becomes difficult for faculty to effectively work with students because of differences in learning styles.

At Plymouth State, we have recently taken a long and detailed look at our first semester course for both Computer Science and Information Technology majors: Computing Fundamentals. This paper details the changes made during the course revision in an attempt to address the challenge of teaching computing to this new generation of students and discusses the results observed from the initial offering of the newly revamped Fundamentals course.

STUDENT GENERATIONS

Most students in college today are from Generation Y, but we are starting to see some students from Generation Z. Generation Y (Gen-Y, Generation NeXt, or the Millenial Generation) consists of people born between 1981 and 1994, and Generation Z (Gen-Z or the Neo-Millennial Generation) consists of people born between 1994 and now [7]. These students, sometimes referred to as digital natives, have been exposed to technology their entire lives. They have been inundated with choices and flexibility since childhood and in many cases have grown up with a very different support system than the one most faculty were raised with.

These students exhibit many traits that are positive including: confidence, team-oriented thinking, and a strong sense of achievement [5]. They can be extremely confident in their actions, and are motivated by future success. They naturally are inclined toward team-based activities, and favor in many cases being able to interact and work with others. They also are motivated by achievement, and enjoy a sense of accomplishment on completion of tasks. These are all traits valued by professionals in the computing industry, especially given the increase in the need for employees that have these types of "soft" skills [3], and should be encouraged throughout their course of study in higher education.

There are also several perceived weaknesses among Gen-Y and Gen-Z students [5]. They tend to be very pragmatic, and will adapt to find a solution that works rather than seek out the "best" solution. They can be very skeptical by nature and may question information they receive from a faculty member. There is an expectation of excellence for students and they have been brought up to believe that this success may come with little effort on their part. Students may not exhibit the politeness and deference expected to the faculty. They may express interest only in achieving the minimum expectations associated with a particular goal, and need to understand the importance of a task before
being convinced to undertake it. These students are used to and expect frequent and immediate feedback on their activities.

A misalignment is likely to occur when a faculty member teaching a course, especially at the introductory level, fails to take into consideration the motivations and weaknesses of their target audience (Gen-Y and Gen-Z students) [4]. Modern students are accustomed to gathering information rapidly from various multimedia-driven sources, where faculty tend to deliver information in a more slow and controlled fashion. Based on the modern, hyperlinked Internet and social network model of sites like Facebook and Twitter, students are trained to expect information in a random-access model and receive updates automatically from various sources. Faculty members tend to plan and organize lectures and courses in a very linear and sequential way. Many Gen-Y and Gen-Z students have a strongly visual or auditory learning style, but find most textbooks and lecture material to be text-based. Students are motivated and interested in practical, fun, and relevant activities as opposed to the more standards-based approach of most courses. Students seek instant feedback and gratification for their efforts but most course evaluation tools provide delayed feedback. When a course is misaligned students tend to become uninterested and fail to engage while the faculty member can become quickly frustrated and disenchanted with the students in the course.

A TRADITIONAL INTRODUCTORY COURSE

At Plymouth State we have developed an introductory-level course that is taken by both Computer Science and Information Technology majors during their first semester in the program. Computing Fundamentals (CS 2010) is a three credit course which includes a weekly two-hour lab session. This course assumes no prior knowledge of computing or programming, and is designed to generate interest in the fields of computer science and information technology. The primary goals of the course are to provide students with exposure to different sub-areas of computing, to help students become comfortable with software and technology used in the department, to facilitate critical-thinking skills, and to prepare students for their first programming course. Upon completion of the Computing Fundamentals course students take Programming in Java (CS 2370), which follows the traditional introduction to programming course model.

The major topics for the traditional Computing Fundamentals course were HTML and JavaScript, computer number systems, machine structures and assembly language, diagramming program flow, basic algorithm structure, variables and functions, logic diagrams, recursion, basic object-oriented programming concepts, search and sort algorithms, and basic artificial intelligence algorithms. Each of these topics was offered primarily at the introduction-level and provided a breadth-first model for exposing students to a variety of common computing concepts and issues.

The topics in the course were delivered as a combination of lecture and lab activities. Students would have a weekly lab assignment as well as homework problems to help them master the material. During the lab sessions students had the opportunity to experiment with several different technologies. They developed simple web pages using HTML, wrote simple assembly programs and simulated their execution, played with a simple robot simulator, built and programmed Lego MINDSTORM robots, and
animated 3D worlds with Alice. The intent was for these hands-on experiences to help students grasp the fundamental concepts of computing and algorithm design so when they moved into the Java programming course they would be adequately prepared.

Several issues with students were observed over the course of several offerings of Computing Fundamentals. In many cases students would become overwhelmed trying to learn and master each language needed to complete the labs in a timely fashion. Because each module in the course was very short in duration (only two or three weeks long), students would become frustrated that as they would start to become comfortable with one particular language the course would move on to the next module and they would have to essentially start all over again with a new technology or language. This caused some students to become very discouraged and disengaged during class activities. Another issue that appeared was the lack of interest in some topics because of a perceived disconnect between activities and any real-world application. Students would complain that the lab assignments were pointless and as a result would put in little effort to complete them. Finally many of the students who did continue on in the program seemed unprepared for the concepts covered in the Java programming course, despite having seen them before in the Alice programming language.

<table>
<thead>
<tr>
<th>Year</th>
<th>CS 2010 Enrolled</th>
<th>CS 2010 Completed</th>
<th>CS 2370 Enrolled</th>
<th>Retention Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005-2006</td>
<td>38</td>
<td>27</td>
<td>9</td>
<td>33%</td>
</tr>
<tr>
<td>2006-2007</td>
<td>35</td>
<td>35</td>
<td>14</td>
<td>40%</td>
</tr>
<tr>
<td>2007-2008</td>
<td>27</td>
<td>26</td>
<td>15</td>
<td>58%</td>
</tr>
<tr>
<td>2008-2009</td>
<td>34</td>
<td>34</td>
<td>19</td>
<td>56%</td>
</tr>
<tr>
<td>2009-2010</td>
<td>32</td>
<td>20</td>
<td>15</td>
<td>75%</td>
</tr>
</tbody>
</table>

Table 1. Plymouth State University First Semester Retention Data.

Currently Computing Fundamentals is offered every fall semester and the follow-up Programming in Java course is offered every spring semester. Table 1 summarizes enrollment data from the previous five offerings of these two courses. On average just over 52% of the Computer Science and Information Technology majors who successfully complete the Fundamentals course continue on to the Java course. With the additional attrition seen from students continuing on to their sophomore year, the low enrollments seen at the upper-division level were of great concern. To attempt to address these retention issues, the Computing Fundamentals has been completely re-designed for the Fall 2010 offering of the course.

AN UPDATED INTRODUCTORY COURSE

Based on the observed problem with the misalignment of the course approach used in Computing Fundamentals with the motivations and interests of Gen-Y and Gen-Z students, major changes were made to the course goals and topics. Carter [1] observed
that many students in introductory computer science courses have little or no idea what computer science actually is. In addition we noticed that students were not very committed to producing quality work when they failed to see the reason for learning the material. To try and provide more context for the topics covered in the course, more focus was placed on providing discussion and examples of how the technologies covered were used in industry.

While some of the topics were retained from the previous version of the course, many were dropped and replaced with different topics. Currently the list of topics includes an introduction to computer science and information technology, careers in computing, computer number systems, basic logic and circuits, simple programming concepts, mobile computing, computer security, robotics, artificial intelligence, and simple analysis and design. To help maintain a connection to real-world computing each unit includes discussion of careers and opportunities for the area of discussion. Because Gen-Y and Gen-Z students are motivated by seeing the practical application of their work [6], several of the lab assignments were modified to provide examples of real-world activities. Students develop a simple mobile device application using Google's App Inventor, conduct simple experiments using several different artificial intelligence algorithms, and play with simple security tools to better understand how computer security is implemented in practice.

To address issues with students being unprepared for the programming class and being overwhelmed by learning many different programming languages, the set of languages used in the lab assignments has been streamlined. While some research indicates that taking an objects-early approach can help students better understand programming concepts using a visual environment with drag-and-drop commands like Alice [2], it has been our experience that students are then poorly prepared to switch from such an environment to a text-based programming environment. To provide the students more experience with this model of programming, we chose to use Python for most of the programming components of the course. This allowed for more consistency in learning the syntax of a single language and could more easily be translated to similar constructs in the Java course.

Finally the overall structure and tone of the course lectures was modified to better align with the learning style of Gen-Y and Gen-Z students. Several lectures were overhauled from a one-way lecture to more of a free-flowing discussion model. Students were encouraged to contribute to the conversation and were posed questions frequently as opposed to being allowed to sit passively during the entire lecture period. Additional active learning activities were built in to engage students and more in-class group work was added to drive discussion and help keep students interested.

RESULTS

Plymouth State's newly revised Computing Fundamentals course was offered for the first time during the Fall 2010 semester. There were 34 students enrolled in the course, with about a 50/50 mix between Computer Science and Information Technology majors. Roughly 80% of the students in the course are first-year students, with the remaining 20%
made up of sophomores and a few juniors, most who were switching into the department or were transfer students.

The reaction from students to the new course has been overwhelmingly positive. Compared to previous semesters, student attendance has increased significantly. The students have been more involved and engaged in classroom discussions and several have expressed interest in learning beyond the material covered in the course. Overall grades on both labs and homework assignments have improved on average compared to previous semesters. On multiple occasions students have commented on how much they have enjoyed the course, which was almost never heard of with the previous version of the course.

We have observed a boost in retention based on students continuing into the Programming in Java course for the Spring 2011 semester. Of the 34 students enrolled in Computing Fundamentals, 25 enrolled in the Java course for a retention rate of 74%. We hope to continue to see this higher level of retention for students for future offerings of the Computing Fundamentals course.

CONCLUSIONS AND FUTURE WORK

The Computer Science and Technology Department at Plymouth State University was facing similar retention issues and a lack of interest in introductory course material seen in many institutions. Much of the evidence indicated that part of the problem is coming from a misalignment of the course model with the expectations of the newest generation of students. By restructuring our Computing Fundamentals course to better align with Gen-Y and Gen-Z students, we have noticed an increase in satisfaction, involvement, and retention in our first year courses. With only one offering's worth of data at this point it is too early to draw any conclusions about the impact on overall retention in our degree programs, but the results observed are encouraging.

We hope to continue to track students as they progress through their programs to gather additional data on retention rates over the longer term. In addition we are planning to develop additional metrics for analyzing the effectiveness and impact of these changes in the Computing Fundamentals course. Finally, we have seen similar concerns in Programming in Java, and would like to undergo a similar process to correct any misalignments in that course as well.

REFERENCES


IMPROVING STUDENT PERFORMANCE WITH AN EXTRA-CURRICULAR ROBOTICS CLUB

Charles N. Stevenson
Computer Science Department
Salem State University
Salem, MA 01970
978-542-6616
cstevenson@salemstate.edu

ABSTRACT
We developed an extra-curricular robotics club in order to strengthen the academic performance of our computer science students and to increase their understanding of different career paths that are open to them. Despite the small size or our program, around 100 students, the club has the determined participation of six to eight students, including women and minority students. The club's activities increased the student's practical knowledge of embedded systems by developing low-cost line-following robots using inexpensive RC cars and 8-bit microcontrollers. Some of the technical content created by the club's activities will now be used in the first offering of a robotics course.

BACKGROUND
Trudy Howles describes the educational and cultural background of many of today's computer science majors and the resulting academic problems with which the faculty must contend [1]. These problems include an overexposure to media, poorly understood career paths, and the distraction of part-time jobs. As a result, students are often unable to persist in their studies to graduation. Our computer science majors certainly suffer from these problems, and they also frequently lack an appreciation for the general usefulness of strong analytical skills, or even the value of being a strong programmer.

Many of our majors regard a degree in computer science as a stepping stone to employment as a network administrator rather than as a professional programmer or a
developer of new technology. While these students express an interest in learning to use existing applications, they are relatively uninterested in developing new content, applications, or ideas. Many of these students have never experienced the personal satisfaction that comes from creating new or unique ideas or applications.

Part-time, or even full-time, employment reduces both the time available and their desire to apply themselves to course work. Many students seem to suffer from a poor engagement with their course work because they fail to see its relevance to their later professional careers. They often express uncertainty that the languages that they are learning, such as Java, are actually used for software development.

Their many distractions exacerbate the academic problems that today's students face, which undermines their chances for success. As indicated by Howles, many students doubt the value of formal learning and the abstract nature of computer science further discourages them from continuing with their education[1]. For these students, it is a long way from their programming classes to developing the skills necessary to create useful software applications, and even farther to seeing themselves as contributing to such efforts.

EXTRA-CURRICULAR APPROACH

Our approach has been to improve our student's chances of academic and professional success by mitigating their background problems with extra-curricular activities designed to supplement and reinforce classroom learning. We were successful in increasing student interest in embedded systems and robotics with our student-run robotics club.

The students selected a project for themselves, with faculty guidance, and then developed their own goals. We decided to build low-cost, autonomous robots that were capable of line following for their first project. This project gave the students experience with C programming, microcontrollers, sensor interfaces, motor drivers, and high-level path planning, all topics found in embedded systems courses or other computer science areas.

Our computer science program had slightly more than one-hundred majors last year, and we had the regular participation of about six to eight students, who were typically either junior or seniors. Some of the most dedicated members are woman or minority students, indicating that one possible way for increasing the diversity of our enrollment is to supplement classroom activities with extra-curricular activities that combine social and technical activities. This result may also indicate that such students may be more interested in computer science as a major if they learn that professionals in the field usually work in teams that involve a good deal of social interaction. These students saw a student club that was focused on technical work as a natural supplement to their classroom work, a way to enhance both their credentials and their professional skills.

TECHNICAL APPROACH

Given the student's limited financial resources, it was necessary for the autonomous robots to be constructed at minimal cost. Furthermore, while such a project emphasizes
the interdisciplinary nature of computer science, it was essential to recognize that the students are not mechanical or electrical designers, and lacked the necessary skills to design a suitable platform. Consequently, inexpensive radio-controlled cars, available at major retailers for under six dollars, were selected for the platform.

A similar approach was taken by Brauni [2], who added a custom controller to model cars to construct autonomous vehicles for playing "robot soccer." This custom controller is based on a 32-bit controller from Motorola. In contrast, our students decided to develop low-cost controllers for their cars using 8-bit microcontrollers from the Microchip Corporation, combined with a readily available quad package of half H-bridges for controlling the drive and steering wheels. The microcontrollers, H-bridges, and a voltage regulator were combined on an inexpensive card for under two dollars.

The use of a targeted controller, rather than a commercially developed general controller, kept costs minimal and offered educational benefits. The students were able to develop the controller at an architectural level, rather than simply utilizing a pre-packaged solution with libraries that managed all the low-level tasks.

We selected an 8-bit controller from Microchip for with on-board pulse-width modulation (PWM) module and analog to digital converters. Microchip's line of controllers can be programmed in the C language using a cross-compiler offered by Hi-Tech, and a reduced version of this compiler is available free to students. Microchip offers extensive, free technical training for educators, a resource that was essential for developing our autonomous cars because they taught us how to use the cross-compiler, the PWM module, and the A/D modules.

In particular, we selected the PIC 16F690 from Microchip for our controller, which is a 20 pin, flash-based 8-bit CMOS microcontroller with an internal clock, 12 10-bit A/D channels, and a PWM module with up to four output channels. A closely related product, the PIC 16F687, is described in detail, along with accompanying C source code, by Bates[3].

For the line-following sensor, we used an integrated infrared emitter and detector, the QRE1113 manufactured by Fairchild Semiconductor, which is available commercially in sample quantities for under two dollars. The combined cost for the two sensors required to detect the left and right edges was about twice the cost of the microcontroller and PWM package combined. The total cost of the robot cars was around twelve dollars.

The students adapted existing, open-source C code for use in their robot cars. Several web sites offer sample code for 8-bit PIC microcontrollers [4][5]. To utilize the PIC microcontrollers requires setting various register flags, which beginning students often find confusing. By studying the open-source code, students were able to see examples which they could then adapt to their own use. From their experiences in the club, these students also came to understand that existing solutions often cannot be simply copied, but rather must be understood and adapted for the solution of new problems. None of the existing open-source code could be directly used; rather it had to be modified for our purposes. Additional C code had to be developed to interface the infrared line detectors to the microcontroller.
A portion of the C code that uses the PWM module to control two DC motors is given in Table 1. The first task that this code accomplishes is to set the registers and flags that are provided by the proprietary architecture. All of these registers and flags are referenced by specific names that are peculiar to this particular product line. Without the sample code provided by Microchip and various open-source examples, students would have experienced great difficulty in identifying these settings.

```c
int main(void)
{
    unsigned char input;
    ANSEL = 0b00000001;
    ANSELH = 0b00000000;

    INTCON=0b01000000; // purpose of disabling the interrupts.
    PIE1 =0b00100000;
    init_comms();  // set up the USART - settings defined in usart.h
        init_adc();
    Stop();

/* Set up output ports */
    TRISC = 0;       /* all bits output */
    PORTC =0;
    TRISA = 0x03;

/* Set up PWM frequency and timer */
    PR2 = 0b10000001;
    T2CON = 0b00000111;
    CCP1CON = 0b00001100;
}
```

Table 1. Code Fragment for Setting Microcontroller Registers

In order to move the cars, it is necessary for students to complete the development of higher-level path following software. The club currently has integrated the infrared sensors with the microcontrollers, which are able to execute higher-level instructions to move left or right, or so forth. This software remains uncompleted and not fully tested.

**IMPACT OF CLUB**

Our robotics club exposed students to new career possibilities in embedded systems and created a groundswell of interest in robotics. The club had an active membership of six to eight students during the course of the year, out of around 100 computer science majors.

Two of the club members are now working professionally as embedded systems programmers or about to accept offers of such employment. In both cases, potential employers were interested in their accomplishments as members of the club as well as
their accomplishments as student leaders. With six to eight graduates per year, helping
to place two students in embedded systems companies is a significant accomplishment
for the club.

Our curriculum requires students to complete a two-semester senior project. For
several years, the department faculty were concerned because almost all of the senior
projects were in web design, where we were hoping that students would consider a broad
array of topics for their projects. During the last year, we have been able to get students
to work on projects from a more diverse set of topics, in part because of their experiences
in the robotics club. Three students are currently completing robotics projects, and three
others had either completed or are working on an embedded systems project using the
Microchip line of embedded controllers that they used in the robotics club. Judged solely
on its ability to expose students to different career options, the club has been an
outstanding success.

IMPACT ON ROBOTICS COURSE

We will be launching our robotics course next fall, and some material developed by
the robotics club will be incorporated into the curriculum. One of the early experiments
in the course will be to complete the work begun by their predecessors in the club by
finishing the programming necessary for line-following cars to actually follow lines. Such
work will give students experience in sensor interfacing and writing embedded software
for closed-loop control.

CONTRIBUTIONS TO THE CURRICULUM

The knowledge gained from the robotics club led one student to develop a small
circular autonomous robot, similar in mechanical design to existing products, but based
on the Microchip line of microcontrollers. There are a number of existing inexpensive
robots available for purchase, but their use would require introducing a new product line
into our curriculum. With two out of six faculty members working in embedded systems,
there simply isn't the time necessary to develop and administer laboratory projects using
microcontrollers from several different firms. We have chosen instead to seek the
efficiencies that come from specializing in the products from one vendor.

Our own design of an autonomous vehicle uses both 8-bit and 16-bit
microcontrollers from Microchip Co. The 8-bit controllers drive the twin motors using
the C code previously developed by the robotics club for their autonomous cars. The
16-bit microcontroller interfaces to ultrasonic range sensors, communicates with a remote
controller using wi-fi, and sends commands to the 8-bit controller. The higher-level 16-bit
controller sends motion control commands to the lower-level 8-bit controller, such as to
steer left or right.

This new autonomous robot design, a direct result of the robotics club, meets the
needs of our curriculum better than existing designs. It utilizes a controller technology
that we are familiar with, and its 16-bit processor enables sophisticated laboratory
projects. The downside of this design is that students must assemble it from commercially
available parts, rather than simply purchasing an integrated unit.
PERILS AND PITFALLS

In an undergraduate setting, every student club is going to require a continual influx of new members as the student body naturally turns over due to graduation. However, our club is limited in its ability to attract new members because it is so strongly focused on technical activities that require students to be fairly far along in their educational program. Students who have not completed our C programming course find it difficult to participate in the development of autonomous robots based on microcontrollers that are programmed in C. This is a direct consequence of the club's formation by a group of students who wanted to improve their knowledge of C and embedded systems.

The largest problem is that the club has a much lower priority in the student's lives than their coursework or part-time jobs. The undesirable consequence is that it has been extremely difficult to bring even the simple project of the line-following robot to completion. The primary incentives to completing club activities are personal satisfaction and the hope of placing such accomplishments on a resume.

FUTURE WORK

Our new robotics course will be offered for the first time in the fall of 2011, and our upcoming challenge will be to integrate the classroom and extra-curricular activities of our students. While both course and club will be used to stimulate interest in the other activity, essential development of the autonomous vehicle laboratory portion of the course was done through the club and related activities.

The most important and difficult challenge will be to draw into the club students who are still early in their academic careers. It was noted earlier that some of the most determined members are woman and minority students; perhaps other students in the program would benefit from the social and technical interactions in the club as well. Students may be more interested in pursuing computer science as a major if they realize that professionals in the field spent much of their time working in teams with a good deal of social interaction.

Our extra-curricular technical club can assist students in persisting to graduation if we keep in mind Howles results from her surveying at the Rochester Institute of Technology. She notes that most of their students select computer science for their college major because of a genuine interest in computing, a result that would seem to apply to our majors as well[1]. She also cites a psychology paper that affirms the well-known result that intrinsically motivated individuals are more likely to persist to accomplishing their goals despite adversity than individuals whose primary motivation is external factors. The robotics club will continue to contribute to our curriculum only as long as it taps into and reinforces our student's fascination with computing which is their intrinsic motivation for enrolling in the program.

REFERENCES


INTRODUCTION OF VIRTUALIZATION IN THE TEACHING OF OPERATING SYSTEMS FOR CS UNDERGRADUATE PROGRAM

Kevin Grammer, Jack Stolerman, and Beifang Yi
Computer Science Department
Salem State University
352 Lafayette Street, Salem, MA 01970
(978) 542-7426
kevingrammer@gmail, viper11@gmail, and byi@salemstate.edu

ABSTRACT
For many colleges, virtualization is a low-cost solution for providing hands-on lab activities for computer science courses. This paper describes the design and implementation of a series of projects for an undergraduate operating systems course. By utilizing Linux virtual machines on students’ personal computers, these projects teach reinforce students’ understanding of operating system concepts as well as teach students the basics of virtualization. After completing the projects, students were surveyed about their experience and a summary of their responses is presented here.

INTRODUCTION
The study of operating system (OS) principles is an integral part of the computer science undergraduate curriculum. Beyond lectures and reading textbooks, hands-on activities add significantly to a student’s understanding of computer science topics. The CS2008 Review Taskforce emphasizes that the study of operating systems should include a “laboratory component to enable students to experiment with operating systems” [1]. While some major universities are able to provide dedicated computer labs for this purpose, smaller colleges must use what they have at hand and often cannot provide labs for every course [4, 6]. The challenge for these colleges is to find other ways to provide students with high-quality, affordable hands-on experience [2, 3].

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.
Virtualization offers a lightweight, low-cost, practical solution to this problem. The student installs a virtual machine on his or her personal computer and it runs on top of the computer’s host operating system [12]. Separation between the host OS and the virtual machine allow the students to manipulate these virtual machines without the risk of damaging the host OS [4, 8].

In this paper we will discuss applying virtualization in teaching an undergraduate OS course with hands-on projects. Projects were designed to introduce students to the Linux operating system. The main objectives of the projects were to have students update the Linux kernel, add a system call, and work on process-and thread-related projects in the Linux environment. We have also included the comments and feedback we received from the students in this paper.

BACKGROUND

As personal computer speeds and storage space increase rapidly and OS virtualization becomes more easily accessible and commonly used professionally, colleges are more seriously considering adopting virtualization in the classroom. Some schools have already developed virtualization techniques to bring students more accessibility to lab-based activities for both university students and distance learning students [7]. Not only does it cut lab costs for colleges that use virtualization as a teaching tool, but it gives students necessary exposure to the latest real-world technologies. Most students’ personal computers meet the minimum requirements for running at least one virtual machine, making it a practical alternative to using university computer labs. Many universities have already implemented virtualization to give their students hands-on experience in system-oriented computer science courses such as operating systems, system administration, and networks courses [4, 5, 7].

OS virtualization allows a CS professor to give his or her students exposure to different operating systems that they may not have prior experience with. Rather than simply learning the software of a single OS, using a variety of operating systems emphasizes the fundamental and lasting concepts of operating systems by revealing the commonalities in function between them [2, 10]. In addition, virtualization gives the professor flexibility as to which OS to use to teach each concept. For example, a professor might use Linux to help students understand the distinction between user and super-user security levels [7].

At Salem State University, Operating System Principles is a 3-credit hour undergraduate course. While many CS courses here have separate lecture and lab time, this course has no accompanying in-class lab hours. From our past experience in teaching OS, students often complained of the theoretical descriptions of OS topics and algorithms without lab practice to help solidify an understanding of the internal structure of the OS. Due to the limited resources in our department (students do not have administration rights on the machine) and course setting (no in-class lab time), we must find non-traditional solutions to overcome these challenges— to design OS projects that can be implemented on students’ personal computers.
PROJECT GOALS AND PREPARATIONS

We set the goals of designing the OS projects as: (1) introduction to virtualization — the students will not only learn a new technology but also complete the OS projects without affecting their personal computer environments; (2) one installation is a Linux operating system — the students will get experience with different operating systems than the norm (Windows/Mac) and work with free, open source software (most CS students here have a very limited knowledge of Linux); (3) patching the kernel with a newer version — the students will acquire direct knowledge of compilation, configuration, and construction of Linux; and (4) adding a system call to the new kernel and practicing with the new kernel through a simple C program — the students will have a chance to look inside the OS to obtain a basic understanding of internal structure of one operating system and to reinforce OS principles.

To achieve these goals, we polled the students in an initial survey. Questions from the survey included inquiries about the students’ experience with computers, their computer usage (such as which operating systems and programming languages they had used) and configurations of their personal computers (such as RAM/hard-drive capacity). From the feedback (of 25 replies at the beginning of 2010 Fall), we noticed the following:

- All the students own one or more computers: all students except one have laptops; about 40% have both desktop and laptop.
- Only two students (8% of total) have a computer with 1 GB RAM; 20% with 2GB RAM; 60% with 4GB RAM; 12% with 6 or 8GB RAM.
- 40% of students did not have any knowledge about Linux; 40% have very limited experience with Linux.
- 20% of students did not use any command line environment; 40% had limited experience through the using of Windows’ DOS.
- More than 80% of students used only Windows operating systems.

Previously, we took a similar survey in the Fall of 2009 for the OS class and received very similar results indicating a little lower percentage in the student population that had computers with 2GB or more of RAM.

These statistics provided us with guidelines to design OS projects. First, we decided to use VMware Workstation as the virtualization software package for virtual machines. Our considerations were (1) compared with other virtualization solutions, VMware is more reliable, more widely adopted in industry, easier to set up, and provides more management tools [9] and (2) each student’s personal computer meets the hardware requirements needed to install VMware. We also encouraged students who had experience with virtual machines to use alternative virtualization packages such as VirtualBox.

Next, we made use of the concepts and topics from the course textbook [11] for the implementation of the projects. Thus, students should be familiar with these topics from reading the textbook. We focused on the following points in designing the projects: (1) any student with no background in Linux should still be able to complete them; (2) students should be given a chance to “see” the inside of Linux and to make some modifications to it, and (3) some projects which covered the most important concepts of the OS (such as processes and threading) should be simple enough to be implemented in
both C and Java programming languages (Java is the first and required programming language taught at Salem State University. Most students had little to no prior experience coding in C when taking the OS course).

PROJECT IMPLEMENTATION AND RESULTS

Project 0: Open Source OS and Virtualization

This project begins with a guide to downloading VMWare (or another virtual machine software) and installing the Ubuntu operating system. The Ubuntu file, although readily available for free download on the Internet, is also provided on the university network and distributed on CDs and thumb drives to students in class. Ubuntu is booted within the virtualization software and from here on, students work within the Ubuntu OS rather than the host OS.

After a basic introduction to the Ubuntu GUI and some of its preinstalled applications (Firefox, Open Office, Gimp, etc.), the student is given instructions to use the command line interface. The student tries some commands, such as `cp`, `mv`, and `uname` to get some information about the OS. Next the student makes a directory, changes directories, and lists parent files. Once the student is familiar with some basic commands, he or she is asked to create a `hello.cpp` file and enter some lines of code (provided in the project description). In order to compile and run the code as a C++ program, the student must first download and install the g++ software package from the command line. As the student completes each step of Project 0, he or she takes screenshots for submission.

The goal of Project 0 is to give the students some familiarity with the virtual machine, Linux, and the CLI. The project focuses on how to do a variety of basic operations with the Linux virtual machine, such as how to install a virtual machine running Linux, how to navigate Ubuntu’s GUI, and how to get basic functionality from the CLI. With the completion of Project 0, the student not only has a bit of experience with Linux and the command line, but also has a “how to” reference for the next project.

Project 1: Updating the Linux Kernel and Adding a System Call

This project is a step-by-step guide to updating the Linux Kernel with a later version and adding a system call to the kernel. Much of what the student is asked to do in Project 1 is complex and most likely beyond his or her basic understanding of operating systems at the time when the project is assigned. Project 0 has given them some understanding of the Linux environment and command line to better follow along.

Initially, the student opens up the CLI and logs in as the root user. From here on, the student can write to the system's directories and must be careful not to accidentally change file or directory names. Because the student is doing this on a virtual machine, if they do make an error, they only run the risk of corrupting the virtual machine, and cannot directly affect their host operating system. Project 1 then takes the student through updating the Linux package and downloading the new kernel.

In order to add a system call to the kernel, the student must act as the root user and edit files within the kernel. Project 1 guides the student through each step of the process,
editing the appropriate system files and adding a simple system call function to the kernel. In the case of Salem State University, most OS students have very little experience coding in C and C++. Project 0 gave students a chance to write and compile a C/C++ program. Project 1 requires the students to manipulate C files. The student should gain enough knowledge from Project 0 to successfully manipulate these files and at the same time become more familiar with the structure of C files.

After adding the system call, the student must configure the kernel using the kernel configuration menu. The student loads the new configuration and then builds the new kernel, appending his or her first initial and last name to the kernel version string. The compiling process takes anywhere from two to five hours. By witnessing the compiling, students said they gained an appreciation for how massive a task compilation really is. The student then installs the kernel and tests the newly added system call with a C test program. Although it is only a simple C program, it re-enforces the basics of C and its uses in the Linux kernel. Writing a working C program that makes a student-created system call can be a very powerful tool for convincing Java-based students that at least a basic understanding of the C programming language is important.

At the conclusion of the project, the student submits screenshots of the process to the professor. The greatest satisfaction the student gets is seeing his or her name appended to the kernel version name while getting a good look at the composition of the kernel of how it functions. The student also gains valuable experience utilizing the CLI and gains an appreciation for the root user power of the command line.

**Project 2 and Project 3: Processes and Multithreading in C/Java**

Project 2 builds on the basic C concepts learned from the previous two projects. In this project, students must create multiple processes that communicate with each other using pipes. A message is sent from a parent process to a child process. The message is then modified and sent back to the parent process. Although a rather simple program, it demonstrates synchronization and the implementation of pipes.

Project 3 is a more in depth look at Java threads. Besides giving students experience using threads, this project directly contrasts Project 2’s use of processes. Threads share data in a very different way than processes. Project 3 gives students a basic understanding of threads by designing a Java program using threads that adds and multiplies all the numbers 1 through N. In addition to this simple threading program, the student is asked to then write a more complex program multiplying two matrices. A separate thread is used to multiply each element of the solution matrix. The students are then challenged to rewrite this program in C by using PThread. A more challenging multithreading project — modeling of one of the OS classic examples, Producer-Consumer problem — was given as a bonus project.

**CONCLUSIONS AND FUTURE WORK**

Students showed great interest in these projects, particularly those involving practice with virtual machines and Linux -- updating the Linux kernel and the addition and testing of new system calls. In addition to everyone completing the first two projects, many
students provided positive feedback. In order to better design for future projects, we gave another survey to the students after they completed the above projects. We invited students to critique their learning experiences and we asked for suggestions to help with future modifications of the projects. The following gives a brief summary of the survey.

- Virtualization and virtual machines: more than 80% of students replied that the projects greatly helped them understand the related subjects and use the latest technology. About 85% of students used VMware for the projects, while others used different virtualization software packages with success.
- Linux and open source software: more than 80% of students considered that the projects provided a good opportunity for them to learn Linux and to make use of open source software. More than half of them said that without the projects they would not have even considered using Linux.
- C programming language: half of the students acknowledged that the projects increased their knowledge of the C programming language.
- OS kernel and system call: more than 85% of students responded that the projects helped them not only understand the OS internal structures “in a very elementary way” but also learn the basics of system programming. Many of them recognized that without the projects they could not fully understand the kernel functions of the operating system.
- Additional comments: we also asked students to provide feedback on other aspects of the projects. Many of them recognized the “complexity” of the operating system (it takes about “3-4 hours to compile” — quoted from a survey) and the superior functionality of the Linux command-line to the GUI. They particularly mentioned the thread/process-related projects, which greatly reinforced their understanding of the concepts of process and threading.

Students not only favorably responded to the survey questions but also provided many suggestions that we will benefit from in future versions of these projects. We will continue to use the projects but will spend a bit more time teaching the basics of Linux before assigning the Linux related projects. We will first ask students to do some investigation on the topic of virtualization and adopt different virtualization solutions. We will design more projects in other areas of OS such as multithreading program but continue to have each project build off the knowledge gained from previous projects.

REFERENCES


**ACKNOWLEDGEMENT**

The authors of this paper would like to thank Professor Joseph Kasprzyk, chairperson of the Computer Science Department of Salem State University, for his support and guidance in the writing of the paper. The corresponding author of the paper is Beifang Yi (byi@salemstate.edu).
dLife is a Java library designed to support teaching and research in the fields of Artificial Intelligence, Artificial Life, Robotics and Computer Vision. A central design principle for the creation of dLife was that it be appropriate for beginning students while also being powerful enough for use in upper level courses and research projects. This demonstration will illustrate this flexibility by highlighting uses of dLife from across the curriculum.

The demo will begin with an application to be used in a First-Year seminar titled "Can a Machine have a Mind?" that allows non-programmers to teach a robot to navigate a maze. The structure of a simple dLife robot controller will be shown to illustrate how dLife might be used in a CS1 course when introducing conditional statements. This part of the demonstration will also highlight dLife's interactive features including the live display of robot sensor and effector information. A brief discussion of how dLife can be used when introducing other CS1 content such as variables, arrays, iteration and objects will also be included at this point. Several examples of assignments using dLife in an upper level artificial intelligence course will be described, and time permitting demonstrated live or via recorded video. These assignments may include path planning, vision based object recognition and tracking and adversarial search. The demonstration will conclude with the execution of some research grade code that uses a genetic algorithm to evolve a neural network controller for collision free robot navigation. A list of other research projects in which dLife has been used will be included to further highlight its flexibility.

ADDITIONAL INFORMATION ABOUT dLife:

dLife includes packages for neural networks, genetic algorithms, computer vision and robot control. The neural network package includes feedforward, backpropagation, Elman and CMAC neural networks. The genetic algorithms package provides a generational genetic algorithm with a variety of selection mechanisms and reproduction operators. The robotics packages support the following robots: MobileRobots Pioneer

* Copyright is held by the author/owner.
3DX; K-Team Hemisson, Khepera 3 and Khepera 2; and Sony Aibo. Robot simulations are supported through the Player/Stage system. The computer vision package provides a collection of basic image filters: color matching; blobification; blurring; edge detection; etc. Images for processing can come from a variety of sources: image file (jpg, gif, etc.); sequence of image files (e.g. img1.jpg, img2.jpg, …); streaming video from the Aibo; and streaming video from any V4L device (Linux box required). All of dLife's packages have been designed with extensibility in mind facilitating the creation of new types of neural networks, genetic algorithms, robots, robot devices, image filters and image sources.

The dLife ControlCenter is a GUI application that provides a convenient means of experimenting with the robotics and computer vision features of dLife. Using the ControlCenter, users can see the values of robot sensors, manipulate robot effectors and observe and modify the effects of image filters without writing any code. The ControlCenter also allows users to load a user defined Controller object (a Java class) that can interact with robots and computer vision features programmatically. Typically the ControlCenter is run on a workstation and commands are sent wirelessly to the robot. dLife has been developed and fully tested on Mac OS X and Linux.

**Biographical Information:** B.S., Dickinson College, 1990. M.S., Pennsylvania State University, 1995; Ph.D., Nova Southeastern University, 2005. Prof. Braught's current research interests are in the area of artificial life and robotics. His specific interests include interactions between learning and evolution; co-evolutionary systems; evolutionary and developmental robotics; and biologically inspired artificial intelligence. He is also active in computer science education. His recent projects in this area include: the development of the dLife library to support teaching and research in the fields of Artificial Intelligence, Artificial Life, Robotics and Computer Vision; and an evaluation of the effects of pair programming on student programming ability in the first computer science course.

**Supporting Materials:** The dLife documentation, examples, binary files and source files will be made available via http://users.dickinson.edu/~braught/dlife. In addition a single page flyer highlighting dLife's major features will be available for distribution.
In an operating systems course, students have the opportunity to learn a variety of different algorithms designed to solve problems such as process scheduling and page replacement. Students can learn these algorithms on a conceptual level, but they gain a stronger understanding of the algorithms by implementing them and seeing them work. For students who lack a strong background in C, studying and modifying the code of an actual operating system such as Linux may not be feasible in an undergraduate operating systems course. Furthermore, with the variety of topics than can be covered in such a course, it may not be desirable to assign major month-long programming projects. As an alternative, a simple simulation has been developed for the undergraduate operating systems course at Widener University that allows students to implement and test their algorithms in Java, the language that they are most familiar with.

The students are given the source code of the simulation, which implements a simple algorithm. They are then asked to modify the code to implement a more complex algorithm. For example, students are given an implementation of the First-Come, First-Served scheduling algorithm and asked to implement a multi-level feedback queue. Students may be given an implementation of the First-In, First-Out page replacement algorithm and asked to implement a 2-bit clock algorithm or Least Recently Used. By running the simulation, they are able to see the results in a simple GUI window and verify that their implementation is correct. Students are given one to two weeks to complete the assignment, depending on the difficulty of the algorithms they are asked to implement.

The simulation is not complex and only simulates the algorithm implemented by the student. These assignments have been used effectively for the past three years. Students have successfully completed the assignments in the given time frame and have demonstrated a greater understanding of the algorithms.

* Copyright is held by the author/owner.
During the demo, recent assignments given to undergraduate operating systems students will be presented and sample implementations of scheduling and page-replacement algorithms will be demonstrated.

BIOGRAPHY

Adam Fischbach is an Assistant Professor of Computer Science at Widener University. He has a Ph.D. in Computer Science and Engineering from Penn State University and has taught an undergraduate operating systems course each fall for the past five years.
iJava - AN ONLINE INTERACTIVE TEXTBOOK FOR ELEMENTARY JAVA INSTRUCTION

DEMONSTRATION

Robert Moll
Department of Computer Science
University of Massachusetts
Amherst MA 01003
moll@cs.umass.edu

iJava is an online interactive textbook for elementary Java instruction. Its principal feature is an embedded evaluator, which means that throughout the text students are asked to solve simple coding problems that reflect textbook material. The text is integrated with a remote-server-based automated homework system and learning management system - the OWL system.

The iJava text is online, and is organized around a set of ~ 175 embedded questions. These are mostly very simple coding problems. They are tied to the surrounding text. The work cycle goes this way: a student reads a few paragraphs, and then works one or several exercises that are tied to the just-discussed material. Since problems are easy, answers come quickly, students are patted on the back, and - most importantly - material leading up to the embedded problems has been read. The embedded problems have a due-date, and to ensure that students keep up, students get triple extra-credit for timely completion. Kids in computer classes are maniacal about "points", and by and large they can't resist a triple credit offer.

iJava runs from a remote server (right now at UMass) and on that server records are kept of every action a student takes, for example how many times a particular problem has been attempted, when attempts were made, and what those attempts are.

The embedded question component is the crucial and unique feature of iJava - no other book, to my knowledge, works in this way. The four figures below illustrate how these problems work. However the book also relies on a number of other interactive features that make the book more lively - short movies, mouse-over interactions, and so forth.

* Copyright is held by the author/owner.
In addition the book is supported by a corpus of 200+ automatically graded homework problems. These regular (and generally more difficult) problems are divided into weekly problem sets that are tied to chapters in the book. Results here too are recorded on the remote server.

While the class is demanding, it has greatly improved retention rates for our introductory Java class. Only 8-20% of students do not pass my final exam.

Finally, the class has done a good job of: 1) making student more effective readers of technical materials (most students do indeed read the book); 2) making students independent learners - less than half the students ever come to lecture, since the book is sufficient for most students to learn Java on their own, and on a final survey, more than 55% of students said the course did indeed make them better independent learners; and
3) arousing and keeping students interested in programming - though computer science majors make up just 20% of my beginning class this term, roughly 50% of the class has signed up for our next CS class, Data Structures.

Robert Moll is Associate Professor and Associate Chair for Academic Programs in the UMass Computer Science Department. He is the author of five CS texts and two children's books, He holds a PhD in Mathematics from MIT (1973) and has had a long-standing interest in CS teaching and instruction.

The first three chapters of the textbook can be viewed at http://ijava.cs.umass.edu/
Nearly any topic in a data structures or algorithms course will be more interesting to undergraduate students if they can apply what they are learning using real-world data and visualize results in a meaningful way. This demo will show how to use highway data obtained from the Clinched Highway Mapping (CHM) Project [3] to explore graph structures and algorithms and to display results using the Google Maps API [1].

The CHM Project has gathered information about the routes of highways in North America and Europe. The project allows travelers to track the segments of roads they have traveled (or, in the project's terminology, "clinched"), to see representations of their travels in map form, and to compare the extent of their travels with those of other project users. The data that describes the highways needs to be at a fine enough granularity to allow users to specify their travels with a reasonable level of accuracy, but coarse enough to be able to present (in both list and map form) the endpoints of road segments available to be marked as clinched. Each route is added to the project ("plotted") by listing the latitude/longitude coordinates of the route's endpoints and at major intersections along the route (the "waypoints" of the route). Project volunteers plot the highways at an appropriate level of granularity using public domain data sources such as route logs from state departments of transportation, map data from the OpenStreetMap project [2], and government satellite images.

The CHM data for a highway system is used to generate a graph that represents that highway. A preprocessing program developed by the presenter reads the set of waypoints for each route and matches up shared intersections based on coordinates. The waypoints (of all routes in a system) are the graph vertices and the road segments connecting adjacent waypoints are the graph edges. Each vertex is labeled with a string based on its route name(s) and waypoint name and stores its coordinates; edges are labeled with the
name of the route that connects its endpoints. Distances (edge lengths or weights) are computed easily from the coordinates.

The highway systems available in the CHM project vary in size, meaning graph data can be generated at a variety of scales. For example, the Interstate highways in Hawaii generate a graph with only 47 vertices and 48 edges, while the New York Interstate, U.S., and state highway systems combine to form a graph with 7265 vertices and 8416 edges.

These graph data files can be used as input for class examples or assignments in a data structures or algorithms course. This data was used in a laboratory assignment for a data structures course at Mount Holyoke College where students were required to build a graph structure representing a highway system then perform Dijkstra's algorithm to compute the shortest route between two specified waypoints. Student programs generated output files listing the road segments along the shortest path. These were then uploaded to a course web server, where an instructor-provided program used the Google Maps API to visualize their results. Students were able to develop and debug their programs using small data sets like the Hawaii Interstates, then use those programs to compute much more complex routes using the larger data sets.

The software has been updated and extended for use during the Spring 2011 semester at Siena College in an algorithms course. Many more highway systems have been added to the CHM project and those will be made available to students.

DEMONSTRATION PLAN

The "demo" presentation will begin with a brief overview of the CHM project and of a selection of graph algorithms that are appropriate for use with the CHM data. It will then describe the CHM data formats, the desired graph data format, and the process used to convert CHM data into these graphs. Next, it will focus on the Javascript program that plots the graph data either in its original form or as output of graph algorithms such as Dijkstra's algorithm. The presentation will then demonstrate examples and assignments that use the system, from both the instructor and student perspectives.

PRESENTER BIO

James D. Teresco earned his Ph.D. in Computer Science from Rensselaer Polytechnic Institute in 2000. He has held faculty positions at Williams College and Mount Holyoke College and is currently a member of the faculty in Computer Science at Siena College.

SUPPORTING MATERIALS

At the time of this writing, available web resources include the lab assignment page (http://courses.teresco.org/cs211_f09/labs/maps/) from the first assignment to use this project, and the mapping site used by the students to visualize their results from that assignment (http://courses.teresco.org/cs211_maps/). A new site is under development that will 1) more thoroughly describe the project from the instructor's point of view and 2) provide additional tools to be used by students in the Spring 2011 algorithms class to
visualize their own results in the Google Maps API. The new site will be available before the demo presentation and its URL will be given at the demo. It will include all the assignments and examples that make use of this project. The software that converts CHM data into graph format and the Javascript programs that plot the data in the Google Maps API will be made available on the instructor portion of this site. They are currently available from the presenter upon request.

REFERENCES


THAT'S NEAT -- HOW DO I DO IT?

DEMONSTRATION

Alice E. Fischer  
University of New Haven  
afischer@newhaven.edu  
80 Killdeer Road, Hamden CT 06517

This presentation is about teaching techniques for a course that introduces GUI interfaces and event-driven programming. The material was developed for a sophomore-level Java course but could be adapted to other languages or other levels.

As student programmers advance, the programs they study and those they must write become more complex. At this level, it becomes both impossible and undesirable to develop an entire program in class. Impossible because programs span two or three files and become too long for one class period. Undesirable because all event-driven programs have large portions that are just like all the others, and therefore not appropriate to cover repeatedly in lecture. The instructor, then, needs to show how each new concept or technique fits into a larger, relatively routine, context.

Even simple programs in the areas of GUI programming, event-driven programming, and multimedia require use of two or more Java classes. Further, elements relating to one new concept can be distributed in different parts of those classes. Presenting the required elements could be both boring and confusing. I use a demo-first method to unify the material while, simultaneously, providing a way for students to engage with the new topics. This approach is especially helpful when used to make abstractions visible.

PRESENTATION METHOD:

• Start each class with a colorful demo that illustrates the day's topic. Then ask, "Do you know how to do this?", "What's new here?". Relate what they see to what you want to teach.
• Briefly introduce a new feature illustrated by the demo and show the portion of the demo code that implements the new feature. Stress usage and semantics, not just syntax.
• Elicit questions and suggestions for changes. Compile and run the results.
• Summarize the new material, adding anything the discussion has missed.

Copyright is held by the author/owner.
When a demo introduces more than one distinct concept (e.g. layouts and button clicks), they should be presented in sequence.

Two examples (boldface) from the following list will be briefly presented. All have been used in my class and are available on my website at http://eliza.newhaven.edu/ccsc11/:

1. Moving: concurrency, event-driven, interactive programming, fonts and screen coordinates. (Visualize screen coordinates and events)
2. Caterpillar: colors, timers, multiple windows, awt drawing primitives.
3. Balloon: colors, shapes, scale, buttons
5. Tree: layouts, check-boxes. Responding to checkbox events.
6. Egg: mouse, window, and keyboard events. (Visualize events)
7. Pumpkin: class derivation and polymorphism. (Visualize derivation)
8. Tic-Tac-Toe: multimedia
9. Cluck vs. Hiss: threads, an interactive game

ASSESSMENT OF THIS TEACHING METHOD:

The simple but fun examples help minimize lecture time and maximize learning by questioning. The approach has been well received by students and leads to minimal or better mastery for almost all of them. This year, the final exam included a practical portion (2 hours) during which students were asked to create a GUI with controls + an animated color graphic scene. All students succeeded in doing either the color graphic portion or the GUI portion. Half of them completed both parts.

Students like this mode of presentation. They are enthusiastic about the material being presented and make positive comments on course-end surveys. A few examples:

- I loved learning Java.
- The most enjoyable thing is writing programs you can see working.
- The computer code examples are helpful.
- I got a good understanding of object-oriented design.

SUMMARY

We are using a demo-first method, with color-graphic demos:

- To motivate students to engage with the material.
- To help then visualize concepts.
- To make the class time fun (at least a little bit).
- To focus attention on one or two sets of concepts per class period, even when those concepts are embedded in a complex program.
- To provide a complete program that can be modified in class, to answer student questions and experiment with variations.
- To develop each student's appreciation for what they can do with Java.
HANDOUTS FOR THE TALK:

- Summary of the goals and the method.
- For the demos presented in the talk, a page, each, listing the teaching objectives and showing two or three views during execution.
- A URL and index to the website supporting the talk, listing all the demos and their purposes.

BIOGRAPHY:

Alice E. Fischer
Professor of Computer Science, University of New Haven, West Haven, CT
Areas of interest: programming languages, software design, systems programming.
BA in Mathematics, 1964, University of Michigan,
MS in Applied Mathematics, Harvard University, 1967
Ph.D. in Computer Science, Harvard University, 1985
EXPERIENCES WITH COMMUNITY-BASED PROJECTS FOR COMPUTING MAJORS

Jeffrey A. Stone
Pennsylvania State University
200 University Drive
Schuylkill Haven, PA 17972
570-385-6267
stonej@psu.edu

Elinor Madigan
Pennsylvania State University
200 University Drive
Schuylkill Haven, PA 17972
570-385-6267
emm17@psu.edu

ABSTRACT

Projects which compel students to work in team situations with community partners are one method for building so-called "soft skills" in computing majors. The use of team-oriented, community-based service learning projects can provide meaningful and productive experiences for students but must be selected and managed carefully by faculty. This paper discusses a sample of recent community-based projects at one campus of a large, multi-campus research university. Experiences with community-based projects involving Web design, systems design and implementation, and digital storytelling are discussed, as well as the benefits and challenges encountered in the projects.

INTRODUCTION

There has been a growing recognition in recent years that so-called "soft skills" - teamwork, oral and written communication, and social skills - are becoming increasingly important for graduates in the Computing Sciences. Projects which compel students to work in team environments with community-based partners are one method for building these skills. The use of team-oriented, community-based projects throughout a four-year degree program may also be a powerful tool for retaining majors as well as providing students with a competitive advantage for future job searches. These service learning projects can provide meaningful and productive experiences for students but must be selected and managed carefully by faculty. This paper will discuss a sample of recent
community-based projects at one campus of a large, multi-campus research university. The benefits and challenges encountered in these projects will also be discussed.

BACKGROUND

Community-based projects are undertaken for many reasons. For example, these types of projects foster positive relationships between the university and the surrounding community, provide a tangible means for students to relate their studies to the "real world", and allow faculty members to integrate students into community-based public scholarship projects [2]. "Service Learning" has become an institutionalized part of university life at some colleges and universities, offering opportunities for students to serve local communities while providing meaningful learning experiences [1]. Computer Science Education has recognized the value that community-based projects/service learning can offer to computing majors ([3], [6], [7]). Prior research into service learning and community-based projects for Computer Science students has focused on experiences with "traditional" computing projects such as systems analysis, design and development [6], database development [5], and software engineering [3]. However, Sanderson [4] has argued that further work is needed to offer guidance on how to effectively integrate service learning into the CS curriculum.

This paper will describe several community-based projects undertaken by computing majors at a single campus of Penn State University. These projects were undertaken in a rural area of Pennsylvania where the predominant local employers involve educational services, health care, and social assistance. The students were enrolled in Penn State's Information Sciences and Technology (IST) program, which combines elements of Computer Science and MIS. The IST program is active learning-oriented and stresses teamwork and oral and written communication skills in addition to technology skills; as a result, projects involving community-based partners are common across the curriculum. This paper describes a sampling of these "real world" projects, including a discussion of the benefits and challenges of these projects.

COMMUNITY-BASED PROJECTS WITH DIGITAL STORYTELLING

One course in the IST program where community-based projects have been integrated is entitled "Information and Organizations" (IST 301). The purpose of this course is to introduce IST students to the more basic elements of business and organizational behavior through the "lens" of information technology use and integration. This junior-level course is unlike any major-specific course IST students will have encountered. Prior to this course students will have completed more traditional technology-oriented courses in C++ and Java programming, databases, networks, calculus and discrete mathematics. Students typically take this course in the same semester as a complementary course in Project Management. These courses are meant to prepare students for future coursework in systems analysis, design, and integration.

Beginning in the fall of 2009, IST faculty at the Penn State Schuylkill campus partnered with a local community revitalization organization (Schuylkill VISION) to build digital storytelling projects into the IST 301 course. The goals of these projects
were to provide our students with a "non-traditional" set of technology projects and to provide a means by which students could collaborate with community-based partners and exercise their creativity. These projects were also tied to a campus teaching theme for 2009-2011, "Diversity within Community", which strived to showcase the diversity inherent within the campus community as well as the surrounding communities. The following sections describe two main types of projects which resulted from this endeavor.

One method of integrating community-based digital storytelling into the IST 301 course focused on event-driven stories. At the request of Schuylkill VISION, IST 301 students were assigned a series of local community events to film for fall 2009. Each team of three students constructed a 3-5 minute digital story for each of four individual community events. Students attended and filmed the events in addition to interviewing important stakeholders. These events included such things as town craft festivals, Halloween parades, and charity fundraising events. The events were chosen for their diversity, both in terms of the host community and the event's content. The result was a series of short, Web-based videos which could be used by Schuylkill VISION to highlight the diversity of the local community.

Digital Stories as Oral Histories

During the fall 2010 semester another group of five IST 301 students embarked on a larger digital storytelling project than had thus far been attempted. The project, done in partnership with two local non-profits, involved constructing a digital history of the local shoe manufacturing industry. In the 20th Century, Schuylkill County, Pennsylvania had one of the largest shoe manufacturing industries in the United States. The industry has since declined, but one of the former shoe manufacturing facilities was recently turned into a local Arts Center. The Arts Center management asked our students to build a set of short digital stories around an oral history of Schuylkill County's shoe manufacturing industry. The purpose of these videos was to educate the public about this rich heritage through archival materials and first-person oral histories of former shoe industry workers.

The shoe manufacturing industry project provided students with an opportunity to learn about many different academic and practical research methods, including historical research. Since the industry has long since declined, much of the historical information necessary to tell these stories could not be found via a simple Google Search. Visits were made to local historical agencies, campus archives were searched, and digital library databases were scoured for information. In addition, students were given the opportunity to interview former workers from the local industry as well as individuals whose families had a long history of involvement with the industry. These interviews - conducted both individually and in a focus group - formed the basis for a series of short stories on the Schuylkill County shoe industry. One video involved a summary of the industry, while others were shorter, more focused stories using interviews with former shoe workers.

Challenges and Benefits

The challenges involved with these digital storytelling projects were significant. The primary challenge was training and equipment. The students were by no means professional videographers. The class involved a non-trivial amount of training in the
technical use of the video equipment (digital video cameras, microphones, iMovie) as well as the more stylistic aspects of video production (proper lighting, ambient sound management, shot framing). Thankfully, a local professional videographer provided both tutorial materials and in-class advice for the students. The limited amount of equipment available to the students was also a challenge: only a few cameras and microphones were available to students through the campus library on a first-come, first-serve basis.

The experience was a change of pace for students. Unlike the traditional projects they experience in other courses (e.g. software development, systems analysis) these digital storytelling projects provided students with a window into their community, its people, and its events. Teamwork, communication, and client interaction were critical project elements. These projects also allowed the more creative-oriented students to flex that creativity using multimedia and "new" technology - e.g. iMacs, iMovie, and other multimedia software. As word spread about the Shoe project in the wider community, other parties (former shoe industry workers or their relatives) contacted the instructor offering their services as sources. Consequently, the Shoe project is ongoing.

More than anything however, these digital storytelling projects provided a vehicle for students to see how multimedia technology can be used to impact their community. The digital story videos involving event synopses allowed various non-profit and community groups to spread information about their activities to a wider audience than otherwise may have been available. The digital history of the Shoe Industry provided a means by which the local area can learn about and share a rich manufacturing history. Students were thus able to realize the ways in which computing and multimedia technology can be used to educate the community and achieve a wider social impact.

COMMUNITY-BASED PROJECTS WITH WEB AND SYSTEMS DESIGN

Community based projects help to connect student's academic work to the community needs. The IST program at the Penn State Schuylkill campus partners with a variety of local small businesses and non-profit organizations as well as campus units. One of the more common requests from these organizations is to provide some level of Web design, integration, and e-commerce services. For these projects, one key challenge is the balance between what the community partner needs and the skill set of the students. In our Web Design course, the majority of the students have no background in Web-based languages. Most students have never created a Web page nor do they have any design skills. Our community partners need to have patience and must be willing to work with the students for both sides to have a positive experience.

Through careful management of the student-client relationship by faculty - as well as positive results from the student teams - community-based Web design projects have been a successful tool for engaging and retaining students. For example, one team of students designed a small site for a local farmer ("The Veggie Stand"). The team not only did the basic site design, but they also researched and recommended a Web hosting service. Once the site was complete the team uploaded it and showed the farmer how to maintain it. The site is still active and the farmer is still pleased with the efforts of these students. Small efforts such as these have enabled larger projects to be undertaken, most often with successful results for the client and positive learning outcomes for students.
Emergency Management Planning System

Students in the IST program have a senior-level, writing-intensive capstone in systems design and integration. This capstone course requires students to design and implement a usable IT system for external clients. One of the most successful of these projects involved a partnership with the Schuylkill County Emergency Management Agency (SCEMA). The project goal was to produce an online Emergency Operation Plan database and report generator. These plans are mandated by the Federal Government and each Pennsylvania municipality must create and maintain this document. Each municipality in Schuylkill County was expected to have access to the new system. The system was to be used to allow municipalities to enter all required data for shelters, dams, bridges, essential personnel and critical infrastructure. The system would then take that data and dynamically create the state- and federal-required document.

The SCEMA project began with a two-student team but, due to unforeseen circumstances with one of the students, ended with one student doing the bulk of the project with faculty support. The resultant system was completed on time and worked as originally intended. SCEMA then asked the student and the instructor to demonstrate it at the annual Pennsylvania Emergency Management Agency meeting that fall, where the system was well received. Unfortunately, it became obvious that some technical problems with the system and errors in the initial requirements from the client would necessitate some further project work. Despite this disappointment, this continued work will be used as an additional course project in the near future.

Small Projects Can Bloom Into Larger Student Opportunities

Not only can community partners provide classroom projects for faculty, they often provide internships for students who work on these projects. For example, a local cemetery was looking for a Web site and requested some help from the IST students. As faculty talked with the cemetery about their requirements, it became clear that what the cemetery envisioned was larger than a simple class project. As a result, the project was transformed into an internship experience for one student. The student assigned to the project quickly became attached to his work and has continued working with the client well beyond the time of his internship. The student has created a number of information products for the client's Web site - video content, a slideshow, downloadable documents, and an online donation form. The student will soon have spent two academic years working on this project, with one of those years as a volunteer.

Another successful project that started out small was a partnership with the local county Cooperative Extension Office. The Horticulture Educator asked the IST program to help put a map of local farms online. The Web Design students assigned to this project decided to create an "experience" for Web site visitors. Students researched different methods of providing views of local farms and their produce. The students ultimately decided upon a Google Maps solution. The students' Web site contains a map with pushpins for the different local area farms. When a pushpin is selected by the user a popup provides more information about the farm and a link to the farm Web site. Should the farm not have a Web site, a simple page containing database-driven information about the farm appears. The students also created a user manual (using Google Docs) as well
as administrative Web pages which allow the Horticulture Educator to maintain the site. This successful project has also lead to a related project about the trees on campus.

COMMON CHALLENGES IN COMMUNITY-BASED PROJECTS

There are several challenges involved with utilizing community-based projects for computing majors. Finding projects that both fulfill an external organization's needs as well as providing a manageable and productive learning opportunity for students can be challenging. Undergraduate students are not yet professionals, and depending on their academic progress may have limited skills for a given project. A certain amount of student training may be necessary to prepare students for "real-world" expectations, standards, and client interaction behaviors. Faculty members must therefore be careful to manage the expectations of community-based partners, choose projects that align with course goals and student skill levels, and provide preparatory training for students.

Faculty should decide on the acceptance criteria for proposed projects in advance. These criteria will help to "weed out" unsatisfactory proposals from local organizations. These criteria may include pedagogical goals as well as the nature and past history of the client. Exceptions can and should be made; for example, we will take on a lesser project for a non-profit that is considered to be making a difference in the local community. The ability of faculty to get the word out to the local community is also critical. One method is for faculty to actively participate on the boards of community organizations. This visibility heightens the profile of the program and can lead to quality projects. Faculty should also ensure that their academic administrators and public relations staff are aware of their community-based activities.

Faculty should be aware of the significant time commitment involved in managing these projects. Communication with student teams is critical: status reports and managed deliverables are two methods by which faculty can keep communication channels open. Perhaps the biggest communication challenge involves community partners. Local organizations do not always realize how time consuming it can be to work with students, many of whom may be learning the technology needed for the project. Sometimes community partners fail to be communicative with student teams - e.g. they will not answer emails, return phone calls, or do not provide the necessary information/materials to complete the project. Faculty must sometimes intervene to get the project to move forward and should have a backup plan if the client abandons their responsibility. Of course, there will also be student teams who have grandiose ideas or simply do not follow through, and faculty must be prepared to deal with those situations as well.

CONCLUSION

Community-based projects partnerships can be a win-win for computing programs, students, and community partners. Students are able to use classroom skills while helping their community, computing faculty and programs can establish important community relationships, and community partners gain the benefit of technology services at little or no cost. However, the experiences described in this paper demonstrate that careful faculty management of these projects is required.
REFERENCES


CHANGING PERCEPTIONS OF COMPUTER SCIENCE AND COMPUTATIONAL THINKING AMONG HIGH SCHOOL TEACHERS

Patricia Morreale  
Department of Computer Science  
Kean University  
Union, NJ 07083  
pmorreal@kean.edu

David Joiner  
NJ Center for Science, Technology, and Mathematics  
Kean University  
Union, NJ 07083  
djoiner@kean.edu

ABSTRACT
Interest in specific college majors begins in high school. The influence of high school math and science teachers is important, both for high school preparation of future computer science undergraduates, and for the understanding and interest in prospective majors high school teachers can encourage in their students. With the simultaneous decline in high school computer science curriculum offerings and undergraduate majors in computer science, an effort has been made to update the perception of computer science and computational thinking among high school teachers by hosting local non-residential summer workshops for high school teachers and their students.

Pre- and post- workshop surveys show that the summer workshops have changed the perceptions high school teachers had regarding careers in computer science, resulting in a 50% increase in the likelihood that a high school teacher would recommend computer science, computational science, or information technology to their students as a career. By changing teacher perceptions of computer science and computational thinking, high school teachers are more likely to include computer science as one of the college majors suggested to their students.

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

Exposure to computer science and computational thinking is frequently absent from the high school experience in the United States, due to other curriculum expectations, budget constraints, or teacher experience. One theory is that today’s classroom teachers are not aware of the materials available to them for use with students and 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]. The larger topic of K-12 teacher certification in computer science is not considered here. Rather, an experience report on university faculty interaction with high school teachers, focused on changing teacher perceptions of university study in computing and computer science, is provided. By holding summer workshops for high school teachers and their students, a public comprehensive university has updated and enhanced ideas current high school faculty have regarding applications of computers in the sciences and the utility of computational thinking in high school. Pre- and post-workshop surveys of teachers and students have shown that math and science teachers are more likely to recommend computer science, computational science, and information technology to their students as a career choice after having attended a summer workshop.

BACKGROUND

The decline in computer science (CS) majors in the United States is known [4, 5]. An additional challenge these days is the removal of computer science and information technology courses from high school curriculum offerings in states where a CS/IT course is not required for high school graduation. Often, CS and IT high school offerings have been provided as mathematics or business technology electives, and with the nationwide decline in federal and state funding for local public school systems, CS and IT electives have been reduced or, in many cases, completely removed from high school curriculum offerings, further limiting opportunities for high school students to be exposed to computer science and computational thinking prior to college. Over the years, a number of university outreach efforts have been used to attract students to the major, with little success. University faculty are challenged when it comes to keeping high school 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? 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 for teachers only, such as those held by the CS4HS group over the past five years [6], demonstrate the utility of teacher workshops in enhancing teacher effectiveness in high school computer science education. The week-long non-residential workshop presented here is more inclusive, as math, science, and computer science teachers from high schools are welcome, not just computer science teachers. Additionally, the
workshop outlined here includes teachers who would otherwise not been able to travel away from home for a week of professional development to attend, but are able to attend a regional event which they can commute to daily. A unique feature of the summer workshop presented here is that teachers are encouraged to bring up to four students from their high school. This gives teachers attending the workshop an opportunity to use materials provided with their students, resulting in immediate feedback on the utility of the materials offered and student experiences specific to the teacher's actual environment. As a result of this thinking, a week-long university-based summer program for high school math and science teachers and their students was developed. Teachers were offered continuing education units (CEUs) if they attended and a modest stipend for their summer participation.

RECRUITMENT AND ACTIVITIES

The goal of the summer 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 "10,000 school project" [7, 8] 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.

Recruiting Summer Workshop Participants

With the professional development model in mind, summer workshop notices were circulated to potential workshop participants, with a note that teachers could bring teams of up to 4 students with them for the week-long workshop. The workshop notice was directed at high school math, science, and computer science teachers and the workshop participants had primary responsibilities teaching mathematics, physics, chemistry or biology. Secondary school budget cuts have removed many computer science teachers from high school programs in the region.

In addition to the notices being sent to students, a mailing list of SAT students in the region was also used, with the request to the students that they consider forming a team of four students, and bring a teacher to the summer workshop. The student appeal was very effective.

Summer Workshop Activities

The summer workshop was scheduled in the weeks immediately following the conclusion of the academic school year. The structure of the summer workshop included a group event in the morning, such as a demonstration or illustration of a scientific concept, followed by a lunch break, with individual or small team activities in the afternoon, reinforcing concepts presented in the morning.
Table 1. Summer Workshop Activities

<table>
<thead>
<tr>
<th>Software Tool</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>VenSim</td>
<td>Easy Java Simulations</td>
</tr>
<tr>
<td></td>
<td>Predator-Prey</td>
</tr>
<tr>
<td>Easy JavaSim</td>
<td>Falling Rock</td>
</tr>
<tr>
<td></td>
<td>Bouncing Ball</td>
</tr>
<tr>
<td></td>
<td>Earth-Sun Model</td>
</tr>
<tr>
<td></td>
<td>N-Body Model</td>
</tr>
<tr>
<td></td>
<td>Harmonic Oscillation</td>
</tr>
<tr>
<td>Supercomputer Environment</td>
<td>Running N-Body Simulations</td>
</tr>
<tr>
<td></td>
<td>Creating Animations</td>
</tr>
<tr>
<td></td>
<td>Stereo Project - 3D</td>
</tr>
</tbody>
</table>

The primary material for the summer workshop was modeled after materials developed and successfully used at Shodor [9]. In addition, a tour of the campus supercomputer, and individual projects are included. Table 1 lists the Summer Workshop activities.

The activities were selected to illustrate different scientific and computational concepts, taking into account that teachers and students are from a range of science and mathematics backgrounds.

SUMMER WORKSHOP SURVEY RESULTS

The summer workshop includes science and mathematics teachers, with their teams of up to four students per teacher. This provides the teachers with not only an opportunity to acquire new ideas, but also an opportunity to use the new ideas immediately, with students whose learning skills and styles are familiar to the teachers. By having new curriculum materials and activities, as well as an opportunity to use the new materials with students outside their formal classroom, it is hoped that the teachers will be receptive to incorporating the ideas into their actual classroom materials during the next academic year.

The student teams were made up of students from the teacher's most advanced class or a group of students who expect to continue on in the next year with a specific science teacher. Often, teachers reported that they had been recruited by their students for the event, as the student teams formed first, usually from the appeal to the SAT e-mail list, and then looked for a teacher to attend. Surveys were administered to both students and teachers before the summer workshop and at the conclusion of the workshop in 2010. The results are summarized here.

Student Survey Results

Due to the non-residential nature of the summer workshop, all students and teachers attending were from the surrounding urban and suburban environment. The students were 40% male and 60% female, and were evenly split between those that had just
completed their sophomore year and those that had completed their junior year. The student participants were all from public high schools and did not have any prior programming language knowledge.

The student post-survey results ranked the VenSim Easy Java Simulations and Predator-Prey highest, closely followed by the EasyJavaSim materials (Bouncing Ball, Harmonic Oscillation, etc.). Students were surprised by the wide variety of programs which were available. A table of selected comments is provided in Table 2.

Table 2. Student Post-Workshop Survey Comments

<table>
<thead>
<tr>
<th>Q: How has the Summer Workshop changed your perceptions of computer science and computational thinking as a learning tool?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: I had no idea how complex computer science really was until I participated in this workshop.</td>
</tr>
<tr>
<td>A: It made me realize it (computer science) can be fun and somewhat easy.</td>
</tr>
<tr>
<td>A: Computational science is vastly applicable.</td>
</tr>
<tr>
<td>A: It definitely raised the importance of computer science and computational thinking for me.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q: How has the Summer Workshop experience changed your perceptions of student team learning in the sciences as a learning tool?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: It is effective in small groups.</td>
</tr>
<tr>
<td>A: It made me realize the power of student team learning.</td>
</tr>
<tr>
<td>A: Student team learning is a great way to solve problems.</td>
</tr>
<tr>
<td>A: Teamwork seems very beneficial to such type of work.</td>
</tr>
</tbody>
</table>

Overall, all students felt they had learned more about computing and that computational thinking is important. The importance of computational thinking to math and science in high school was also strongly supported. The Summer Workshop was highly recommended to others. Individually, the post-workshop student survey showed more enthusiasm for computer science, computational science and information technology as a career, as well as interest in applying to the host university for college than had been demonstrated by the students in the pre-survey administered before the workshop.

Teacher Survey Results

The teachers attending did not teach computer science courses or languages. Half the teachers had been invited to attend the workshop by students - the student teams assembled, based on the appeal to the SAT email list, and then the students looked for a teacher to accompany them. This was quite interesting, as it showed strong self-motivation on the part of the students, despite the student interest in mathematics and scientific areas other than computer science. Teachers enjoyed the modules presented as the students had. Table 3 provides representative examples of post-workshop teacher comments.
Table 3. Teacher Post-Workshop Survey Comments

<table>
<thead>
<tr>
<th>Q: How has the Summer Workshop changed your perceptions of computer science/computational thinking as a learning tool?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: I can do it!</td>
</tr>
<tr>
<td>A: Should become more a part of our existing high school curriculum.</td>
</tr>
<tr>
<td>A: Various computational techniques can be used to solve complex mathematical relations.</td>
</tr>
<tr>
<td>A: It has enhanced my ability to use different visuals to explain certain concepts to my class.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q: How has the Summer Workshop experience changed your perceptions of student team learning in the sciences as a learning tool?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Reinforce my positive perceptions.</td>
</tr>
<tr>
<td>A: The techniques learned can be used in almost any science field.</td>
</tr>
<tr>
<td>A: That team learning may be the better way to learn and present certain concepts.</td>
</tr>
<tr>
<td>A: The students are amazing and can jump right in.</td>
</tr>
</tbody>
</table>

The most important part of the summer workshop was the change in perception of computer science as a major and the host university as a choice for their students. After the workshop, there was a 50% increase in the number of teachers who would recommend CS, computational science, or IT to their students as a career and recommend that their students consider attending college at the host university.

CONCLUSION

The results from the one-week summer workshop are very encouraging. Working with local high school math and science teachers to increase their awareness and understanding of computer science and computational science is an effective way of reaching the educators teaching the next generation of computer scientists. Furthermore, by keeping these educational professionals apprised of how computer science and computational thinking infuses scientific and mathematic curriculums, future students will be better prepared for computer science, computational science, and information technology study at the university level. In a comparison between one-day in-service professional workshops for teachers-only offered earlier in the year [10], the longer (one-week) summer workshop offering changed the teacher perceptions to a greater extent. This was probably due in part to the greater period of time for the summer workshop, as well as providing teachers with the opportunity to work with their students on the concepts and curriculum materials presented. However, the absence of computer science teachers in the summer workshop may also be a factor in the greater change in perception by the summer workshop teacher attendees. Both the one-day and one-week workshops were effective in changing teacher perceptions.
SUMMARY

Overall, the local, non-residential teacher-student summer workshops have been very successful, with participants suggesting ideas for future summer workshops and being impressed with the wide range of scientific fields which would lend themselves to the computational techniques presented. A follow-up survey of both teachers and students is planned, to determine the lasting impact on both teachers and students as a result of their attendance at this summer workshop.

In addition to the significant increase in positive perceptions of computer science, computational thinking and information technology expressed by teachers attending the summer workshop, the student-led recruiting for the workshop, resulting from the use of a SAT email list, was one of the most interesting outcomes. Like many other innovations in computer science and information technology, a "bottom-up" approach to changing teacher perceptions may be demonstrated to be highly effective in developing the next generation of computer scientists.

REFERENCES


  www.csta.acm.org/Communications/sub/CSTAVoice_Files/csta_voice_01_2010.pdf

[8] www.computingportal.org/cs10k


AN INTERDISCIPLINARY APPROACH TO INJECTING
COMPUTER SCIENCE INTO THE K-12 CLASSROOM*

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

Brandon Milonovich
Department of Mathematics, Department of Teacher Education
The College of Saint Rose, 432 Western Avenue, Albany, New York 12203
milonovichb626@strose.edu

ABSTRACT
As society's use of technology expands, students are using computers and
technology at increasingly younger ages; however, enrollment and graduation
rates in college-level computer science programs are decreasing [2,8]. As a
result, the need for professionals in the field of software development is
rapidly outpacing the supply of graduates [9]. According to The National
Association of Colleges and Employers, computer science ranks as one of the
top five highest-paying career paths and is projected to further expand within
the next decade [6]. To bridge the gap between poor enrollment and
technology-related job force needs, computer science topics must be
introduced at both the elementary and secondary education levels. In general,
students are exiting the K-12 program with a lack of skills or even exposure
to computer science and other STEM (Science, Technology, Engineering, and
Mathematics) fields [3,4,5]. Introducing creative and fun interdisciplinary
study into K-12 curricula holds promise for bringing more young scholars to
the field of computer science. In this paper, we explore potential computer
science topics and how they can effectively be integrated into K-12 curricula
in an interdisciplinary fashion to support STEM initiatives, including efforts
to also increase enrollment numbers for women and underrepresented
minorities [3,4,5].

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

Enrollment in degree-bearing computer science programs is failing to meet the demand for computing professionals in the American workforce [9]. Overall, enrollment and graduation rates at the college level in computer science are decreasing [2]. High schools are reducing or eliminating computer science programs, as evidenced by the College Board's eradication of the Advanced Placement (AP) Computer Science AB exam this year [2,8]. Based on reports from the College Board and the Association of Computing Machinery (ACM), as noted in [8], the number of students taking the AP Computer Science exam has remained essentially constant since 1998. Other reports show a rapidly declining trend in the number of secondary schools offering AP courses or program in computer science (e.g. [4,5]).

In contrast to these disturbingly flat or declining trends in participation, AP exams in Calculus, Biology, Statistics, Chemistry, Physics, and Environmental Science have all shown steady increases since 1998 [8]. Regarding why computer science has seen such stagnant numbers, NSF program officer Janice E. Cuny observes in [8] that "high-quality computer-science instruction is all too rare in public schools."

The above trends are surprising, as computer science ranks third amongst the highest-paying career paths in the United States and open positions are projected to expand within the next decade [6]. A serious gap exists between education at the K-12 level and the available "pool" of computing professionals, which has led to a variety of recent STEM (Science, Technology, Engineering, and Mathematics) initiatives, including efforts to increase interest in computer science in women and underrepresented minorities [3,4,5].

We believe that these trends are due to how much society takes the use of technology for granted. Because our younger generations more readily and more naturally use technology, there is a feeling that minimal computer and technology instruction is required at the K-12 level. This is counterproductive, as evidenced by the aforementioned lack of growth in students taking the AP Computer Science exam and the decline in college-level computer science graduates. As further anecdotal evidence, we often hear of veteran teachers and administrators at the K-12 level lacking the knowledge or expertise in computer science to understand its importance in today's technological world.

We propose that increased exposure to computer science topics via interdisciplinary study at the K-12 level will encourage more young scholars to enter the field of computer science. Though establishing computer science as a field of study at the K-12 level may be a far-reaching goal, an interdisciplinary approach will hopefully enlarge the "pipeline" of incoming college students in computer science and other STEM-related fields. This paper presents a number of interesting and instructional topics for use in K-12 curricula in support of STEM initiatives. While these topics may not be new, what is new here is how they can be integrated in an interdisciplinary fashion, in some cases without students knowing that they are about to learn a core topic of computer science.
Interdisciplinary Computer Science in Physical Education: the "Binary Tango"

Computer science concepts can be integrated into physical education by combining the binary number system with a physical activity synchronized by rhythm. The goal is to illustrate counting using binary representation as it relates to standard 4/4 rhythms. Student participants wear custom-made binary T-shirts with a large "0" printed on the front and a large "1" printed on the back.

Capitalizing on the fact that 4 is a power of 2, we are able to relate each single quarter note beat to the least significant bit (2^0), every other beat to the 2^1 bit, each measure to the 2^2 bit, every other measure to the 2^3 bit, and so on. Participants stand side-by-side and are assigned a bit position. Upon hearing their rhythm cue, students turn 180 degrees, flipping their bit from 0 to 1, or vice versa.

This process is repeated until we have reached the maximum binary value for all bit positions (i.e. all 1s). And while this can correspond to music, it certainly does not have to. A snapshot of this fun exercise performed without the musical connection described above is captured in Figure 1 by eight of our undergraduates (with corresponding video available online at http://www.youtube.com/CSatSAINTROSE).

Interdisciplinary Computer Science in Art using Alice

The New York State Education Department (NYSED) Standards for the Arts provide school districts the basic structure for the development of local curricula that link instruction and assessment to the content standards [7]. In particular, the New York State Board of Regents outlines four learning standards. The first standard is "Creating, Performing and Participating in the Arts." For the visual arts, part of what constitutes meeting the standards for the elementary level is to experiment and create works of art in a variety of mediums, including computer graphics. At the higher learning levels, it is reflecting on the effectiveness of selected media to convey the artist's intended meaning.

The second standard is "Knowing and Using Arts Materials and Resources," in which it is suggested that students can use a drawing or paint program on the computer to create graphic images and use the computer as a design tool to communicate visual ideas. The third and fourth standards are "Responding to and Analyzing Works of Art" and "Understanding the Cultural Dimensions and Contributions of the Arts."
We propose that by using Alice [1], an introductory programming environment that allows students to create three-dimensional visual stories, teachers can teach fundamental computer science concepts while teaching art and also meet the required content standards. Developed at Carnegie Mellon, Alice is a free software program in which students can create visual stories or interactive games by dragging and dropping objects into a scene, then directing them and their behavior. Students can select from hundreds of graphical "objects" that can be rotated or moved in any direction.

Once the scene is developed, students create actions that the objects take; to do so, students drag and drop instructions that correspond to programming statements in such high-level object-oriented programming languages as Java, C++, and C#. Students using Alice are able to visually express their stories and ideas, view the work of others, and assess the effectiveness of computer graphic art form. Applicable at just about any grade level, Alice provides a gentle introduction to fundamental concepts of computer science. Further, teachers need little instruction or computer science expertise to use Alice; in other words, learning to use Alice is straightforward and intuitive.

Interdisciplinary Computer Science in Music

In a typical undergraduate data communications course, students learn a variety of methods for digitally encoding analog data. One of the most common examples of analog data, and one of profound interest to students, is music. In the past decade, CDs have been replaced by iPods and other portable MP3 players, yet many students are disappointed and puzzled by the fact that CDs often sound better than their digital MP3 counterparts. Seldom do students learn that the answer lies in the encoding techniques. For linear digital encoding, the digitization process approximates the height of the wave at regular sample intervals, as shown in the progression from Figure 2(a) to Figure 2(b). Fortunately, digital encoding is a straightforward concept, accessible at many grade levels. A simple explanation of the relationship between sounds and analog waveforms (e.g. frequency components tied to pitch, amplitudes tied to volume) could be comfortably introduced. An instructor can illustrate the difference between a course-grained (i.e. low-bit) sample size and a fine-grained (i.e. high-bit) sample size. Likewise, an instructor can demonstrate the effect of low versus high sample rates (i.e. samples taken per second).

In both scenarios, of course, an instructor can play the resulting encoded tracks, allowing students to hear the differences and relate them to the visual depictions shown in Figures 2(a) and 2(b). A variety of fun exercises can be used to further illustrate these concepts. At lower grades, a connect-the-dots game could be applied; at higher grades, students could determine, by ear, where the optimal point is between compression and sound quality.
Interdisciplinary Computer Science in Mathematics using Scratch

Applying the use of the computer to mathematics instruction is not new. We add to the evidence a case study at the Arbor Hill Elementary School within the Albany City School District in Albany, New York. In 2010, over the course of three lessons in two third-grade classes, Scratch [10] was integrated into the classroom, especially for mathematics instruction, by author Brandon Milonovich, a secondary education major with a focus on mathematics and computer science at The College of Saint Rose. According to the teachers involved, there is observable evidence that several students who were usually disengaged in their lessons found Scratch to be exciting and interesting—and not just on first use—thus providing a means to motivating these students to perform better in the classroom.

Through the program, students were exposed to 45 minutes of instruction in Scratch centered on integrating mathematics into sequential programming. The first lesson introduced students to the Scratch interface, while also allowing students to manipulate movement of a Sprite within an x-y coordinate plane. This not only engaged students in higher-order thinking skills, but also strengthened connections between what they have seen in their mathematics instruction and the computer games many are used to playing daily (or perhaps see older siblings playing).

The second week further increased the skill levels of students by introducing them to manipulations of effects on the Sprites. This brought a higher-level of abstract thinking to the number line concepts introduced with the coordinate plane in the first lesson. During the third week, students started to develop the beginnings of their first game, a word-counting game that provided students the means to develop sequential programming that interacts with a user and counts words. Figure 3 shows an example screen shot of the "programs" they developed.
In future lessons on Scratch, students will complete the game and other challenges, such as determining combinations of words to sum to various totals, thereby increasing their understanding of number sense. Students will also develop the project further by incorporating decision-making if-statements to verify answers given by players of the game and create interactive learning games that will continue to develop their understanding of mathematics and computer programming.

![Figure 3: A screen shot of computer programming using Scratch, as developed by third-graders](image)

Based on the promise and success of Scratch, we now offer a 1-credit course in Scratch for our elementary education majors at The College of Saint Rose, thus preparing them to offer the same benefits, as evidenced above, to their future students at all grade levels in mathematics, as well as other subjects.

**Interdisciplinary Computer Science in Social Studies**

From the ancient Mayan base-20 number system to the cryptic German Enigma code during World War II, to the rapid expansion of the Internet and its impact on modern cultures, computer science concepts have often played a pivotal role in history. Consider the current online "social network" revolution that has taken hold of us all. Unfortunately, many of these topics, often outside of an instructor's comfort zone, are glossed over in social studies, history, and civics curricula.

Covering the details of how the Enigma code worked may be beyond a reasonable scope; however, a light introduction to encryption may be accomplished though an explanation of the classic Caesar cipher, which closely resembles today's cryptogram puzzles often found in newspapers. Students may find it both educational and
entertaining to explore the puzzles on their own, practicing a technique that reinforces fundamental computer science material, logical thinking, and problem solving.

**Interdisciplinary Computer Science in English and Foreign Languages**

Students of English or foreign languages often learn specific rules of the language through the application of a grammar. The grammar specifies how sentences are constructed in the language, which directly parallels grammars for specifying computer programming languages. While spoken languages tend to have numerous exceptions and auxiliary rules, grammars used in computer science typically do not.

As students of a spoken language learn their grammar rules, an interesting topic to incorporate here is how grammars for programming languages also produce valid programming instructions or validate such instructions. Further, grammars for programming languages typically do not require expertise in any spoken language. Computer programmers may not speak the same spoken languages, but they are able to communicate via the same programming language and successfully build a large software system.

**CONCLUSIONS**

As noted above and in [8], there is a profound lack of educational resources deployed to support computer science within the K-12 curricula. Unfortunately, such an approach has led to a sharp decrease in enrollment and graduation rates of college-level Computer Science programs. In support of STEM initiatives already underway to help improve this trend, we have offered a number of interdisciplinary approaches to integrating Computer Science into a variety of subject areas within the K-12 curriculum, the key here being the interdisciplinary aspect of our approach. Nonetheless, this paper offers only the "tip of the iceberg" and will hopefully motivate others at all educational levels to increase student exposure to computer science. In addition to this, other future work includes organizing, cataloging, and publishing these topics in a more uniform manner for direct use by K-12 educators.

**REFERENCES**


METHODOLOGIES AND TOOLS FOR THE SOFTWARE QUALITY ASSURANCE COURSE

Vladimir V. Riabov
Department of Mathematics and Computer Science
Rivier College
Nashua, NH, 03060
603 897-8613
vriabov@rivier.edu

ABSTRACT
Tutorials, labs, projects, and homework assignments were designed to help students explore modern techniques of software quality assurance; debugging C/C++ and Java codes; and developing high-quality computer projects. Different methods (predicate-logic and topological approaches of graph theory; metric theory of algorithms, and object-oriented methodology of rapid prototyping) have been explored by using various tools in analyses of complex computing code. Applications cover software test strategies, code reusability issues, and ways to significantly reduce code errors and maintainability effort. The related course materials provide students with knowledge, instructions and hands-on experience, and motivate them in their research studies.

1 MOTIVATION
Students pursuing careers in software development should be familiar with various methods of software quality assurance (SQA). A year ago we launched a new SQA course that addresses the issue of quality throughout the software development process, including system analysis and design, rapid prototyping, implementation, testing, and delivery. Special attention is given to setting quality standards [1], developing quality measurement techniques, writing test plans, and testing the user interfaces. It becomes a challenge for an instructor to provide students with the state-of-the-art hands-on technology-exploration experience in this field.

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.
Topics covered include the integration of quality activities into the software project life cycle, CASE tools overview, structured testing methodology, complexity and object-oriented metrics, configuration management, capability maturity models, software engineering practices and standards, code re-engineering strategies, and miscellaneous topics.

Students were encouraged to examine various software-development projects. Exploring different testing strategies, they analyzed computer code by using various software-organization metrics including cyclomatic complexity [2], Halstead's [3] and object-oriented metrics [4], and re-designed the code with lower risk level and errors.

This paper contains an overview of the SQA software tools, tutorials, lab manuals, homework assignments, project reports, and research papers of students who took the Software Quality Assurance course. The advantages of using these tools for instruction in online and hybrid courses are also discussed.

2 TOOLS FOR SOFTWARE QUALITY ANALYSIS

2.1 Industrial SQA Tool Used in Software Engineering

One of the most popular SQA tools, the McCabe™ IQ software package, was selected for exploring various study cases and projects in the course on software quality assurance. This industrial tool became available for students under the free-license agreement with the McCabe™ University Program. The tool allows them to explore McCabe's structured testing methodology [2] that became a widely used method in code analyses, unit and integration test planning, and test-coverage estimations. Following specially designed computer-lab assignments and using the McCabe™ IQ tool, students study how to apply the theory of graphs for the complexity code analysis, develop test strategies, and predict possible errors [3, 6] in the code developed by themselves and companies. Unfortunately, the McCabe™ IQ package could be used only on campus; therefore, other Open Source free-license SQA packages were evaluated for use by students at home, specifically when taking courses online or in the hybrid format.

2.2 Open Source Free-License SQA Tools

There are several Open Source free-license SQA tools available for students. The Java Source Metric™ package [7] has been used to analyze Java source code with quality metrics like the Inheritance Depth, Lines of Code, and McCabe Complexity Metric suite. The CCCCTM tool [8] generates a report on various metrics (including the Lines of Code and McCabe's complexity) of C/C++ code. The freeware program SourceMonitor™ [9] has been used for code analysis to identify the relative complexity of code modules. SourceMonitor™ measures metrics for source code written in C++, C, C#, VB.NET, Java, Delphi, Visual Basic (VB6), and HTML. It operates within a standard Windows GUI and exports metrics to XML or CSV (comma-separated-value) files for further processing with other tools. The COCOMO-II™ tool [10] was used by students to estimate the cost, effort, and schedule associated with their software development projects.
3 LAB AND HOMEWORK ASSIGNMENTS

The main goal of labs and homework assignments is to introduce software quality metrics and help students build their individual skills of code analysis, testing, and redesign to improve code quality and enable possible reuse in other projects.

3.1 Introducing the Structural Testing Methodology

The first set of lab and homework assignments deals with implementation of the structured testing methodology offered by McCabe [2]. The approach is based on graph-theoretical complexity-measuring techniques in code studies and control of program complexity. Using the experimental results of Miller [5], McCabe suggests that code modules approach zero defects when the module cyclomatic complexity is less than 10. During lectures, the instructor provided an overview of the graph-based complexity metrics and the results of his systematic metric analyses of software for two industrial networking projects [6]. Following the lab assignments, students explored the McCabe™ IQ tool and used it to perform metric analyses of several codes by applying cyclomatic complexity (v), essential complexity (ev), module design complexity, system design complexity, and system integration complexity metrics [2] in order to understand the level of complexity of a code module's decision logic, the code's unstructured constructs, a module's design structure, and the amount of interaction between modules in a program, as well as to estimate the number of unit and integration tests necessary to guard against errors.

3.2 Estimating the Number of Code Errors and Efforts to Fix the Errors

The second group of the lab and homework assignments was designed to introduce students to the comparative analyses of algorithm implementations in different languages (FORTRAN, C, C++, Java and some others). Following Halstead's procedures [3], students identified all operators and operands, their frequencies, and estimated the program length, vocabulary size, volume, difficulty and program levels, the effort and time amounts to implement and understand the program, and the number of delivered bugs (possible errors), B. They compared their findings with values calculated by using SQA tools (McCabe™ IQ, Java Source Metric™, CCCC™, SourceMonitor™, and COCOMO-II™), and found that the results are sensitive to the programming language type (procedural or object-oriented).

In particular, students found that efforts to implement and understand the program were higher for procedural languages (FORTRAN and C) than for the object-oriented language (Java), even for simple algorithms, like Euclid's algorithm for calculating the Greatest Common Divisor. They also found that large C/C++ source files [6] contain more actual errors than the number of delivered bugs (B) suggested [3].

3.3 Interpreting Object-Oriented Metrics

The third group of the lab and homework assignments was developed to help students identify clusters of object-oriented metrics that would better describe the major
characteristics of object-oriented systems (properties of classes, polymorphism, encapsulation, inheritance, coupling, and cohesion) implemented in computer code written in C++ and Java. Their findings are summarized in Table 1 below.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Class Polymorphism</th>
<th>Encapsulation</th>
<th>Inheritance</th>
<th>Coupling</th>
<th>Cohesion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weighted Methods Per Class</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response For Class</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Percentage of Public Data</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accesses to Public Data</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of Cohesion of Methods</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of Children</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Number of Parents</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Depth in Inheritance Tree</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Coupling Between Objects</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Attribute Hiding Factor</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method Hiding Factor</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Polymorphism Factor</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1. Clusters of Object-Oriented Metrics

Students also identified some specific object-oriented metrics (Weighted Methods per Class, Response from a Class, Lack of Cohesion between Methods, and Coupling between Objects) that are the important factors for making a decision about the code module/class re-usability.

3.4 Comparing Two Releases of the Code

The last, fourth set of the lab and homework assignments was offered to students to identify major factors that forced programmers to change the code [6] in the project redesign efforts. After analysis of the project software (about 300,000 lines of C-code) using the network-protocol approach 271 modules of the old Code Release 1.2 [6] were recommended for redesign. The re-engineering efforts resulted in the deletion of 16 old modules and in the addition of 7 new modules for the new Code Release 1.3. Analyzing the deleted modules, students found that 7 deleted modules were unreliable ($v > 10$) and 6 deleted modules were unmaintainable ($e v > 4$). Also, 19% of the deleted code was both unreliable and unmaintainable. Moreover, all seven new modules are reliable and maintainable.

After redesign, code changes resulted in the reduction of the code cyclomatic complexity by 115 units. 70 old modules (41% of the code) were improved, and only 12 modules (about 7% of the code) deteriorated. This analysis demonstrates a robustness of the structured testing methodology and successful efforts in improving the quality of the Code Releases. Studying the relationship between software defect corrections [6] and cyclomatic complexity, students found a positive correlation between the numbers of possible errors, unreliable functions (with $v > 10$), and error submissions [6] from the
Code Releases (see Fig. 1) in the implementation efforts for six network protocols (BGP, FR, IP, ISIS, OSPF, and RIP).

![Diagram of Network Protocols](image)

**Figure 1:** Correlation between the Number of Error Submits, Number of Unreliable Functions \((v > 10)\), and the Number of Possible Errors for Six Network Protocols.

### 4 STUDENTS' PROJECTS AND RESEARCH STUDIES ON SQA

As the main assignment for the SQA course, students were asked to develop their individual or team projects on the quality analysis of moderate-size computer programs written in an object-oriented language (C++ or Java) and compare different releases of the code. The project assignment included the study of the code complexity and quality based on the analysis of cyclomatic complexity metrics, Halstead's metrics, and object-oriented design metrics by using available SQA tools (McCabe™ IQ, Java Source Metric™, CCCC™, SourceMonitor™, and COCOMO-II™). The project reports included the code structure chart, "Battlemap"; McCabe's Complexity Metrics summary; examples of flowgraphs with low, moderate, and high complexity; scatter diagrams with identification of unstructured-unreliable, reliable-structured, unreliable-structured, and reliable-unstructured modules; examples of flowgraphs with the independent paths for the unit tests; Halstead's Metrics report with estimation of the total number of the delivered errors \((B)\); Object-Oriented Metrics report with basic interpretation of the metrics; and recommendations to improve the code.

Michael Jeffords in his project, "Using SQA Metrics to Guide Refactoring of Medium to Large Scale Projects", developed a plan to reduce the overall complexity of the code while adding important functionality to the system that provided visualization of geophysical data. His code refactoring efforts took approximately 20 hours, but have reduced the number of tests by about 200 unit tests. Summarizing these efforts, he offered two effective methods of refactoring. In the first method, he used weighting factors to "boil" McCabe's and Halstead's metrics down using a formula \(M = \frac{v}{10} + \frac{ev}{4} + \frac{V}{3000}\). This approach gives equal weights to \(v\)-cyclomatic and \(ev\)-essential complexity metrics. Combined they have double the weight of Halstead's \(V\)-metric of the program volume \([3]\). In the second method, three thresholds for cyclomatic complexities were used to chart improvement over time. His goal was to achieve 100% of methods below the cyclomatic complexity of 20 and less than 97% of methods below the cyclomatic complexity of 10. He achieved 100% for \(v < 20\); 99.67% for \(v < 15\), and 98.01% for \(v < 10\). Michael described three stages of refactoring: 1) He started by refactoring classes to remove...
public variables and adding accessors; 2) Next, he chose any of the methods that ended up on the "hot" list that he could understand the basic methodology, and then tried to tighten up the algorithms; 3) Finally, he worked on the hardest methods once he felt that his plan had taken shape and he could see a direction for fixing the architecture of the worst classes.

In the other project, Timothy Houle and Douglas Selent analyzed the complexity of the Light-Up Puzzle program [11], its maintainability, testability and metrics related to the quality of the software program's design. The application's vulnerabilities were identified by using the McCabe™ IQ tool. This approach helped them to identify vulnerable code areas, reduce error rates, shorten testing cycles, improve maintainability, and maximize reusability. In order to verify the effectiveness of the McCabe™ IQ tool, they re-factored the program in the areas reported to be highly complex and error-prone. After this, they compared the McCabe's Metrics reports on the initial analysis to the reports on the re-factored analysis and charted results to clearly indicate the improvements made to decrease complexity. In addition to the McCabe metrics, the students also added various UML diagrams [1], which helped them understand the concept of the program structure and identify areas where object-oriented design principles could be applied to increase code reusability and maintainability.

5 EFFECTIVENESS OF THE COURSE AND STUDENTS' RESPONSE

All 16 computer-science graduate students that took the SQA course in the spring of 2010 expressed in their course evaluations full satisfaction with course organization, content, and material delivery. The overall course-evaluation score was high (4.71 out of maximum possible 5.0). In their anonymous comments, students shared mostly favorable observations, e.g., "This is a practically-oriented course and it has a lot of demand in the job market..."; "The instructor opened my eyes to the importance and usefulness of SQA metrics..."; "I learnt a lot of how the SQA is handled in the software development process..."; "Great experience...", and "I met my expectations more than I thought." Later these students effectively used the SQA techniques to improve their software programs in other courses, including the final capstone projects. All the students are currently employed by local computer companies.

6 CONCLUSIONS

The author has described the challenges and experience of running software quality assurance courses for undergraduate seniors and graduate students. The knowledge of graph theory and its applications in software engineering is beneficial for students with Computer Science majors. Detailed analysis of code complexity reveals areas of code structures that should be revised. The code revision allows students to find those code areas with potential errors and to improve code design and testing practices. In particular, the resulting analysis can be used in identifying error-prone software, measuring optimum testing efforts, predicting the effort required to maintain the code and break it into separate modules, reallocating possibly redundant code, and providing a fundamental basis for unit and integration testing. The complexity code analysis and structured testing
methodology should become a necessary attribute of software design, implementation, testing, sustaining, and re-engineering practice and training.

REFERENCES


A SOFTWARE ENGINEERING COURSE WITH A
LARGE-SCALE PROJECT AND DIVERSE ROLES FOR
STUDENTS*

Bonnie K. MacKellar
Division of Computer Science, Mathematics and Science
St. John's University
Queens, NY 11439
718-990-7452
mackellb@stjohns.edu

ABSTRACT
Industry software projects tend to be large enough and involve enough people that no one person can understand the entire project in detail. Teams are structured into groups that are responsible for different features of a product, with coordination between groups required. Typical industry software projects also involve people in many non-programming roles. Undergraduate software engineering courses, however, tend to be based on small projects that only require communication within each group of students rather than among groups. In this paper, we describe a software engineering course which uses a large-scale class project and diverse student roles to simulate a heterogeneous development environment.

INTRODUCTION
Software engineering is one of the core knowledge areas listed in the 2008 ACM Curriculum Update[1], which notes that software engineering is one of the topics that "receives attention in the dialog with industry". As a result, most computer science programs offer some type of software engineering course. Students often mention their software engineering projects in their resumes when job hunting and employers tend to be highly interested in what students have learned in this course. Our university is no
exception; software engineering is a required course in our computer science major. At our school, students enroll in the software engineering course with various backgrounds - some have done little programming and others know quite a bit. Many students have transferred in from community colleges. A number of our students do not aspire to careers as software developers and are more interested in testing, user interface design, or project configuration. These students want to learn about the software project management process at a fairly high level, and need to know more about non-programming roles - a topic that is often given short shrift in computer science programs. On the other hand, some students have very strong programming skills and may be working at software jobs in industry while taking the course. These students are interested in a software engineering course that is focused on program design and development. Our core problem was to meet these diverse needs while also realistically simulating an industry software development project.

Problems, however, often lead to opportunities. We observed that at many companies, development teams are formed into cross-functional groups based on role - for example, a QA group, a technical writing group, or a user interface group. Thus, we organized this course to mirror a role-based organization. Some students were placed on a QA team, others on a user interface standards team, and others on subproject development teams. In order to accomplish their work, students had to communicate not just within their teams but with other teams as well. Once we had decided to include non-developer roles in the project structure, we realized that the best way to do this was with one large project that involved the entire class.

Traditionally, software engineering courses are based on standalone projects suitable for smaller teams of 4 to 5 students. A literature survey [11] on project group characteristics stated as one of its findings that 4 to 5 participants is most optimal from a course management perspective. Many other papers describe small project teams in software engineering, including [7], [6], and [9]. A new alt[2] describes a course structure in which students rotate between groups, but the groups are still small (3 to 4 students) and are not interdependent in any way. In typical SE courses, students usually have homogenous roles in the project - all work on requirements together, then design, and then coding. Communication occurs between individuals in each group rather than among groups.

This is not typical in real-world software development, where unless one is working in a very small startup company, people tend to interact with a larger number of people who have heterogeneous backgrounds and roles. According to Bruegge and Dutoit, "The development of software brings together participants from different backgrounds, such as domain experts, analysts, designers, programmers, managers, technical writers, graphic designers, and users. No single participant can understand or control all aspects of the system under development…". [3]. It is precisely this experience that is hardest to duplicate in a typical software engineering course. Often students come away saying they could have gotten the project done more quickly if they didn't have to work in groups. This happens because the type of software project that can be finished by 4 students in a semester is small enough to be understood by one person; hence the feeling that the overhead of working in a group has merely made the task more difficult.
In many software companies, developers may be divided into smaller groups based on components or project features, but these smaller groups are interdependent. A decision made in one group about a data representation, for example, may well impact what other groups can do. It is also very common to have non-development groups, such as documentation specialists or testing specialists, organized into their own groups, who then interact with all of the feature-oriented teams. It is especially common for the QA engineers to be organized into a separate group, in order to ensure independence from the developers. This type of organization, with its need for cross-group coordination, is the way in which we organize our software engineering course.

There have been a few large scale software engineering course projects described in the literature. In a 1991 paper surveying approaches to the software engineering course, Tomayko and Shaw[13] strongly endorsed the use of whole-class projects. However, this did not seem to catch on in the wider community, since only a handful of reports on this type of project have appeared in the literature since that paper appeared. Nita-Rotaru, Dark and Popescu [8] describe a course that integrates students of varying backgrounds into one large development team; however, this course involves graduate students as well as upperclassmen at a large engineering school - a very different demographic from our students. In addition, the group organization described in that paper is quite different from ours. Their organization is solely in terms of features, and seems more typical of a large research effort than a commercial software development effort. Rebelsky and Flynt [10] describe a whole class project in which students work in groups focused on project subcomponents; this course did not include non-developer roles such as testing or user interface design. Tan and Philips[12] describe a course setup that allows for a number of different development roles; however, the project teams are small (5 students per team). Finally, Coppit and Haddox-Schatz [5] describe a whole-class project in which task allocation and grading is tightly tied to the issue tracking system. This is an interesting approach but it seems very complex and may be difficult for students to manage.

In the course reported in this paper, students are assigned to groups and tasks by the instructor who functions as a project manager. The students submit resumes as their first assignment, and these resumes are used to allocate students to groups based on background, skills, and interests. Just as in industry, the main criterion for assigning students to groups is the likelihood of project success. Since the aim of this course is to teach students how to function in a real-world development environment rather than to teach them any specific programming skills, we feel this approach to group formation is justified. The students are informed of the criteria as they write their resumes. In addition, we use this exercise as an opportunity to discuss resume-writing skills.

**COURSE STRUCTURE**

The first semester, the course was one semester long and met 3 times a week for an hour each session. Currently, the course is being taught in a once-a-week, 3 hour format. The enrollment for the course is capped at 25 students. The class time is structured so that some time is spent in traditional lecture, presenting topics such as project management, UML, quality assurance, and requirements gathering. Time is also spent in hands-on lab activities, learning to use development systems such as Bugzilla and Subversion, as well
as some of the specific technologies that will be needed for project development. Students enrolled in this class had, for the most part, never used a version control system or a bug tracking system before.

The class project was the development of a website for a weekend children's Chinese school. This project exemplified a typical web-oriented software project - a web browser as front end, a middle layer of "business logic", and a database backend. Depending on the component, students used Java, C++, JavaServerFaces, SQL, HTML, and CSS. The project was divided into a number of subprojects: the student and teacher management module, a set of teaching applications including pinyin and traditional character flashcards and a dictionary, a parent volunteer scheduling application, and a set of database reports. A small student team was assigned to each of these development projects, based on their interests and backgrounds as stated on their resumes. In addition, some students were assigned to the following cross functional groups: QA, user interface standards, technical writing, and build engineering.

Since students now no longer had uniform roles and tasks in the course, a different set of deliverables and grading policies needed to be created for each group. All students participated in the initial writing of the requirements document. Once the requirements document was created, however, tasks and schedules began to diverge. The teams working on development subprojects followed a traditional software engineering course timeline of first submitting a detailed design, then submitting milestones as well as weekly status reports. Once QA testing commenced, the weekly status reports had to document responses to bug tickets as well. The QA team, which had three members, was responsible for creating test plans from the requirements document, executing the tests, tracking results and submitting tickets in Bugzilla, and then working with the subproject teams who were responsible for tickets. Their final report consisted of all test plans, showing the status of tests. The user interface group was responsible for designing an overall "look and feel" for the website, documenting user interface standards, helping the other groups achieve the user interface standards, and coding the Cascading Style Sheets and the top level HTML page that tied together all of the subprojects. The user interface group was responsible for a user interface report which documented the look and feel, and justified it in terms of user interface standards. The technical writing group built an online help system. They initially worked from the requirements and detailed design documents created by the class to create their first milestone effort. Then they modified their initial help system to reflect what the developers had actually created. The QA group included the online help system in their test plans, so the user interface group had to respond to bug tickets just like the other subproject groups. The role of the build engineering group was to create an automated build and deploy the system onto a Linux server. Unfortunately, they did not get as far as the other groups, largely because of the complexity of the build requirements (some subproject groups brought in third party software which greatly complicated the build) and lack of knowledge about application server environments.
GROUP COMMUNICATION

One of the key goals of the course was to create a project large enough that students would have to communicate and coordinate between groups as well as between individuals. With this project, inter-group coordination fell into two categories:

- **Software coordination:** The groups had to agree on certain common software functionality as well as database structure, and coordinate changes to the software and database as they came up. Coordination between the groups to achieve a consistent look and feel was also necessary; this was the responsibility of the user interface group.

- **Task coordination:** The groups also had to coordinate in terms of tasks. For example, the QA team had to communicate bugs to the relevant development teams, and the students responsible for fixing bugs had to communicate status back to the QA team. The user interface group had to communicate interface standards to the development teams, and work closely with each group in order to achieve a consistent interface.

A number of communication mechanisms were used to coordinate between the groups, such as email and discussion boards. One communication mechanism which was very successful was the use of in-class presentations. Each group had to do an in-class presentation at about the time that serious development was starting, to introduce the group and their role. The discussions at these presentations were very animated as each group suddenly realized their interdependency with other groups.

We also used class time to coordinate between groups. Since almost all students in the course are commuting students with outside jobs, we recognized that it would be very difficult for these students to regularly meet outside of class. As it turned out, this time was frequently used for inter-group coordination, often because students had trouble identifying which member of a different group to communicate with when in a non face-to-face setting, or the best means of coordinating with that person. This difficulty in identifying information sources in other groups was interesting and fits with work done with professional software teams [4].

ASSESSMENT

In order to measure student perspectives on what they had learned in the course, a 2 part survey was developed and administered at the end of the semester. We will use this survey again at the end of the current semester. The first part of the survey consisted of 7 questions measuring their beliefs about how well they had learned and understood various aspects of the course. The survey scale was 5=strongly agree, 4=agree, 3=neutral, 2=disagree, 1=strongly disagree. Table 1 shows the results of the first part of the survey.

Interestingly, the question asking if the student is likely to pursue a career in software development had the most "Strongly disagree"/"disagree" results - a total of 4. In addition, 3 were neutral, meaning that 7 of the 17 students in this software engineering class were not aiming towards a career in software engineering. From conversations with the students, it was clear that some of the students came into the course with plans to pursue a non-development career. It would be interesting to measure this attitude at the
beginning of the course as well as at the end, to see if there are any changes as a result of the course.

Table 1. Results of student survey

<table>
<thead>
<tr>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Mean response</th>
</tr>
</thead>
<tbody>
<tr>
<td>I feel I understand what a professional software developer does on the job.</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>9</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>I am comfortable that I could participate in the planning and development of a real world software project</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>3.59</td>
</tr>
<tr>
<td>I understand the role of testing in a software project.</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>4</td>
<td>4.05</td>
</tr>
<tr>
<td>I feel that I know how to use the kinds of tools that software engineers use on the job.</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>12</td>
<td>0</td>
<td>3.70</td>
</tr>
<tr>
<td>I am likely to pursue a career in software development</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>3.65</td>
</tr>
<tr>
<td>I feel more comfortable now with the idea of getting a big job done in a large team.</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>4.00</td>
</tr>
<tr>
<td>The tools available to the class helped me get my job done.</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>3.88</td>
</tr>
</tbody>
</table>

The second part of the survey listed 10 class topics and asked students to list the ones they had learned the best, the ones that were the most difficult, the ones that they felt were most important to their future career, and the ones that they felt were least important to their future career. The questions were also more open ended and allowed them to insert comments. A majority (10) of the students felt that the topic they learned best was requirements writing, even though only a couple of weeks were spent on this topic in the course. This is probably because the textbook had a lot of coverage of the topic, and also because it was the least technical topic. Seven students believed they had best learned group communications, and also that this was a topic that is very important to their careers. The fact that communication within and among groups was such a key component of the course probably led to that result. And not surprisingly, students chose the most technical topics as the ones they had the most difficulty with - web programming and automated tools. Because of the lack of strong prerequisites, many students were very weak in their technical preparation.
DISCUSSION AND CONCLUSION

The goals for this course were to realistically simulate industry software development by using a project big enough to require students to be divided into interdependent groups and to productively integrate students with varying backgrounds and skills into the course. Students responded well to the course organization and were active and engaged throughout the semester. Students with little programming background were able to make important contributions by developing the online help system or designing and carrying out test plans. These roles were far from being a backwater and forced the students to interact extensively with every other group, following the product through all of its changes. Students with experience developing web-based interfaces were able to use those skills in the user interface group, and students with skills in database development could contribute that knowledge. The result was that students came away with a better understanding of the different interdependent roles within a software product group. The course has been offered a second time this past fall, and is currently being offered again this spring. We made some adjustments, using the experience of the first semester. We used Swing instead of a web-based project to simplify deployment details. We provided more explicit guidance to the students in recognizing risk factors in their projects or roles early on. And finally, we are tracking communication patterns among the students closely, with plans for future research into modeling these communication patterns.

In this paper we describe a problem - lack of uniform background and skills among students taking a software engineering course - that was turned into an opportunity to make the course better reflect the way that software is produced in many companies. Since real world software development makes use of a relatively large number of people, all with differing backgrounds, talents, and roles in the project, we structured the course to reflect that. The project was large enough to require different specialized roles also involved students in a situation where each individual development team impacted the work of other development teams. As a result, students learned the importance of communication and coordination among groups, and the varying roles that project team members may play in a software project.

REFERENCES


SOURCE CODE ANALYSIS AND ATTACK TECHNIQUES AS EDUCATIONAL APPROACHES TO TEACH SECURE PROGRAMMING

W. Scott Harrison and Nadine Hanebutte
Department of Mathematical and Computing Sciences
St John Fisher College
3690 East Ave.
Rochester, NY, 14618
585-385-8000
sharrison@sjfc.edu

ABSTRACT
We believe that a sound knowledge of security is necessary for students, and in this paper, we intend to justify this statement and show how this can be accomplished using a focused set of two courses. We will describe the layout of the courses in such a way that it would be possible for an instructor to implement our model. Although the paper makes references to different types of security issues, such as race conditions or buffer overruns, it is not necessary to have technical knowledge of what these terms mean. Space limitations restrict our ability to explain these terms; however, we have included citations to explanatory materials.

INTRODUCTION
The need for computer security is not debatable. Daily we hear about lost databanks of social security numbers, private records, as well as nation-states conducting cyber-warfare on their enemies, as happened between Russia and Georgia in 2008 [1].

However, this does not mean that all computer science students will need to be security experts. Certainly there is a need for these types of graduates; however, there is a need in all fields of computer science. We argue, then, that even if a student does not

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.
intend to go into the area of computer security, a knowledge of the field is a benefit. In fact, in its commentary to the 2008 Computer Science curriculum update, the ACM suggests the security should be added into computer science core classes, in particular, programming classes [2]. Graduates in computer science will, logically, be programmers, managers, software engineers, and the like. All of these groups of people will benefit from knowledge of basic computer security issues.

It is critical, we feel, that students understand the impact of the choices they make when writing code or making design choices. We have argued in the past that poor programming leads to security issues [3]. A buffer overrun, which involves copying data into a too-small allocated memory space, can cause an attacker to take complete control of the program, and, possibly spawn a shell giving the attacker access to user or administrative rights (an excellent, if somewhat dated, discussion of this can be found in [4]). We have done a survey of introductory programming texts, and discovered that most, if not all of them, contain serious security flaws in their own examples and explanations [5]. This leads students to writing software containing security flaws, which of course can lead to serious repercussions later.

Students should be keenly aware of what actually happens when they do, for example, a strncpy as opposed to strcpy (the former can restrict the size of data to be copied, while the latter does not, in C). The question, then, is how do we make them understand this? We can tell them, but simply stating that it should not be done leaves something to be desired. The next logical question is why? And instead of simply answering the question, we reverse it on the students; we ask them why. The method by which we do this is through a series of two in-depth courses aimed first at identifying security flaws, and then actually using these security flaws as an attacker would; seeing exactly, first-hand, what can happen when code is poorly written.

In the next two sections, we will give an outline of both courses. This is intended to serve as a model for those who may be interested in implementing a similar system. Finally, the last section will conclude with commentary about impressions and effectiveness of such a series of courses on students.

SECURE SOFTWARE DEVELOPMENT

This course, offered at the junior level focuses on identification (and subsequent correction) of potentially vulnerable code in software, as it is being developed. It is conducted in C on a UNIX operating system. The course addresses potential areas in which security flaws can be introduced in any stage of the software engineering lifecycle. It is a traditional lecture-type class supplemented with homework and quizzes.

The course begins with a strong emphasis on the motivations of such a course; why the students are being expected to think about potential security flaws in code, and what questions they should be asking themselves during the software engineering process. The lectures then proceed to address secure coding principles, such as redundant levels of defense, graceful (and secure) failure, and least privilege. Following this, topics such as compartmentalization, avoidance of overly complex code and high coupling, privacy issues (including, for example, clearing a buffer of data before reusing it and encryption),
and finally, that input to a program should not be trusted, regardless of the source (including user input as well as, for example, network packets).

After this overview, specific types of errors and avoidance are discussed. The first error discussed is a buffer overrun; and specifically, we discuss safe versus unsafe C functions, as well as how shellcode can be introduced (shellcode is the ability to implant a command line interface into faulty code; it is a popular exploit, and is discussed in detail in [5]). After a discussion of what a buffer overrun is, the material then shifts into how to avoid them. Following on this, format string exploits (where the arguments to printf() are incorrect, which can allow attackers to take control of a program, discussed in [4]) are covered, however, this is covered more lightly, as an in-depth discussion takes place in the next course.

The next topic expands on knowledge gained from our Operating Systems course, and describes access control models in both Windows XP and UNIX. The importance of topics such as understanding how setuid and setgid modifies ownership and what this means to system security is discussed. Many examples are given to illustrate the complexity of how these bits can change, depending on the operation.

Following this, we discuss race conditions. While this is also discussed normally in Operating Systems, in this case, we look at how not understanding a race condition exists can result in security failures of code (a number of exploits are actually based on race conditions) For example, we discuss what can happen if temporary files are not properly protected, as well as what should and should not be in a temporary file. We also emphasize secure file access in this set of lectures. Finally, we discuss various auditing tools and their strengths and weaknesses.

By the end of this course, we have introduced a number of important security faults, and, using examples with code, have shown how to identify and mitigate them. However, this only fulfills part of our objective; in addition to knowing what to avoid, we also wish to demonstrate what can happen if these are not avoided. This is the goal of the next course, Applied Security Concepts, which we now discuss.

**APPLIED SECURITY CONCEPTS**

The purpose of the Applied Security Concepts course, then, is exactly as the name implies. However, instead of merely identifying vulnerabilities and repairing them, as was done in Secure Software Development, in this course, the focus is more on the mind of an attacker; specifically, what sorts of things do attackers do, how do they do it, and how can it be defended against? This course introduces the students to real-world attacks, they perform buffer overrun as well as format string attacks, insert shellcode, and analyze real-world viruses and denial-of-service tools. The course is taught in C, and, as with Secure Software Development, the prerequisites enforce this. We also require senior standing to take this course; a certain amount of maturity is desired before students start writing and examining real attacks.

One advantage of teaching this course at a liberal arts college is that the students have typically taken ethics courses before taking this course, but we discuss ethical issues in this course as well. We would also like to take this opportunity to explain why we feel this is such an important course. While it is certainly possible to learn about
vulnerabilities and attack techniques in theoretical form without examining any real source code (as some suggest), this, we feel, hampers the students in a number of ways.

In general, the problem with not showing, in a laboratory setting, how attacks actually work is twofold. The old adage of "Tell me and I will forget, show me and I may remember, but involve me and I will understand" is the mantra of this course. Up to this point, we have told and shown students, but not involved them. The reason we feel it is so important for students to actually look at attacks and write buffer overruns themselves is to give a feel for how difficult it is to take advantage of a vulnerability and break into a system. Keeping track of all the information needed to accomplish a successful attack is daunting. For example, in a format string attack where shellcode is inserted, one needs to keep track of the current stack pointer address, know the top of the stack is, know the offsets of promising addresses, know the address of the shellcode, know the byte-order of the architecture, and finally, monitor how a %n directive changes as the format string is built. This is a lot of information to juggle at once. While students can certainly memorize "what do I need to know and how do I figure it out" (and, of course, some of this is necessary and enforced by quizzes), it's much more meaningful to stand in front of the classroom and say "O.K., we want to modify the value of X. What do we need to do to accomplish this?" At this point, we're looking at actual code and trying to exploit vulnerabilities within it, as an interactive session.

The other issue with not demonstrating how real attacks work is a secondary result: the students learn to respect the people that they might be defending themselves against. Once they understand the complexity of the task of exploiting software in a useful way, the course starts to examine real-world Denial-of-Service tools, virii, worms, and so forth. The students are required to analyze the source code and explain how a particular attack works. And without fail, after this exercise, they are amazed at the skill of those who write such attacks. They realize that these are not amateurs, but instead, highly skilled programmers that should not be lightly dismissed. They learn that a single line of poorly written code can compromise a computer, via examples where this exact situation has happened.

It should be pointed out at this time that this course obviously needs a special classroom laboratory. No one could benefit from our college accidentally releasing viruses into the world. To this end, we have secured a designated laboratory for this course. This classroom consists of a number of reclaimed PCs that are connected via an air gapped network.

As a side note, the machines are all installed, configured, and maintained by the students in the course, as well. When, for example, we get a new set of PCs (donated to us instead of being destroyed during regular upgrades), the students are required to set them up in our laboratory configuration (specifically, dual-boot XP and Linux). Students are usually very eager to work on this project, as they get little opportunity to do this type of network configuration and setup on such a large scale and level of detail. It also gives them a nice set of hands-on skills. The machines all have two network cards so that we can reconfigure the system as a gateway, attacker, or victim. We have two routers, one devoted to connecting the Windows side of the machines, and the other connecting the Linux side. We can, of course, easily interconnect as we wish, but this is our standard configuration. We had students configure the router and determine the network
connections to be made. All of this work gives them a sense of connectedness with the lab; it isn't simply something they step into, but something they are an integral part of. Further, it gives them a sense of satisfaction to know that the efforts they are making will be used by future classes.

As for the course's content, we start with a focused refresher on C. Pointers tend to be problematic for students and this course requires absolute mastery of them, for most of what is done is manipulating memory. To that end, we spend the first two weeks or so drilling the students on all aspects of pointers, including homework and quizzes to ensure their knowledge. After this, we move into actual methods of exploitation, beginning with buffer overruns (This is not a remote or isolated problem: in 2008, for example, buffer overruns inside a game allowed wii users to boot the device from an SD card, which previously had been impossible without modifying the hardware [5].) Our students learned what a buffer overrun is in Secure Software Development and how to avoid them, now they see what harmful impact this type of code error can have. The approach is along the lines of showing the students some odd-looking errors, and speculating about what they are seeing. Eventually, they are able to understand how variables can be changed as well as return addresses and how shell code can be inserted via a simple buffer overrun.

The course then moves into an examination of format string attacks. While not as common as buffer overruns, these are interesting attacks to examine because they allow complete manipulation of the source code (whereas a buffer overrun tends to simply destroy stack data, a format string attack allows selective manipulation of the stack). The technique of instruction is similar; the students examine what first appears to be random-looking output, and eventually figure out that none of it is actually random but instead reveals a great deal of information about the program. This period of instruction culminates in the students attempting to gain privilege escalation by exploiting a piece of software written specifically for this purpose and this course.

The class then moves into a second phase, which involves examining a number of real-world exploits which use exactly the techniques that the students have learned. We examine the attacks from several angles: history (what the attack did), mechanism (how it works, including demonstrations), and defenses (how it can be detected and/or stopped). The course then ends with the students themselves presenting the same type of information within groups of two or three students. During this period, they have complete access to the laboratory, and are free to experiment with different attacks and defense methodologies on their own.

CONCLUSION

We feel that, with the continued expanding reliance on computing technology, it is of paramount importance that not only do Computer Science students learn to program, but learn to program correctly. As technology gets more pervasive, it is important to realize that not only does a program have to work correctly, it has to resist attempts to exploit it.

To that end, we feel every Computer Science student needs a background on basic computer security. Although it is the case that not all of our graduates go into security
fields, it is the case that all of our graduates will need to have this background knowledge as the demand for programmers with these skills increases.

We have accomplished this at our institution by creating two new courses, which we have described in some detail. It should be pointed out that it does not take an expert in security to offer such a course, the references we have given, as well as some additional outside reading, can allow any Computer Science instructor to teach such a course. At our institution, these courses are now part of the required coursework. However, we initiated these courses as electives, to get a feel for the difficulty and to solicit student responses.

Our students enjoyed the courses and have benefitted from them. It has happened more than once that students (and former students) drop by and tell us how important this knowledge is with respect to their jobs. While the introduction of the courses was a great success, making them a required part of the curriculum has greatly benefitted our students.

REFERENCES


LEARN HOW TO USE EXECUTABLE FLOWCHARTS TO
ENHANCE LEARNING IN GENERAL EDUCATION, CS0, and
CS1 COURSES*

TUTORIAL PRESENTATION

Dee Gudmundsen
The College of Saint Rose
Albany, NY 12203
518-454-2105
gudmundd@strose.edu

Lisa Olivieri
Chestnut Hill College
Philadelphia, PA 19118
215-248-7092
lolivier@chc.edu

Namita Sarawagi
Rhode Island College
Providence, RI 02908
401-456-9865
nsarawagi@ric.edu

Visual Logic © (www.visuallogic.org) is an interactive, graphical software tool that enables a novice programmer to develop interactive, executable flowcharts. Visual Logic is extremely user-friendly, requires minimal syntax, and is very simple for a student to learn. In this tutorial, we will present the ease, power and versatility of using Visual Logic to teach programming concepts in three different courses at three different colleges. The first course is a General Education CS course introducing algorithmic thinking. The second is a CS0 course which uses Visual Logic for six weeks and then transitions to Python. The third course is a CS1 course which uses Visual Logic as a tool to teach programming concepts and to brainstorm solutions. Visual Logic is not taught separately, but integrated with the various Java control structures.

The goal of this tutorial is to highlight the ways Visual Logic can be used to improve student understanding of programming concepts and to enable the participants to easily adopt Visual Logic in introductory courses. Participants will explore and experiment with Visual Logic first-hand through supplied exercises. The presenters will provide deliverables, such as sample programming projects and comparisons of Visual Logic syntax to Python and Java syntax.

TUTORIAL OUTLINE

1. (10 minutes) Introduction to Visual Logic and the rationale for using it.
2. (20 minutes) Brief overview of how Visual Logic is used:

* Copyright is held by the author/owner.
in a General Education course with a varied student clientele. Students taking this course lack prior programming experience. Visual Logic is the appropriate programming language tool for these students to develop their algorithms and test their logic due to its user-friendliness and (almost) syntax-free nature.

in a CS0 course that teaches programming concepts using Visual Logic during the first half of the course and transitions to Python. Similarities between Visual Logic and Python such as lack of variable declarations, use of pre-defined functions, and the interpreted nature of Python and Visual Logic will be examined.

in an objects-late CS1 (Java) course that uses Visual Logic as a tool to illustrate concepts and brainstorm solutions. Examples of lecture notes with Visual Logic-created flowcharts and Visual Logic demonstrations of concepts before introducing Java syntax will illustrate how the visual approach helps students grasp concepts and reduces the frustration associated with syntax errors. The presenter will also show the successful strategy of having students incrementally develop a solution using the Visual Logic tool and then code the solution in Java by associating standard building blocks of Java code with flowchart steps.

3. (45 minutes) Hands-on exercises illustrating major features of Visual Logic.

DELIVERABLES

Presenters will make the following available:

- A list of exercises from various disciplines correlated with programming concepts in a General Education course.
- A textbook appendix covering traditional flowcharting symbols and their meaning.
- Comparison of Visual Logic Syntax to Java Syntax.
- Materials that support student transition from Visual Logic to Python.
- Sample Visual Logic programs illustrating major concepts.

REQUIREMENTS

This tutorial will be offered in a computer classroom with Windows-based computers and teacher station with large-screen projection capabilities. Participants should bring a USB flash memory drive for installation of (non-registry- altering) software.

BIOGRAPHICAL INFORMATION ABOUT PRESENTERS

Dee Gudmundsen is an Assistant Professor of Computer Science. Dee has been teaching CS1, using an objects-late approach, at The College of Saint Rose for 10 years. Since Fall 2009, Dee has been using Visual Logic in CS1 to introduce programming concepts before introducing Java syntax and as a tool to brainstorm solutions to programming problems.

Lisa M. Olivieri, SSJ, PhD is an Associate Professor of Computer Science and Technology and chair of the Computer Science and Information Technology Department.
Having taught the introductory programming course for sixteen years, she has used Visual Logic in conjunction with C++ or Python to teach this course for the past four years.

**Namita Sarawagi** is an Assistant Professor in the Mathematics and Computer Science Department. She has taught introductory programming courses for several years. She was involved in the development of a new CS General Education course: "Introduction to Algorithmic Thinking". She has presented a poster at CCSC-NE 2010 illustrating the effectiveness of this tool in a course targeted for students of all majors.
STUDENT INTEREST AND CHOICE IN PROGRAMMING ASSIGNMENTS

Lisa Torrey  
Computer Science Department  
St. Lawrence University  
Canton, NY 13617  
315 229-5446  
ltorrey@stlawu.edu

ABSTRACT
This paper describes a study, conducted in an introductory programming course, on the factors that make students interested in programming assignments. These factors include whether the assignment is perceived as easy or difficult, and the paper analyzes these in detail. There are also significant factors involving the end product of the assignment. The study also looks at the impact of these interest factors on choice, when students must actually choose a program to write. The same factors are involved, but some become more important while others become correspondingly less important. These observations may be useful for instructors considering the design of assignments or the role of student choice in introductory programming courses.

INTRODUCTION
Introductory programming courses typically see a wide range of student aptitude. Faced with this diversity, computer science educators may struggle to design programming exercises. Easier assignments are trivial for high-aptitude students, but harder assignments may not be effective learning experiences for low-aptitude students.

Ideally, each student would perform exercises at an individually appropriate level, working up to more challenging levels over time. It is interesting to note that this is exactly what modern video games get players to do. Education researcher James Gee
suggests that these games encourage learning because they "operate at the outer edge of a player's growing competence" [4].

Perhaps we could create a similar learning environment by designing exercises with a range of difficulty and encouraging students to choose the ones that interest them. If challenges just at the edge of our comfort zone are the most interesting, this framework could be very effective. However, it is worth asking whether student interest in programs really would follow this pattern.

I have conducted a small study on this topic at my undergraduate liberal arts institution. This paper presents the results of that study, and attempts to answer the following questions:

- Are my introductory programming students truly most interested in mild challenges?
- Other than the level of challenge, what other factors affect my students' interest in programs?
- If I did give my students a choice of programs, what factors would most affect their choice?

BACKGROUND

The impact of interest on learning is well established in psychological research. Studies in this area most frequently focus on reading comprehension as the learning task, but their findings may apply more generally. In one early example, Hidi [6] found that interest in subject matter is correlated with motivation, and leads to higher attention, persistence, and memory. Ainley et al. [1] suggest an explanation: interest increases the emotional value of the reading, which encourages persistence. Edelson and Joseph [3] also found that interest leads to increased effort and a higher mastery of skills.

One study on mathematical word problems may be more directly applicable to the programming setting. Renninger et al. [7] found that interest in problem topics leads to higher focus and tolerance for frustration. However, they also had a cautionary finding: familiarity with a topic can lead to overconfidence in a problem solution.

Some educational research has also addressed the factors that promote student interest in learning activities. Tobias [8] argues that prior knowledge about the topic of a learning task is an important factor. Edelson and Joseph [3] suggest that the perceived usefulness of an activity is also relevant.

There are a few studies on student interest specifically in the programming setting. Hansen and Eddy [5] had CS2 students rank assignments by level of engagement and frustration, and found that the two were correlated. They proposed a measure that they call *niftiness*, which increases with engagement and decreases with frustration.

Cliburn and Miller [2] allowed their students to choose between three types of assignments: traditional, game-based, and story-based. A large majority chose the game assignments because they found games interesting. Story assignments were the least popular, but only because students found them to be too open-ended.
DATA COLLECTION

To answer the questions posed above, I conducted survey activities with students in my introductory programming course. Participation was voluntary and earned students a small amount of extra credit in the course. I conducted three different surveys, spread evenly throughout the semester, to increase the opportunities for participation.

I was able to collect data from 14 students, exactly half of my class of 28. Of these, 8 participated once, 3 participated twice, and 3 participated all three times. Based on their final grades, these students represented a good cross-section of the course: one ranked first in the class, one ranked 22nd, and the rest fell in between.

The surveys each presented a list of 7 program descriptions. Each description specified a program to be written in Python, using concepts recently covered in the course. Table 1 shows some examples. All of the programs were somewhat short, to keep the activity brief, since it was being conducted outside of class on a volunteer basis. None of the programs were games, since preferences for those are already known [2], but some did involve graphical displays. Graphics programs used a course library, a simple wrapper for Tkinter that allowed drawing, animation, and mouse interaction.

Given this list of programs, each survey asked students to do the following:
• Give each program an interest rating: boring, neutral, or interesting.
• Give each program a difficulty rating: easy, moderate, or hard.
• Choose one program to write.

The surveys asked students to provide written explanations for their ratings and their program choice. These questions were entirely open-ended, so that students were free to mention any relevant factor.

Table 1: A few examples of program descriptions from surveys.

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write a program that inputs people's weights and prints whether the total weight entered is safe for a 1000-pound capacity elevator.</td>
</tr>
<tr>
<td>Write a program that inputs a number and prints whether it is a power of two. A power of two can be divided by two repeatedly, and the result is always an even number, until it hits 1.</td>
</tr>
<tr>
<td>Write a program that draws animated circles moving randomly across the screen.</td>
</tr>
</tbody>
</table>

DATA ANALYSIS

Student responses explaining their interest ratings contained several strong themes. I extracted these six factors from their responses:
• The program involves graphics.
• The program has a useful practical application.
• The program is entertaining to use.
• The program connects to my major or hobby.
• The program looks challenging to write.
• The program looks easy to write.
Several of these factors focus on the end product of the assignment. Many students found programs that involved graphical displays to be more interesting than console programs. Others appreciated when they could see "a point" to a program, by which they meant either practical use or entertainment value. Connections to a major or hobby were important for some students.

The other factors focus on the perceived difficulty of the program. Two distinct preferences appeared here. Some students were more interested in programs that looked challenging, while others favored programs that looked easy.

Table 2 shows how many students mentioned each of these factors at some point during one of their surveys. Overall, end-product factors had substantially more impact on student interest than challenge factors did. Those who did mention challenge factors were approximately evenly split on whether they preferred easiness or difficulty.

When actually choosing a program to write, students explained their choices with the same factors. However, they cited some factors more frequently than before, and others less. Table 3 shows how many students mentioned each factor to explain a choice.

Table 2: How many of the 14 students mentioned each factor to explain an interest.

<table>
<thead>
<tr>
<th>Interest factor</th>
<th>Number of students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics</td>
<td>10</td>
</tr>
<tr>
<td>Useful</td>
<td>9</td>
</tr>
<tr>
<td>Entertaining</td>
<td>8</td>
</tr>
<tr>
<td>Challenging</td>
<td>5</td>
</tr>
<tr>
<td>Easy</td>
<td>4</td>
</tr>
<tr>
<td>Connects</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3: How many of the 14 students mentioned each factor to explain a choice.

<table>
<thead>
<tr>
<th>Interest factor</th>
<th>Number of students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>8</td>
</tr>
<tr>
<td>Useful</td>
<td>4</td>
</tr>
<tr>
<td>Entertaining</td>
<td>4</td>
</tr>
<tr>
<td>Challenging</td>
<td>2</td>
</tr>
<tr>
<td>Connects</td>
<td>2</td>
</tr>
<tr>
<td>Graphics</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 1 illustrates how factors influenced choice differently than interest. I estimated the importance of a factor as its percentage of the total weight in Table 2 or 3. In the shift from interest to choice, easiness went from being one of the least important factors to one of the most, and graphics went from being one of the most important to one of the least. The other factors remained relatively stable.
Tables 4 and 5 summarize the difficulty and interest ratings that students gave to programs. Overall, across all the programs, the most common difficulty rating was moderate and the most common interest rating was interesting. However, the programs students actually chose to write were most commonly rated easy for difficulty level and interesting for interest level.

Table 4: How many times students assigned each difficulty rating to a program.

<table>
<thead>
<tr>
<th>Difficulty rating</th>
<th>All programs</th>
<th>Chosen programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy</td>
<td>29</td>
<td>11</td>
</tr>
<tr>
<td>Moderate</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>Hard</td>
<td>20</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5: How many times students assigned each interest rating to a program.

<table>
<thead>
<tr>
<th>Interest rating</th>
<th>All programs</th>
<th>Chosen programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interesting</td>
<td>37</td>
<td>15</td>
</tr>
<tr>
<td>Neutral</td>
<td>32</td>
<td>6</td>
</tr>
<tr>
<td>Boring</td>
<td>20</td>
<td>2</td>
</tr>
</tbody>
</table>

It is more informative to look at difficulty and interest ratings jointly. Table 6 summarizes the rating pairs that students gave to programs. Easy programs were most likely to be labeled boring, moderate programs neutral, and hard programs interesting. Table 7 shows the rating pairs only on programs that students chose to write. Programs labeled moderate-interesting were plentiful overall, so it is not surprising that students chose them frequently. However, easy-interesting programs were less frequent, and students appear to have chosen them disproportionately.
Table 6: How many times students assigned each rating pair to a program.

<table>
<thead>
<tr>
<th></th>
<th>Easy</th>
<th>Moderate</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interesting</td>
<td>7</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>Neutral</td>
<td>7</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td>Boring</td>
<td>15</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7: How many times students assigned each rating pair to a chosen program.

<table>
<thead>
<tr>
<th></th>
<th>Easy</th>
<th>Moderate</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interesting</td>
<td>6</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Neutral</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Boring</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

DISCUSSION

Some of the interest factors my students cited resonate with previous studies. Edelson and Johnson found that usefulness is important in learning activities [3], and one of the top interest factors for my students was the usefulness of a program. Tobias claims that prior knowledge about a topic enhances interest [8], and some of my students were interested in programs because of connections with their majors or hobbies. Cliburn and Miller noted student preferences for games [2], and while none of the programs in this study were games, my students cited two interest factors - graphics and entertainment value - that are surely related.

The graphics factor had somewhat surprising effects: a strong impact on student interest, but a weak impact on student choice. The easiness factor made the opposite shift. This may indicate that students perceived graphics programs to be difficult, or that graphics were a surface attraction that they were willing to forego in exchange for expedience. Note that they were less willing to forego other factors, such as usefulness and entertainment value.

Overall, my students rated harder programs more interesting than easier ones. This lends some support to my conjecture that challenges just at the edge of our comfort zone are the most interesting. However, when offered a choice, my students disproportionately chose to write less challenging programs than their interest patterns had suggested. They did not typically choose boring programs - just ones that were interesting for reasons other than their challenge level. Note that these programs would score highly on Hansen and Eddy's niftiness scale [5].

At the beginning of this study, I was imagining a learning environment in which students chose among exercises of varying difficulty. Unfortunately, the study has negative implications for this type of learning environment. Rather than choosing programs at the optimal level of challenge, many students would choose easy programs that happen to contain other interest factors.

Instead, I plan to ensure that my homework assignments focus on the three most important interest factors for my students: graphics, usefulness, and entertainment value.
Every program can include at least one of these factors, and most can contain more than one. It may always be challenging to cope with a wide range of student aptitude, but by focusing on these factors, I hope to increase motivation and persistence in all of my students.

REFERENCES


A CONTENT-BASED IMAGE RETRIEVAL PROGRAMMING

ASSIGNMENTS FOR

INTRODUCTORY COMPUTER SCIENCE COURSES*

Michael Eckmann
Skidmore College
Mathematics and Computer Science Department
815 N Broadway
Saratoga Springs, NY 12866
518 580-5294
meckmann@skidmore.edu

ABSTRACT
Introductory computer science courses cover certain content like arrays, loops, sorting, etc. There is evidence that students enjoy working with media (e.g. images, video, sound) and it keeps them interested in the course material [1]. This paper introduces ways in which core content covered in introductory computer science courses is used in Content-Based Image Retrieval (CBIR) systems. We also provide specific programming assignments that are intended to be engaging to the students while requiring them to show their mastery of the course content. This CBIR content and related programming assignments are intended to achieve two goals. One goal is to increase student interest and another goal is to introduce them to an active research area.

INTRODUCTION
Content-Based Image Retrieval (CBIR) systems allow users to search for images in databases, often using an image (query) as input. The content of that query image is then used to search the database to find similar images (results). This is in contrast to searching images using word tags. CBIR systems often use the color content, the texture

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

Color images are stored as a grid of pixels, where each pixel is represented by a number or set of numbers. Grayscale images (colloquially and less accurately black-and-white images) have each pixel represented by one number, usually of size 1 byte (range 0..255), where 0 represents black and 255 represents white and all the numbers in between different levels of gray. Color images often store 3 bytes per pixel, each byte representing one of the red, green and blue (RGB) components of the color. For example a pixel whose 3 RGB byte values are 255, 0, 0 would represent a pure red color, whereas 0, 128, 128 would represent teal and 135, 235, 135 represents a light green.

We taught an introductory computer science course to non-majors (a CS0-style course) using Guzdial’s "Introduction to Computing and Programming in Python, A Multimedia Approach" [2]. It was useful in a CS0 course in our opinion. However, the drawbacks to using this textbook or the Java version [3] of the textbook in a CS1 course is that the approach uses classes that are not in the Java API. Instead, they are written by and provided by the authors to make it easier on the students to utilize sound, images and video in their programs. While it may make writing multimedia programs easier, having students use non-standard libraries is undesirable in the author's opinion. The approach described in this paper, at least for utilizing images, is a simple, intuitive approach using only the standard classes in the Java API (without even using the Java Image I/O API) that would typically be taught in a CS1 course. Other motivation for us was to find interesting topics to which the students can apply the core introductory computer science content (arrays, bit manipulation, sorting, etc.) There is much research in the literature that students enjoy working with media (images in particular) ([1], preface of [2].) Students also often have their own personal image collections that they can use as the database of the CBIR system.

We introduce how to use a simple text based image file format and the steps involved in creating a CBIR system that searches images based on color content. We also divide the system into multiple programming assignments.

IMAGE FORMATS

There exist a variety of image formats. The one most often used today is jpeg. Jpeg and other popular image formats (such as gif or tiff), store compressed images to save space. PPM [5] is an easy to use, intuitive to understand image format which is part of the Netpbm [4] open source software package. Utilizing this image format as opposed to popular formats like jpeg is good for introductory computer science students due to (I) a text editor can be used to view/create/edit ppm files where students can see the pixel values in plain text, (ii) ppm files can be read/written into programs using the common Java text file I/O classes (e.g. Scanner, BufferedReader, BufferedWriter) which are often covered in introductory CS courses, (iii) jpeg is a less intuitive, compressed, often lossy image format that cannot be viewed/created/edited using a simple text editors, nor read/written in programs without using a Java jpeg package. Note: if file I/O is not covered in the introductory courses, the professor can easily provide the code for reading ppm files into a two dimensional array to the students to use in their programs.
A simple PPM file is shown below. P3 specifies that this file is a text ppm file. The # is a comment. The 6 6 line says it is 6 pixels by 6 pixels. The 255 is the maximum value of a color component. 210 220 5 represents yellow and 0 0 200 is blue. These are alternated to create a 6 by 6 checkerboard of alternating yellow and blue.

```
P3
# yellowbluecheckerboard.
ppm
6 6
255
210 220 5 0 0 200 210 220 5 0 0 200 210 220 5 0 0 200
0 0 200 210 220 5 0 0 200 210 220 5 0 0 200 210 220 5
210 220 5 0 0 200 210 220 5 0 0 200 210 220 5 0 0 200
0 0 200 210 220 5 0 0 200 210 220 5 0 0 200 210 220 5
210 220 5 0 0 200 210 220 5 0 0 200 210 220 5 0 0 200
0 0 200 210 220 5 0 0 200 210 220 5 0 0 200 210 220 5
0 0 200 210 220 5 0 0 200 210 220 5 0 0 200 210 220 5
```

COURSE CONTENT AND CBIR PROGRAMMING ASSIGNMENTS

Content-based Image Retrieval typically consists of the following steps.

1. Decide on an image representation (e.g. a color histogram, textures, shapes in the image, etc.) and compute that representation for each of the images in a database.
2. Allow the user to provide an image (query) of which s/he wishes to find similar images.
3. Compute the query image's representation.
4. Compare the query image representation to some/all of the representations in the database.
5. The results of the comparisons are a score for each compared image in the database. Sort the scores associated with the images and provide the top scoring images from the database to the user.

In this paper, we focus on color histogram as the image representation that is easy to explain to students and can be implemented using the skills the students have been taught in the introductory course. A color histogram is an image representation that does not take into account any spatial information (it ignores where the various colors in the image are) but instead only represents the count of how many pixels are of certain colors. There are 256*256*256=16777216 different colors that can be represented using 3 bytes for RGB color pixels. Since colors that have close RGB values look similar (e.g. 100, 211, 40 would look similar to 95, 215, 38) we do not want to keep track of so many colors. Each bin will represent a different color. One decision for the bins would be to separate the red values into 4 groups, the green values into 4 groups and the blue values into 2 groups. This leads to 4*4*2 = 32 bins. Since each color component is represented by one byte, to divide red into 4 groups we could choose 0..63, 64..127, 128..191, 192..255. We would divide green in the same way and we would divide blue into 0..127 vs. 128..255. Given the 3 byte pixel color, to determine the bin number associated with it, one could use the bit shift operators on each of the color components ( >> 6 for the red and green components and >> 7 for the blue component.)

For example, this would group all pixels that have a Red component of some number between 0..63 AND have a Green component of some number between 128..191 AND have a Blue component of some number between 0..127 to be in the same bin and
hence treated as the same color. For example, a pixel of color 5,129,120 would be considered the same color (put in the same bin) as a pixel with 60,190,4.

<table>
<thead>
<tr>
<th>Bin</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0..63</td>
<td>0..63</td>
<td>0..127</td>
</tr>
<tr>
<td>1</td>
<td>0..63</td>
<td>0..63</td>
<td>128..255</td>
</tr>
<tr>
<td>2</td>
<td>0..63</td>
<td>64..127</td>
<td>0..127</td>
</tr>
<tr>
<td>3</td>
<td>0..63</td>
<td>64..127</td>
<td>128..255</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>31</td>
<td>192..255</td>
<td>192..255</td>
<td>128..255</td>
</tr>
</tbody>
</table>

For each image, compute the number of pixels that fall into each bin and the color histogram representation of the image is simply an array of those 32 numbers. The array index represents the bin number, which in turn represents a color. The value stored in each element of the array is the frequency of that color in the image. To be able to handle images of different sizes, one should normalize the histogram to become a relative histogram. That is, simply divide the number stored in each bin by the total number of pixels (after this process, the sum of all the values in the 32 bins will be 1.) Then, comparison between the histograms of images of different sizes is more meaningful.

One has many options in how to compare two histograms. One simple way is Euclidean distance between the histograms. It is a simple like-bin distance measure that results in a low number when the histograms are similar and a high number when the histograms are dissimilar. The Euclidean distance is easily calculated with the following code.

```java
// assumes hist1 and hist2 are two arrays of doubles
double d, sum=0, euclideanDistance;
for (int i=0; i<hist1.length; i++)
{
    d = hist1[i] - hist2[i];
    d *= d;
    sum += d;
}
euclideanDistance = Math.sqrt(sum);
```

Other options students can explore would be to convert the RGB color values into a different color space such as HSV or CIELab and experiment with dividing the colors into different numbers of bins. Other options for comparing histograms include other distance measures such as L1, or cross-bin types like Mahalanobis distance or Earth Mover's distance. The cross-bin types allow one to specify how close or distant a color in one bin is from a color in another bin (e.g. the color represented by bin 0 vs. the color represented by bin 31 are very different, whereas the color represented by bin 0 and bin 2 are not as different).

The distances between the query image histogram and each of the database images' histograms could be stored in a list of two columns in a text file, one column being the database image file name and the other being the distance. Since those images whose distance from the query image is small are expected to be those images that are most similar (in color content) to the query image, one should sort the list from low to high distance. The images whose file names associated with the lowest distances should then be displayed to the user as the results of her/his query.
The topics that the students will have been expected to be introduced to prior to being given programming assignments for CBIR are as follows.

- Arrays, (e.g. one dimensional arrays of doubles (for histogram representation), two dimensional arrays of bytes (for grayscale image storage), two dimensional arrays of a class type RGBColor (for color image storage).) Note: RGBColor should be a class created either by the professor or students that store 3 bytes (one for red, one for green and one for blue.)
- Loops to process the array data.
- Bit manipulation (bit shift operators) for determining bin numbers.
- Text file I/O for reading in a simple image format like ppm and writing/reading a file of image names and distances.
- Sorting algorithm(s) to sort the distances

PROGRAMMING ASSIGNMENTS

Besides the core content that would be introduced ordinarily in an introductory course, the amount of extra class time that is needed to prepare the students for these programming projects is minimal. It includes:

- how images are stored as a grid of pixels and that each pixel contains a number (for grayscale) or three numbers (for color RGB)
- the ppm file format
- what a histogram is (or whatever kind of representation you prefer to teach)

The time spent on this preparatory work, though, we believe is well spent as it will convince the students of the usage of these topics in real world applications, introduce them to an active research area (CBIR) and allow them to utilize their own personal image collection for programming projects. Once they are introduced to this extra image and CBIR related material, some programming projects that can be assigned are:

- Write a program that takes an image file name (ppm format) as input and computes a color histogram and outputs it to a text file (adhering to a specified format.) Deliverables: a ppm image file, source code for program and the text file output containing the color histogram of the image.
- Write a program that reads in a histogram file name (representing the query image) and compares it to all the histograms in a directory and outputs a text file list of score/image file name pairs (adhering to a specified format.) Deliverables: a histogram file, source code for program and the text file output containing the score/image file name pairs.
- Write a program to read a text file containing score/image file name pairs and sort it by score. Deliverables: a score/image name file, source code for program and the sorted score/image name file.

Students who seem to take to this well may explore more than just "RGB color histograms" as representations of the images. Different color spaces, such as CIE Lab and...
HSV are more perceptually relevant to humans than RGB. Besides color, they could try to implement some interesting texture descriptions.

The students will typically have jpeg images that can be batch converted very simply to ppm files using freely available software. The professor could provide a script to do that as well as a script to run the image representation computation on a database of images. If the students are not familiar with the code used to display images, the professor could also provide a program that displays the results of the query — show several images, given their file names.

Pedagogical materials such as lecture notes on images, the ppm file format, color histograms and CBIR in general can be found at www.skidmore.edu/~meckmann/CBIRfor-intro.html. Scripts to batch convert jpeg images to ppms, as well as code to display multiple result images also can be found there.

CONCLUSIONS

In this paper, we outlined how CBIR can be used in an introduction to computer science course to (i) introduce the students to an interesting application they can perform on their own images, (ii) reinforce the usage of the core concepts taught including arrays, loops, sorting, bit manipulation and text file I/O and (iii) introduce the students to an active research area. We explained the general outline of most CBIR systems and suggested how this could be assigned as several programming projects.

The students in an introductory course that utilizes this material will have learned how and when to use arrays, loops, file I/O and sorting via interesting subject matter. The additional topics that the students will have learned (PPM files, how images store colors as numbers, RGB color space, etc.) will be a byproduct and provide them with new knowledge of the inner workings of things they see every day (digital images.) The ideas presented in this paper provide a way for students to not only learn the subject matter (introductory computer science topics) but also learn the basics about an active research area (there are current conferences on CBIR, and many CBIR papers are continually published in the computer vision journals.)

REFERENCES


ABSTRACT

Computer science (CS) degree programs privilege programming as a fundamental tool to teach algorithmic thinking. CS curricula start with an introductory course sequence that focuses on learning how to program, and often delay introducing material with a significant ontological component until later. The importance placed on programming as a skill, coupled with its position in the course sequence and the vacuum left by postponing identity development has understandably resulted in a rousing debate about how to teach these introductory courses. As such, the literature is filled with passionate arguments for and against particular languages, processes, and paradigms. This wealth of writing should not come as a surprise. The choices made in the teaching of this sequence are very personal ones. I believe that we, as faculty each support the teaching of programming in ways that support our own identities as computer scientists, and our own definitions of the discipline. Computer science is not programming, however, and computer scientists are not programmers. I say this not to attempt to sever the ties between the two, but to disagree categorically with the notion that the two can be directly equated. I believe that we need to consider what we want students to learn beyond the syntax of a programming language, and this paper is an attempt to start that discussion.
BACKGROUND

As a discipline, we have struggled (and ultimately failed) to agree on just what the best way is to teach programming to novices [5, 6]. There are a variety of reasons why we don’t come to a consensus ranging from the influence of outsiders on the CS curriculum to our own internal preferences and biases. Most CS faculty will agree that learning to program is integral to the process of becoming a computer scientist [17, 19]. Interestingly, we tend to be equally in agreement that computer scientist is not equivalent to programmer [2, 15].

Choosing an appropriate first language to teach then becomes a question of at best attempting to reconcile the two (trying to create an identity as a computer scientist by learning how to program), and at worst simply teaching the Next Big Thing™ that industry—in the guise of the current job market for our students—is demanding of us.

For a long time, there was little controversy about the appropriate languages for introductory teaching. Languages are designed to be effective in particular contexts, and several were developed specifically to provide teachers with appropriate teaching environments [10, 14, 16]. The notion of a "marketable" language, however, changed the landscape. Introductory programming in CS went from PASCAL to C as systems languages became more popular. When the notion of appropriate programming paradigms entered the discussion [4, 5, 18] we rapidly moved to C++, and then Java. Throughout this, functional programming (FP) adherents calmly continued to offer introductory programming courses. They are a small but strong minority, and courses taught using FP offer their own ontological component not addressed in this paper. Instead, this discussion reflects on introductory programming courses in the imperative/object-oriented tradition.

I have spent a great deal of time over the past 12 months thinking about the processes we employ when teaching introductory programming. My thinking has been heavily influenced by several things, most notably work on Threshold Concepts [12] and an invited editorial in SIGCSE Inroads about enrollments and programming [13]. In this second paper, when Manaris speaks of "nonobvious ideas," he is speaking about student engagement with, and understanding of, the material in an introductory programming course, but in a very real sense he is also talking about the ability of a programming language to "get out of the way" of student learning and understanding of core concepts in the discipline (or at the very least, core concepts in programming and problem solving).

As such, I am convinced of two things: first, that we weren’t wrong before when we considered future course work—and even industry requirements—when we went through the language selection process for the introductory programming sequence; and second, that we weren’t exactly right, either! We cannot lose sight of the learner in the teaching process, and some of our choices, at least at the beginnings, need to take into account student roles. Berglund and Lister [3] published an excellent paper on Kansanen's didactic triangle [9] and the teaching of introductory programming that addresses just this need. I struggled, however, with reconciling these two points. It seemed to me that they were relatively orthogonal, and it would prove difficult to find a way to maximize both.
DISCUSSION

The idea that students must master "nonobvious" ideas when learning to program is a profound one. It is no coincidence that the examples in Manaris' editorial [13] are for Java and Python. Java is a powerful but demanding programming language with a myriad of (at times finicky) syntax requirements. Python is much smaller and more agile. Additionally, it is interpreted (with an interactive shell), which lowers the barrier to first success in programming for all students. Research has shown that assignments that provide early feedback and early success (and that are compelling and visual) are important in improving not just retention, but also gender equity [1, 11].

We taught our introductory sequence in Java for many years. Research has shown no significant difference in performance when students are asked to switch languages between CS1 and CS2 [7, 8], so starting in the Spring of 2008 we decided to investigate various other languages for use in the first course in our programming sequence. I first switched the language focus of our CS1 course to Ruby (we are retaining Java for CS2). Since then I have taught Ruby twice more in CS1, followed by Groovy and now Scala. What follows is an attempt to identify why the particular languages were chosen, and provide anecdotal commentary on the process.

Ruby: When I first considered switching languages in CS1, I was struck with the fact that scripting languages require the mastery of far fewer "nonobvious" topics, and interpreted languages that provide us with an interactive shell lower the barrier to early programming success. I figured (unsurprisingly) that I would be better off selecting a language that I had more than a passing familiarity with. As such, I chose to offer the course in Ruby. Ruby follows the principle of least astonishment, and I believed that this would make it easier for students to develop problem-solving skills. Numerous interesting powerful frameworks have been developed in Ruby. The language would clearly remain relevant for students after the course, and student interest should remain high because the language has recently seen a strong increase in its use, and has developed a lot of positive buzz.

There was a definite downside to the choice of Ruby as a CS1 language. There is no appropriate introductory textbook. While books and resources about programming-and even learning to program-in Ruby are plentiful, they are not written to support the concepts and processes covered in a CS1 course. The lack of appropriate materials, however, was not the biggest reason why the choice of Ruby was not ideal. The students were switched to Java in CS2, and this transition proved troublesome. Students reported a great deal of dissatisfaction with Java because Ruby was easier for them to think and program in. While expected, the depth of difficulty was a bit surprising. The students persevered, but we should provide more to them than the opportunity to persevere. Three iterations of the course did not significantly improve this transition, and measurable gains in student retention were not realized. Even attempts at "pre-teaching" elements of Java programming at the end of CS1 proved to be ineffective.

Groovy: A colleague taught CS1/CS2 during the 2009/2010 academic year, and he reported similar problems when he used Python. When I next taught the course (during the spring semester of 2010), I switched the language to Groovy in an attempt to overcome these difficulties. I chose Groovy specifically because it was both a scripting language promising to provide a similarly low barrier to programming, and a JVM
language. As hoped, the switch to Groovy provided a considerably easier path to Java. Groovy scripts can seamlessly interact with Java, and the students accepted the transition to Java much more readily than with Ruby. Groovy was not without additional problems, however. Specifically:

Groovy syntax is very different from Java syntax. Groovy idioms do not look like Java idioms. Novice programmers think about problems and their solutions in relation to simple idioms in the language they are learning. While not a significant boundary, the switch to Java was made a bit more difficult because of this idiomatic shift.

Java is statically typed and Groovy is dynamically typed. Static typing turns out to be very helpful when teaching programming. I have found that novices can grasp notions of type faster when they must explicitly set a type that is enforced by the compiler.

Groovy has a wealth of control structures. Much like Ruby, Groovy has quite a few control structures provided in an effort to support a variety of programmers' styles. While in theory this could provide instructors with an opportunity to specifically target control structures for schema development, this turned out to not be the case. Scala has very few control structure and students were relatively quick to grasp the concepts.

Groovy does not respect private access modifiers. It is difficult to teach object-oriented principles when the language doesn't respect access modifiers. Interestingly, Groovy allows you to set access modifiers on entities. It just doesn't care. I hesitate to call this issue a true problem with the language, however, because it provided an ideal reason to shift the course focus to Java in CS1. Once the students recognized the importance of hiding data, they were much more open for the switch to a language that supports it.

Scala: This fall I am teaching CS1 again, and I made a conscious decision to attempt to address all of the issues I have mentioned. I wanted a language that supported programming idioms that were not clearly dissimilar to Java. I was looking for an interpreted language that had an interactive shell for rapid prototyping and development. I wanted a statically typed language. And finally, I was looking for a JVM language that privileged Java interoperability as a way to facilitate the transition to Java in CS2.

In recent years a number of programming languages have been developed that produce code that executes on the JVM. With a more nuanced list of requirements, one language immediately moved to the top of the list. Scala has been steadily developed since 2001, and represents a multi-paradigm language, combining functional and object-oriented features. Scala supports an interactive shell, and Scala scripts can be compiled into bytecode to seamlessly interoperate with Java. Additionally, the language has very few control structures (although the list is complete).

While I will admit to some trepidation about the presence of functional programming idioms—I was concerned that the students would see the presence of both functional and object-oriented idioms as confusing—I was pleasantly surprised that they were readily able to grasp functional techniques as they came up in the course. As they mature as programmers, they should have no problem thinking in-and ultimately programming in-the functional idiom.

Finally, the transition to Java at the end of the semester progressed smoothly. Because of the seamless integration with Java, it was trivial to use Java libraries when
building Scala applications (including the FANG game engine library), and due to the similarities between Java idioms and Scala object-oriented idioms, I was able to demonstrate programs in Scala and easily transition students to similar examples in Java.

CONCLUSION

The choice of language in teaching CS1 is a somewhat personal one. We each approach the course differently, and thus teach with very different expectations. In this paper I am reporting my personal approach to the course, and thus my requirements/expectation for a programming language. I am compelled by the argument that the syntax of an introductory language must minimize "nonobvious" topics. In addition, I feel that introductory languages need to be used in upper-division computer science courses, or at the very least ease the transition of students. Finally, I feel strongly that statically typed languages are safer for novice students as they learn about data and type.

For me, the choice of Scala is turning out to be quite a successful one. The retention rate between CS1 and CS2 this year (as measured by registration for the spring semester) is one of the highest we have recorded, and I can report anecdotally that the level of discourse I developed with my students is much higher than in past semesters. I attribute much of this apparent success to the switch to the Scala language.

The purpose of this paper is to hopefully foster discussion about programming languages in CS1 and CS2 as a way of identifying and updating what we want students to learn about programming. Interestingly, I feel that this exercise has served as a bit of a Threshold Concept for me. Having gone through the process, I find myself thinking quite differently about teaching and learning in introductory programming courses.

REFERENCES


INTRODUCING BIOINFORMATICS INTO THE COMPUTER SCIENCE CURRICULUM

PANEL DISCUSSION

Mark LeBlanc, Professor of Computer Science, Wheaton College (Norton, MA)

David Rilett, Assistant Professor Computer Science, Wentworth Institute of Technology, (Dobbs 303, 550 Huntington Ave., Boston, MA 02115), riletd@wit.edu

John Russo, Associate Professor Computer Science, Wentworth Institute of Technology

Michael Werner, Professor of Computer Science, Wentworth Institute of Technology

Hongsheng Wu, Associate Professor Computer Science, Wentworth Institute of Technology

With recent developments in genetics and molecular biology, and the impetus of the human genome project, the analysis and interpretation of large genomic datasets is more crucial than ever. This requires interdisciplinary expertise and the close collaboration of biologists, statisticians, software engineers and computer scientists. In response to these new challenges and opportunities, computer science students need to have a solid understanding of the fundamentals of high performance computers and networks, basic bioinformatics and its related algorithms, data mining and computational statistics. Open source tools like Linux, scripting languages and R often make up the underlying enabling science and technology.

POSITION STATEMENTS:

Mark D. LeBlanc: Modeling Interdisciplinary Science by Example

Our interdisciplinary course called "DNA" brings together biology, computer science, and ethics, providing a fast-paced introduction to scripting and offering opportunities for experimental design in genomics, while exploring the ethical aspects of living in a post-genomic world and its challenges of "personalized medicine." Activities include: a showing of the movie "GATTACA" and follow-up discussion, talks and discussions with a genetic counselor and professor of bioethics, and student-produced YouTube "commercials" of companies currently promoting genetic-based medical profiles.

See: http://genomics.wheatoncollege.edu

* Copyright is held by the author/owner.
David Rilett: *Introducing Bioinformatics to Undergraduate Programmers*

The goal of our introductory course is to investigate bioinformatics methods in Perl and to use common tools at NCBI to develop and validate projects. Translation and transcription are coded. Large Fasta files are searched for various patterns after being fetched by BioPerl and application pipelines are created. The goal is to spark interest in the programming language of life and the tools of modern informatics.

See: http://biowit.cs.wit.edu

John P. Russo: *Biological Data Mining for Computer Science Undergraduates*

It can be difficult to teach biological data mining to undergraduates who may only be required to take one database management course. One way to do this is to develop a course that introduces data mining topics using bioinformatics data sets. Introductory bioinformatics and a database management courses are prerequisites.

Michael Werner: *Bioinformatics Algorithms*

I am now teaching Bioinformatics Algorithms for the third time since 2007 using procedural programming languages (C++, Java and C#) to implement and benchmark algorithms in exact, approximate and multiple pattern matching, as well as sequence alignment, clustering and hidden Markov models. Students learn the algorithms in sufficient detail to implement them and benchmark them in several environments including their own laptops, the school's 16-processor Sun server and in virtual clusters constructed in Amazon's cloud environment. The algorithms are then re-examined to improve their efficiency, looking for opportunities to parallelize algorithms in a multi-processor environment and enhance performance.

Hongsheng Wu: *Computational Statistics for Computer Science Undergraduates*

The biostatistics course introduces students to basic statistical methodological concepts and to the software packages that are central to computation and database operations. Students with a background in biostatistics will be at a competitive advantage when entering the work force or graduate school.

- Students explore several analytical techniques and determine the appropriate analytical methods for data to be used to design experiments.
- Students learn to use statistical software packages (such as SAS or R) to solve biological problems.
- Students develop analytical tools to investigate simple biological systems.
- Students develop skills in presenting data in easily interpretable formats.

**PANELIST BIOGRAPHIES:**

**Mark D. LeBlanc,** Meneely Professor of Computer Science at Wheaton College and co-leader of the Wheaton Genomics Research Group, has been teaching interdisciplinary bioinformatics since 1998. NSF funding over the years has supported the development of course materials for integrating bioinformatics in courses across the computer science program. He co-authored Perl for Exploring DNA (Oxford, 2007) with his research and teaching collaborator Betsey Dyer, Biology. mleblanc@wheatoncollege.edu
David Rilett, Assistant Professor of Computer Science, Wentworth Institute of Technology, holds a Master's degree in Software Engineering and a bioinformatics graduate certificate from Brandeis University.

John P. Russo, Associate Professor of Computer Science, Wentworth Institute of Technology, has been actively involved in computer science education since 1999. He has been actively involved in bioinformatics research, including collaboration on modeling autism genetic networks with scientists at Harvard Medical School and Cold Spring Harbor Laboratory. russoj@wit.edu

Michael Werner, Professor of Computer Science at Wentworth Institute of Technology. His PhD in Computer Science is from Northeastern University. He also holds Master's degrees from Boston University in Computer Information Systems and from University of Illinois in Mathematics. wernerm@wit.edu

Hongsheng Wu, Associate Professor of Computer Science at Wentworth Institute of Technology, holds a PhD and M.A. in Biostatistics from Boston University, a M.S. in Computer Science from Ohio University, and B.S. in chemistry from Shandong University. wuh@wit.edu
UNDERSTANDING NSF FUNDING OPPORTUNITIES

TUTORIAL PRESENTATION

Sue Fitzgerald
National Science Foundation
4201 Wilson Blvd, Suite 835
Arlington, VA 22230
(703) 292-4641
scfitzge@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 and guidelines, the review process and 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 presenter 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].

- Transforming Undergraduate Education in STEM (TUES)
- Federal Cyber Service: Scholarships for Service (SFS)
- Research Experiences for Undergraduates Sites (REU Sites)
- Scholarships in Science, Technology, Engineering and Mathematics (S-STEM)
- STEM Talent Expansion Program (STEP)
- Advanced Technological Education (ATE)

* 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


BIOGRAPHY

Sue Fitzgerald is serving a temporary assignment as a Program Director at the National Science Foundation's Division of Undergraduate Education (DUE). She is a Professor of Computer Science at Metropolitan State University, a former department chair and past co-chair of the SIGCSE Technical Symposium. At NSF, she works on the Transforming Undergraduate Education in STEM (formerly CCLI, now TUES), the Federal Cyber Service: Scholarship for Service (SFS), and the Scholarships in Science, Technology, Engineering, and Math (S-STEM) programs.
This tutorial will introduce participants to process-oriented guided inquiry learning (POGIL) in computer science. POGIL has been developed, and validated over the last 15 years, primarily in chemistry education. In POGIL, teams of learners (typically 3-5) work on scripted inquiry activities and investigations designed to help them construct their own knowledge, often by modeling the original processes of discovery and research. Teams follow processes with specific roles, steps, and reports that encourage individual responsibility and meta-cognition. Multiple studies have examined the effectiveness of POGIL, and generally find that POGIL significantly improves student performance, particularly for average and below-average students.

The tutorial will be organized as follows. First, we will introduce ourselves and briefly review some relevant background. Second, participants will work through a sample POGIL activity to understand how it works. Third, we will review POGIL's key concepts, history, and supporting research. Fourth, participants will begin to draft their own POGIL activities to better understand the opportunities and challenges. If time permits, we will review and discuss each other's activities. Fifth, we will conclude with pointers to additional information and general discussion.
ABSTRACT

Creating a CS1 course that teaches good programming techniques and computer science concepts while also motivating students is a challenge faced by all computer science educators. We have all seen that students who are excited about their assignments make a greater effort and learn the concepts embodied in those assignments better. Based on this experience we have designed a CS1 curriculum that motivates students through assignments that involve image manipulation and creation. For this course we have somewhat reordered how concepts are typically taught in order to get students quickly into image manipulation. The choice of which concept to introduce is motivated by what it enables students to do to images. All concepts are covered using C for the first three quarters of the semester and then revisited during the final quarter using C++ to reinforce material and present object-oriented concepts.

INTRODUCTION

At Hampshire College we have no set majors and a student body that tends towards the arts. We therefore strive to offer CS1 courses that will provide a solid grounding to students focusing on computer science while also attracting students who would not usually take a computer science course. Bearing this in mind and also considering that
students tend to learn more when excited and motivated by the subject matter, we set out to design a CS1 curriculum that focuses on image manipulation and creation.

We chose image manipulation and creation because it gives students visual feedback that lets them check the correctness of their projects by visual inspection and because gives a tangible feel of accomplishment. The approach to image manipulation includes both image processing and computer graphics, each used when it is the more appropriate way to convey a new computer science concept.

The typical order that programming material taught is somewhat changed so that each new concept enables students to progress with their image manipulation. For example, students use libraries well before they understand specifics of how libraries work and learn arrays and for loops before if statements. Reorderings of this type did not appear to cause students any difficulty despite our concerns while designing the course.

The decision was made to teach this course using the C family of programming languages for two reasons. First, our computer science students take classes at a number of different colleges and these colleges do not use a common programming language. We felt that background in C and C++ would put our students in a good position to quickly learn any newer language required for the various courses. Second, we believed that by having material introduced first with C and then with C++, our students would gain a better appreciation for the differences between procedural and object-oriented programming. In addition, revisiting programming concepts in the context of a new but similar language would enable students to more firmly grasp the concepts.

RELATED WORK

Using image manipulation and graphics to help motivate students to learn concepts in the computer science curriculum and in CS1 courses in particular is not new. It has been introduced at every level from 9th grade [4] up to college database courses [6]. The integration varies from a couple of image-related projects [7, 8] to full courses [3, 5]. Common to all of these courses is that the instructors found that integrating image-related projects into the courses motivated students and helped with students' understanding of material.

One approach taken was to give students access to prebuilt libraries to give them a head start on image processing, which enables them to do more complex projects. McAndrew and Venables [4] used an environment that did not require much programming and primarily enabled students to run prewritten image processing techniques that were based on computer vision. This was effective for the 9th and 10th grade students in their course but is not as applicable to a full-semester CS1 course. Hunt [3] also used prebuilt image libraries but focused on having students use and expand those libraries. He had his students use Java's built-in image classes so that they could learn how to interact with prewritten code. While this proved effective, we wanted to start from a more rudimentary level and to build the larger framework embodied by Java's image classes.

Another approach taken was not to base a full course on images but instead to just have a couple of projects that included images. At Swarthmore College two projects were added to the CS1 course: a greyscale image processing project and a color image...
processing project [8]. These projects enabled students to learn basic GUI development while reinforcing arrays and other programming concepts. Students became fully engaged with each project. At the College of Staten Island computer vision techniques were introduced as a way of introducing research into a CS1 course [7]. Their results focused on the effects of adding a research component to CS1. However, these projects used computer vision techniques to scan brain images and some of the positive results may have come from students' excitement over getting to use techniques on images.

Clemson University has taken the most aggressive approach to integrating images into the computer science curriculum by bringing graphics into lower-level computer science courses. Graphics have been used for both a CS1 course [5] and a database course [6] as part of their Τέχνη project. For the CS1 course the approach is similar to ours using C programs to create portable pixel map (PPM) format images. UNIX image tools are used to convert images to more popular formats like JPEG and GIF so that students can view what they created. This approach focuses on 4 major image projects and uses only C; our course has 13 assignments that relate to images and also introduce C++ and a little bit of web design. Our use of a greater number of assignments enables us to give smaller assignments that enable a more gradual progression of skills.

**COURSE ORGANIZATION**

The goal of this course is to teach programming and the concepts included in a standard CS1 course while keeping students engaged and excited about the material. Part of this goal is to give students a fundamental understanding of programming that can be applied to other languages and environments. With this in mind, all course material is presented using the Emacs text editor within a command line environment. All programming is demonstrated using the terminal built into OSX; students who used Windows-based machines are encouraged to install Cygwin [2] so that they can also use a command line interface. We believe that by learning this baseline style of programming, students gain a better understanding of programming language vs. programming environment and are better positioned to learn more supportive programming environments in the future.

As with all programming courses this one begins with a "hello world" equivalent. (The exact syllabus used for the course can be found at http://helios.hampshire.edu/~pedcs/classes/cs110Spring10/syllabus.html.) The course then quickly covers variables, for loops, and arrays. Immediately after this, on the sixth day of class, the students are introduced to image libraries that enable them to read and write PPM images. These libraries are given without explanation of code and students are only shown how to use them. The students are told that before the end of the semester they will understand how the libraries work. Students tended to be a little overwhelmed when they are first asked to use these libraries but quickly move past that as they become excited by being able to create their first images. The use of Makefiles eases the introduction of libraries because students do not have to focus on how to include the libraries in their code.

At the same time students are introduced to the image libraries, they are given a basic introduction to web design and shown how to create webpages that contain images.
This is done so that students can display their images as part of homework assignments and more easily show the images to friends. Students have been excited by a computer science course that enabled them to show off their work in ways more common to art courses. Because students uploaded the web pages to our school's Linux-based student webspace, creating and posting webpages also reinforced the command line material covered.

Once students have these image libraries and have seen how to use for loops to access every pixel in the image array, they are introduced to conditionals and if statements. If statements are the logical next step as they enable students to select certain pixels in an image that they wish to change. Conditionals with their Boolean logic follow as students see how they can be applied to select the pixels they wish to manipulate.

From here the course proceeds to nested for loops. Students find this normally difficult concept easier to grasp since it is far more logical to traverse an image array by x and y coordinates than as a single dimensional array as they have been doing. With this ability to traverse the image in two dimensions, students are then introduced to basic filtering concepts like blur and haze, which are typically used with photography. Students have found it exciting to better understand how filters in Photoshop actually work.

From here students are introduced to while loops that are introduced in the context of applying image compression techniques (which keep running a process to decrease the number of colors used until a certain number is reached). Students then have structures explained and the Pixel structure they have been using all semester is put into context. They are then introduced to functions in the context of how functions can simplify their code and previous pieces of code are rewritten using functions.

We next address pointers, and this goes more smoothly than would normally be expected since the images that have been used all semester have been referenced as pointers in the code. This familiarity makes pointers easier to understand and gives students a good starting point for seeing where pointers are useful.

The final areas covered in C are libraries, input, and output. The concept of libraries is introduced and students are brought to understand how the libraries they have been using all semester function. This leads to the concept of input and output, both user and from the disk, and again the image libraries work as a map, showing students examples of how to read and write from image files.

From here the course shifts to C++ and the concepts of object-oriented programming. Students spend the end of the semester reimplementing their image manipulation algorithms in C++ as part of a piece of image manipulation software. Shifting code from procedural to object oriented sheds light on the differences between the two ways of writing code. Students seem to find this section empowering since it gives them greater insight into the similarities between syntax in programming languages that they had heard so much about while also showing them just how much programming they have learned.

The final assignment given is to build a text-based interface to their image manipulation software written in C++. This program runs from the command line and enables a user to load or create images, filter or modify images, and save images. At each step the user is given a list of options that to perform on the images. All of the options and
the interface are created by students. This assignment shows students that they had the knowledge to build an entire piece of stand-alone software that might have application beyond a single assignment or the class. This assignment gave students a great sense of accomplishment.

**COURSE STRUCTURE**

The ordering of the material presented within this course is successful only because of the structure and support built into the course. The course has 18 homework assignments given over its 26 meetings. This means that homework assignments are due twice a week about every other week (the course met twice a week). The constant assignments mean that students are up to date with material presented in a previous class and that they have worked on their own with that material before they learn the additional material that builds on it (e.g., they have done an assignment with for loops before being exposed to images).

These homework assignments are generally open ended, which encourages students to experiment. A typical example is the assignment given after students learn if statements. The assignment specifies that they create two images with code that included if statements. We discovered that such assignments tended to bring out the best in students since they tended to play with their assignments until they get interesting images, spending a lot more time programming than they would have otherwise.

The other piece crucial to this course is the use of undergraduate TAs (UTAs). We have had three UTAs who each attended class and held evening lab hours. They graded one assignment per week and generally helped the course go more smoothly. A description of our UTA program, its benefits, and our justification of it and using UTAs to help ease grading burdens has been published [1].

**CONCLUSIONS**

The goal for this course was to build a CS1 course that would keep students excited by the material throughout the semester and also appeal to students whose primary focus was not computer science. The course has been a success: students learn how to program and are engaged with the material. Part of the success is that students are given a chance to build real software that creates output that they can show to their peers. Very often the "coolest" thing a student creates during a CS1 course is an ASCII art pine tree. For this course students created real images that they took pride in showing to others.

The final assignment of the semester was a great success and left students with a sense of accomplishment. The act of building software that could be used by others and included hundreds of lines of code left them feeling that they had built something concrete. By rewriting all of their code so that it became part of this software, they not only reinforced their understanding of material but also came to better understand why commenting code is so important as they tried to understand the code they had written a month before.

By the end of the semester all students had a decent grasp of Unix and enough knowledge of web design to build basic websites. Seven of the 21 students in the course
took computer science courses again the next semester. Because Hampshire College does not have set majors, we do not have data on numbers of computer science majors.

Using image manipulation and creation as a basis for motivating students to learn CS1 material appears to be a valid method for teaching CS1. The reordering of material necessitated by this approach appears to have no ill effects on student understanding. We highly recommend this curriculum because the levels of student motivation make it a pleasure to teach.

REFERENCES


THE I-PHONE/I-PAD COURSE: A SMALL COLLEGE

PERSPECTIVE

Lubomir Ivanov
Department of Computer Science, Iona College, New Rochelle, NY 10801
Tel.: 914-633-2342
Email: livanov@iona.edu

ABSTRACT

The paper details the experience of teaching an iPhone Application Development course at a small, liberal-arts college. The course offers students the opportunity to integrate the knowledge and skills acquired in other Computer Science courses while working on exciting, real-world applications. The paper discusses the benefits and challenges of introducing an iPhone course, outlines the material covered in the course, describes some student projects, and shares a few instructor observations.

1. INTRODUCTION

Apple's revolutionary iPhone device was introduced in 2007, and has since established itself as one of the most remarkable developments in the field of personal electronics. In 2008 Apple released the first version of the iPhone SDK, which allows third-party developers to create applications for the iPhone. Soon after, several universities began offering courses on iPhone applications development. Stanford University was first, and its iPhone course (also available for free on iTunesU) became widely popular. Other universities quickly followed with their own courses - MIT, University of Maryland, University of Michigan, NYU. Additionally, the iPhone and the newer iPad have found application in many traditional courses across the curriculum [1, 2]. Some interesting and unorthodox classroom uses of the iPhone/iPad devices have been reported as well [3, 4].

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.
This paper presents the experience of offering an iPhone Applications Development course at a small, liberal arts college. We describe the material covered in the course and the projects assigned to students, share some observations arising from preparing and teaching the course, and discuss some benefits and challenges that such a course offers.

2. THE iPhone COURSE AT IONA COLLEGE, NY

The iPhone course at Iona College, NY was offered for the first time during the Spring'10 semester. It was taught as an upper-level undergraduate/mid-level graduate course, meeting once a week for fifteen weeks. Each lecture was two hours long, and was conducted in a lab setting. During class sessions each student had access to an iMac with Xcode installed, and could experiment in real time with the software being presented.

The course objectives set forth for the iPhone Applications Development course were:

- Focus on problem solving & software design principles while working on real projects.
- Reinforce students' knowledge acquired in other CS courses by integrating elements of database design, networking, computer graphics into the course, and demonstrating how app performance is affected by the hardware and the iOS operating system.
- Introduce the students to yet another programming language, Objective-C.
- Reinforce students' skills of using software modules and libraries.

The first three lectures familiarized students with the capabilities and limitations of the iPhone device and the iPhone OS (later iOS) operating system. The Xcode development environment was introduced, followed by an introduction to object-oriented programming in Objective-C. Unlike other iPhone courses, we decided to avoid using Apple's Interface Builder (IB) tool, and teach students how to create the necessary program elements in code. There are multiple reasons for this: Using IB obscures some significant aspects of software developments - we would like our students to understand how and why things work, not simply learn to drag objects around the screen. We find some aspects of IB to be poorly implemented and potentially confusing to students. Most importantly, while IB is somewhat useful in static applications, it is of little use if elements have to be dynamically created and rearranged as the program executes. As students gained some familiarity with Objective-C, we introduced more advanced issues - inheritance, composition, introspection, categories, protocols, etc. We also introduced Apple's Foundations Framework, and discussed its fundamental structure and the way it impacts iPhone/iPad software design. By lecture four, most students were relatively comfortable with Objective-C, so we introduced the application design process. We discussed Apple's Model-View-Controller application design approach, considered the basics of event handling and delegation, and introduced the first widgets - labels, buttons, text fields, switches, sliders, and pickers. For their first app project, students were asked to develop a scientific calculator using the design principles and program elements discussed in class. Testing their apps for the first time on actual iPhone devices was a significant point of excitement and pride for students.
The next project built further on the excitement of the first apps - designing a simple game. The lecture material introduced all the necessary elements: single- and multi-touch handling, simple animation, and the NSTimer class. Students were given complete freedom to choose what type of game to design. The only condition was that they employ (multi-)touch handling, animation, and timers. The submissions included several shooting gallery type games, a moon-lander game (Fig. 1), billiards (Fig. 2), and even a "Beavis and Butthead" role playing game (Fig. 3).

The next lecture elaborated on more advanced aspects of game design - multi-view handling and device accelerometers use. The lecture also provided an introduction to Apple's Core Graphics library, used for creating 2D drawings and animation. Students were asked to design a drawing using Core Graphics. Once again, student creativity was revealed in the ambitious work most of them created. Below are just two examples - a 3D spheres composition (Fig. 4) and a Picasso-style abstract drawing (Fig. 5).

To complete the game design experience, students were introduced to sound and video processing. The use the AVAudioPlayer, AVAudioRecorder, and MPMoviePlayer classes was illustrated in class by the design of a sound player/recorder app and a video player app. By the midterm, students had accumulated sufficient knowledge in different aspects of game design to be able to create a more immersive game experience.

After the midterm, the focus of the course shifted to more traditional topics. An introduction to tables and file processing was followed by a discussion of database implementation on the iPhone. Apple provides a relatively simple framework, Core Data, which transparently uses an SQLite backend for storing data. The use of Core Data was illustrated with a Restaurant Menu example, which involved the use of view controllers and a multilevel table. For their next project, the students were asked to implement a database accessible through a multi-level table interface. Once again, a number of
creative designs were submitted: Fig. 6 shows a "Birthdays" database. Fig.7 is a flight database containing log information and procedures for operating a multi-engine airplane.

![Fig. 6](image1)

![Fig. 7](image2)

The remaining lectures focused on topics such as Locations Services and MapKit, which allow the user's location to be determined through GPS, cell-tower triangulation, or wi-fi, and plotted on a map. We also illustrated the use of the GameKit framework for voice chat and text messaging. The last project for the semester asked students to develop a "Find Me!" app which had to employ Bluetooth to determine the location of another user running "Find Me!" on a different iPhone, plot the other user's location on a map, and allow the users to communicate using voice chat and/or text messaging.

Throughout the semester, design correctness and efficiency were emphasized. We considered the iPhone and iOS limitations - the small amount of application memory, the limited multithreading support (prior to iPhone 4), the peculiar implementation of virtual memory due to the flash-based secondary storage. We also focused on memory leaks - students were required to screen their apps for leaks and eliminate any leaks found. In the process, students became familiar with the Xcode-integrated Leaks tool. The last lecture of the semester summarized everything that had been discussed about app performance, and introduced students to a few additional instruments such as Activity Monitor and Allocations. We also considered the process of submitting an application for distribution through Apple's App Store. The final projects students were asked to submit had to adhere closely to Apple's submission guidelines. Student were also expected to make a public presentation of their work. The actual projects were left to student imagination, and, once again, the students did not disappoint: As expected, most final projects were games - a space shoot 'em up game (Fig. 8), a "Super Mario Brothers" adventure game (Fig. 9), and a "Whack-a-Mole" game, complete with a sci-fi storyline.

![Fig. 8](image3)

![Fig. 9](image4)
Other students chose different topics for their final projects: an NFL database (Fig. 10), a flight calculator (Fig.11), and a collaborative wireless/Bluetooth drawing app (Fig. 12).

Each student had ten minutes to present his work before his classmates, faculty, and other students. The presentations gave students the opportunity to showcase their work and practice public speaking on a technical topic - an important skill for a successful career. Students were asked to evaluate each other's performance based on a number of criteria: technical merits, project complexity, originality, user interface look-and-feel, and presentation quality. The attending faculty were also asked to evaluate students according to the same criteria. The results revealed that students were much more critical of each other's designs (average score 29.8/50) compared to faculty (41.5/50). Faculty, however, were more critical of student presentations (average 6.4/10 vs. 8.6/10 for students). This last disparity is due, most likely, to students' lack of experience with public speaking.

3. BENEFITS AND CHALLENGES - AN INSTRUCTOR'S PERSPECTIVE

An iPhone course offers numerous benefits - for students, faculty, the CS department, and the College. The level of student enthusiasm is unprecedented: For many, this course represents everything they chose Computer Science for in the first place. The main appeal, of course, is the opportunity to design iPhone-based games. More mature students, however, move past games, and focus on other creative projects. In either case, the ability to develop applications and see them ported to their own iPhones is exciting for students. They know that learning iPhone app programming is a way to earn extra income, so they readily devote significant time to the projects, unleashing their creativity and imagination. For the instructor, seeing students put in such effort into their work is a pleasant surprise. Since the dynamics of the classroom are quite different from other CS courses, the instructor may opt to make changes in the way he/she presents the course material and assigns projects, allowing students more freedom to choose projects which are exciting to them and stimulate their original thinking. The instructor may assign some projects for group work, which helps students learn to operate in a team environment, teaching them leadership skills and collaborative strategies. Public presentations of final projects helps develop students' public speaking and presentation skills. Another aspect of the iPhone course which the instructor can take advantage of is the breadth of topics related to the course subject matter. The material covered in the
course relates to just about every other Computer Science course - Programming Languages, Operating Systems, Architecture, Databases, Data Structures, Computer Graphics, Networking, and Software Engineering. This allows the instructor to relate the course topics to students' prior experience, revealing essential links between the Computer Science sub-disciplines, while reinforcing students' knowledge in a hands-on fashion. There also appears to be evidence that the iPhone course has been helpful to students taking other courses: Some students enrolled in Database Systems concurrently with the iPhone course improved their performance in the Database System course and implemented some of their database projects on iPhone devices. In the Operating Systems course, while discussing virtual memory, students brought up the iOS implementation as an example of the dependence of the operating system design on the hardware.

For the CS department, the iPhone course expands the list of offerings, adding cutting-edge material to the list of traditional courses. Student enthusiasm for the course can lead to an increase in the number of students interested in Computer Science. At Iona, a number of students have inquired as to when the course will be offered again, while those who took the course have requested that a follow-up course be offered. Student projects can be used at open houses to showcase student achievements and attract new students. At a recent open house, a large number of prospective students expressed vivid interest in the program as soon as they learned about the iPhone/iPad course and saw some of student final projects. The addition of the iPhone course to the list of departmental offerings can also benefit departments seeking ABET (re-)accreditation. It demonstrates commitment to staying current and offering innovating courses on new technologies.

Introducing an iPhone course at a small liberal arts college, however, presents a number of challenges. Foremost is the investment in technology: The CS department needs to provide a lab with Intel-based Macs on which students can carry out software development. Testing the completed app on actual devices requires that the department purchases several iPad, iPhone, or iPod Touch devices. The College needs to join Apple's University program, which requires interaction between school officials and representatives of Apple Corp.: One individual must be selected as the official College contact person, and must be able to enter into binding agreements with Apple Corp. Based on the experiences of other schools, we anticipated that joining the University program may be a lengthy process. Our approval, however, was quick - less than three weeks. Once the University program has been joined, an IT representative or a member of the CS department needs to install and configure the Xcode environment and related software on all machines to be used in the classroom. Access protocols are required by Apple to ensure that only students enrolled in the iPhone course will be granted access to the development systems. For us, this presented a significant challenge since the CS department has access to three computer labs, but two of them - the ones with iMacs - are public and open to any student on campus. The third lab, controlled by the CS department, was equipped only with high-end PCs. Fortunately, the problem was resolved before the beginning of the semester by the upgrade of the computers in the public labs, which allowed several of the iMacs to be moved to the CS Department-controlled advanced lab. Out of its own budget, the Department purchased six iPod Touch (gen.2) devices which were made available to students enrolled in the course in the lab or on a sign-out basis. The department intends to purchase several iPad devices before the next
offering of the iPhone course. Another important issue which needs to be seriously considered prior to the start of the course is that of intellectual property rights. Accordingly, students must be informed at the beginning of the course what their rights are with respect to any software developed as part of the course. For the instructor, the main challenge is selecting an appropriate list of topics to provide a broad, solid foundation for students to become familiar with software development on mobile devices. The instructor needs to familiarize himself/herself with the intricacies of iPhone/iPad software development, and be aware of potential pitfalls and issues that may arise during the course. The instructor also has to contend with new developments in the course-related material. For example, while the course was being offered, Apple introduced the iPad devices, the iOS 4.0 operating system, and made significant changes to Xcode. As a result, a number of topics had to be revised mid-way through the semester, projects had to be adapted to the new hardware and software specifications, and the instructor had to familiarize himself with the new features introduced in the latest releases. Challenging as this is, it demonstrates most convincingly to students the dynamic nature of the discipline of Computer Science and the necessity to remain up-to-date in all the latest technologies.

4. CONCLUSION

In this paper we outlined the material presented to students in the iPhone Applications Development course at Iona College, NY. We discussed several student projects, considered a number of benefits offered by the introduction of the course into the curriculum, and outlined some of the challenges faced by the instructor and the CS department. The course offering has been perceived as a tremendous success by students and faculty alike. We expect to offer the iPhone/iPad course again next semester. We are also are preparing a follow-up course on Advanced Mobile Applications Development.

REFERENCES


SOUNDLIB: A MUSIC LIBRARY FOR A NOVICE JAVA PROGRAMMER

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

ABSTRACT
We describe the design, pedagogy, and student's experiences with a library that allows a novice Java programmer to design sound and musical accompaniment for interactive graphics-based games, as well as explore the programming of simple musical compositions, sound recordings, or visual representations of music and sound. The library has been used for three semesters in our classes and is publicly available at our website.

We wish to highlight two aspects of the library. First, the library explicitly supports our test-first design approach to teaching object-oriented programming. Second, the context of musical sounds: notes, pitches, duration, instruments, how they create a melody, how they can be represented in a number of different ways, presents a unique design playground for practicing class-based design.

INTRODUCTION

Introductory programming is hard to teach. Our goal is to give the student an opportunity to design a class-based system of non-trivial complexity, yet simple enough to be manageable with only the basic programming skills the student has mastered. To support this type of design exploration we have used for a number of years libraries (named draw with variants idraw and adraw) that create a Canvas for simple drawing of shapes, and a World class that handles time events and key events (the View and the

* Copyright © 2011 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.
Control), with students designing the game behavior (the Model). Student's game code extends the World class by providing the `onKeyEvent`, `onTick` and `draw` methods and it starts the game play by invoking the `bigBang` method.

Our curriculum enforces test-first design. Students are taught to design unit tests for every method as soon as its signature and purpose are defined. The programming of the game behavior supports this pedagogy. Students can define tests that check whether the state of the world after a key event or a timer tick corresponds to the expected one.

This simple environment for designing complex games has served us well for a while, but then it began to lose its lustre. In our first course students use a series of functional languages with similar game libraries. (Indeed, these libraries have served as models for our Java libraries.) As the supporting software and the first course evolved, the games became more complex, the game libraries added sophisticated image manipulation as well as a support for programming distributed client-server based games. Coming into the second course that introduces class-based design in object-oriented language (Java) the excitement for designing games with just a simple graphics waned. The first game, designed in a mutation-free sub-language of Java was still a challenge. When asked to design yet another game in a mutable style students would recycle old games and retrofit them into the new style adding little to their class design repertoire.

Inspired by the work of Erich Neuwirth who uses programmatic music composition (in a mini language for spreadsheets, and in Logo) for introducing computer science concepts to beginners, we designed a sound library for Java. This library provides fresh design opportunities, challenges, and possibilities — very different from those the students have already seen. Additionally, the design of the library itself includes several interesting case studies in class-based design that we plan to leverage in our course.

We present this tool and our experiences with it as follows. The next section describes the design considerations for both the tunes package and the `isworld` package and highlights the key features, including a deliberate design for testability. We then illustrate the use of the library through our demo program, as well as through several student projects, and conclude with acknowledgements.

**LIBRARY DESIGN**

The SoundLib library consists of the tunes and the `isworld` packages. The tunes package implements the music/sound component and allows the programmer to compose programmatically musical sequences (melodies, sounds) and play them. The `isdraw` package extends the functionality of the original imperative `idraw` library by allowing the programmer to play notes and other sounds on each tick or in response to a specific key event. It also adds the methods for responding to mouse events.

**The tunes package: Building the orchestra**

The tunes package defines a `MusicBox` class that initializes a MIDI Synthesizer with the default Soundbank and defines the initial MIDI program change selecting 16 instruments that provide a variety of options for the student. There are methods that allow the programmer to see what is the current MIDI program (assignment of instruments to
channels) and that allow the programmer to change the instrument assignments (the MIDI program). Additional methods allow the programmer to start and stop playing one or more Tunes, where a Tune represents a channel choice and a Chord (a collection of Notes to be played or stopped). A simple sleepSome method that allows the given time to pass before resuming completes this class. The programmer can play tunes by starting and stopping a sequence of tunes with pauses in between.

To make the MIDI musical notation accessible to the programmer and to allow students with minimal musical background to use the library, the interface SoundConstants defines names for all MIDI instruments (e.g. PIANO, TUBA, or BIRDTWEET), provides simple names to represent the notes in the middle of the piano keyboard e.g. noteC, or noteDownG), and contains a mapping of MIDI instrument numbers to their names. This makes it possible to define a Tune as simply as:

```
Tune pianoA = new Tune(PIANO, new Note(NoteA));
```

Once a synthesizer has been initialized and the assignment of the instruments to the sixteen MIDI channel has been completed, the tunes to be played need to specify only the note and the instrument on which the note should play.

When designing a musical component for a game, all that student needs to do is to decide which notes or tunes should be played on each tick, or in response to the key event. Adding the chosen notes or tunes to the appropriate TuneBucket plays the selected melody. We carry the tunes in a bucket, so even the most musically challenged programmer could carry a tune :).

The Note class allows the programmer to define any of the 128 MIDI pitches with a selected duration in a number of different ways. The class is designed to accept a number of different formats for defining the note, supporting different types of users in a comprehensive manner. The programmer can specify just the pitch. The default duration is one tick. Of course, students may not want to remember that 60 represents the middle C, and so the note names defined in the SoundConstants interface provide an easy way out. So, the middle C note of duration 1 may be defined in any of the following ways:

```java
Note midC = new Note(60);
Note midC = new Note(noteC);
Note midC = new Note(60, 1);
Note midC = new Note(noteC, 1);
Note midC = new Note("C4n1");
```

The last variant uses a String representation of the note, given by the note name C, the octave 4, the modifier (one of n natural, s sharp, or f flat), and duration (1 tick). The design of the constructors in this class provides a case study for designing classes for user's convenience and for assuring data integrity. The class provides additional methods: nextBeat and skipBeat that either decrease or increase the note duration. These methods are used when a note is playing for the duration of several beats.

A Chord is a collection of Notes that are to be played at the same time. It can be initialized in a constructor to a given sequence of Notes, or ints (pitches) or Strings (note names). It can also be modified later by adding notes to the chord. This class also includes the methods nextBeat and skipBeat that either decrease or increase the note
duration for all notes in the Chord. But here we come with an interesting challenge. When the programmer decides that a given Chord should be played by adding it to the TuneBucket, she does not expect the Chord to change as the program plays the tune. So, we need to make a deep copy of the Chord before it starts playing, to assure that the mutation of the state as we progress through the beats has no ill effects. This provides a nice example for learning about the meaning of deep copy and illustrating the need for it.

The class Tune represents a chord that is to be played on the selected instrument. Each tune specifies the instrument number from the MIDI program and the Chord that should play. The class TuneBucket then contains 16 Tunes, one for each of the 16 instruments in the current MIDI program. The programmer can add to the TuneBucket one note, one Chord, a Tune, a collection of Tunes, and also clear the contents of the TuneBucket that stops playing all notes and removes all Tunes from the TuneBucket. The methods nextBeat and skipBeat work in the same way as for a single Chord.

To represent a melody, student starts with a sequence of notes. For example, the sequence of notes (noteC,0,noteD,0,noteE,0,noteC,0) represents the first phrase of the Frere Jacques tune. The pitch 0 represents a silent note, a pause. The next note to be played can be generated by a circular iterator, creating an infinite melody loop. We can combine two traversals over the melody with the appropriate delay in the second traversal to play a musical canon or we can play several instruments in parallel as an orchestra would do.

The isworld package: Making games

The isdraw library provides a Canvas for drawing simple shapes (rectangles, lines, circles, disks, and text) in any color and size. It defines an abstract class World that handles the creation of the frame with the Canvas, the drawing of the representation of the current world scene, the handling of the timer events, the key events, and mouse events. The programmer designs a class that extends the World and implements the methods that produce the changes in the World in response to the clock tick (onTick method), in response to a key press and release (onKeyEvent, onKeyReleased), and a method that draws the current state of the World. Optionally, the programmer can override the stubs of methods that respond to mouse events.

To start playing a tune at a given time, the onTick method includes a command that adds the selected Tunes to the tickTunes TuneBucket. The selected notes will play on the given instrument with duration measured in clock ticks. The Tunes added to the keyTunes TuneBucket play as long as the key is held down, regardless of the given duration. The onKeyReleased method does not affect the notes played --- it is provided so the programmer can take other actions when the key is released.
Testing support

The SoundLib library has been deliberately designed to support the test-first pedagogy. Every class in the tunes and isworld packages comes with methods that allow the programmer to check the effects or outcomes of methods defined in that class. The MusicBox class allows the programmer to check the current channel assignments to instruments (getProgram(int channel) method), and to check which tunes are currently playing provided by the nowPlaying method. The Note class includes a method sameNote that checks whether this note represents the same note as the given note. It verifies that the notes have matching pitch and duration (for example, the two notes represented by the Strings "G4s2" and "A4f2" are considered the same). The Tune class and the Chord class both allow us to check their size, and whether they contain the given Note. The TuneBucket class allows us to check whether it contains a given note played on the given instrument, and it reports the size of the TuneBucket.

THE DEMO PROGRAM AND STUDENT PROJECTS

To illustrate some of the ways the library can be used, and to help the students to understand how to program music we have designed a demo program. The goal is to show the multiple ways of how the musical ideas can be represented, and how the user can observe and control the music that is played.

The right panel shows the current MIDI program: the assignment of instruments to the 16 MIDI channels. The user can choose any one of the instruments via mouse click. This way the students can hear how the different instruments sound. The displayed keyboard on the top left provides a guide to the user who wants to play a piano. The labels on the keys show the note that is played and the key press that plays the note. The note that is currently played is highlighted. The bottom left panel shows a piano roll. The
musical staff shows the notes that will be played on each tick. The arrow keys stop, pause, reverse, and stop the playing of the selected tune. As the piano roll plays, the currently played notes are shown in red.

**Student experiences and what they taught us**

The first time we used the library we had no duration for the notes, no chords, and the instrument-note request had to be added to the TuneBucket one at a time. Yet students came up with truly engaging jazzy tunes, using silence between the notes to create the right timing. Before the second semester we improved the design with note duration and a more complex structures for creating musical compositions (almost everything described here except the mouse interactions and key press duration).

The final projects during the second semester were quite creative. In the frogger game a jazzy tune plays in the background, each collision and when the frog reached the other bank of the river produces sound effects, and a great final tune plays when the game ends. A memory game asked the player to remember the sequence of square choices and music that went with it. One student sonified the Game of Life. Another project was a tool for composing music with playback (see below). The user could select one of four instruments, and pick the pitch to play at the given time on a graphical board that resembled the piano roll: the vertical dimension represented the pitch, the horizontal direction represented the timeline. Pressing the start key played the composition representing the current time by a thin horizontal line. This project made it clear that we need to add mouse events to the library, as all pitch selections were done by just moving the cursor using the arrow keys.
Another student tried to capture the melody played by playing the keys like a piano keyboard and playing it back on demand. His project motivated the re-design of the way the key press and release is handled in the current version of the library. Overall, the complexity and creativity exhibited in student projects confirmed the expected benefits of providing this library.

ACKNOWLEDGMENTS

The author would like to thank Eric Neuwirth for his inspiration and encouragement of exploration of programmatically generated music. His work on using music with spreadsheets and Logo have influenced the library design and its pedagogy.

REFERENCES

UNTANGLING THE MAZE OF PEDAGOGICAL STRATEGIES
FOR INTRODUCTORY PROGRAMMING

PANEL DISCUSSION

Frances Bailie
Associate Professor of Computer Science
Iona College
New Rochelle, NY 10801
(914)633-2335
fbailie@iona.edu

Adel Abunawass
Professor of Computer Science
University of West Georgia
adel@westga.edu

Smiljana Petrovic
Assistant Professor of Computer Science
Iona College
spetrovic@iona.edu

Deborah Whitfield
Professor of Computer Science
Slippery Rock University
deborah.whitfield@sru.edu

Many Computer Science Departments are continually searching for the ideal way to introduce programming to novices: C++, Java, Python, Alice, Greenfoot, BlueJ, objects first, objects last, objects never, and so on. By varying the language and the approach, faculty seek the right combination to teach problem solving skills using techniques that capture the students' attention, form the basis for future instruction and encourage continuation in the major. The purpose of this panel will be to explore strategies that have been implemented at three institutions in introductory programming courses. Panelists will discuss the pros/cons of different techniques, offer their advice and assessment of the techniques, and present a comparison of the different strategies. Panelists will encourage participation from the audience, thus providing a forum for sharing ideas and experiences in introductory programming.

___________________________________________

1Contact person

* Copyright is held by the author/owner.
PANELISTS' POSITION STATEMENTS

Adel Abunawass

In our efforts to improve our retention, progression, and graduation rates we have continually revised and evolved our introductory courses. In response to low retention rates in CS1 and CS2, we drastically overhauled these two courses and added a new CS0 course. Our goal in creating the CS0 course was to prepare students to meet the challenges of CS1 and CS2 and to provide non-majors with a gentler introduction to computer science. In the process, we have utilized a number of languages and environments including Alice, Java, Ruby, C#, Greenfoot, Eclipse, BlueJ, and Visual Studio. Over the past eight years, our retention rates have improved in our introductory classes, with variance over time. Currently we use an objects-first approach with Java in our introductory courses with Greenfoot in CS0 and BlueJ in CS1 and CS2. Although we have seen a definite improvement in our retention rates over the years, our efforts are ongoing and we do not anticipate finding a "silver bullet." As various federal and state STEM education initiatives bear fruit, we hope to see students who are better prepared for the study of computer science as well as more women and minorities choosing computer science as a field of study.

Frances Bailie

Faculty in the Department of Computer Science at Iona College have struggled to attract students to the major, as have our colleagues in many other institutions. Of those students who do enroll in our introductory course, many believe that the demands of the course are too rigorous and eventually drop out. We started teaching Java over ten years ago, both with an objects first approach (using BlueJ) and with an objects last approach. Assessment results indicated that students did not truly understand objects and classes under either approach, and so we kept Java but switched to a completely procedural approach and postponed objects until the following course. This experiment showed marginal improvement in retention and comprehension but assessment results indicated that students were having difficulty not only with data structure concepts such as arrays but also with the syntax of the language. In Fall 2010, we switched to Python in the introductory course. We also moved to Python in the programming unit of our core Computer Science course (having moved from Visual Basic to JavaScript previously). Preliminary results of this last change are encouraging, especially from faculty. Students appear to be more engaged and so we are hopeful that this move will improve student comprehension and retention. Time will tell.

Smiljana Petrovic

Dr. Bailie will present here the departmental experience of Iona's Computer Science Program; I can offer some observations from my personal experience. While Java was the language of choice for the introductory programming courses at Iona College, one attempt was teaching it using a multimedia approach. The motivation was to use a setting that students can easily relate to and where their solutions are applicable to real life. Image manipulation problems allow for some programming concepts to be naturally
introduced. However, to avoid a narrow association of programming concepts with a particular type of problem, and to provide upper level students from other science departments with examples that relate to their majors, equal attention must be given to non-multimedia problems. DrJava programming interface with its Interactions Pane allows using Java as an interactive language, and works well with a multimedia approach. Currently, our department is using Python in its introductory programming courses. Python's clear syntax allows directing more attention to problem solving skills. I further emphasize it by demanding a full solution of a problem that includes a list of relevant test-cases, an algorithm, a working, commented code and runs on test cases that match expected outputs. This procedure raises students' independence and confidence in their solutions, which I find particularly important.

Deborah Whitfield

The Computer Science Department at Slippery Rock University found that student retention was low in CS0 (introductory programming) and CS1 (advanced programming with introduction to objects). Furthermore, assessment data showed that as soon as conditionals were introduced in CS1 critical thinking and problem solving skills decreased. In 2003, the department began experimenting with using Alice in CS0 and Java in CS1. Experimentation included: 1) CS0 taught entirely in Alice, 2) ten of fifteen weeks using Alice and then introducing Java, and 3) three to five weeks of Alice as an introduction to Java. Although students taking CS0 for general education requirements enjoyed Alice, the students majoring in a computing, math, or science disciplines found Alice to be too childish. Using Alice attracted more non-majors to CS0 and increased retention in CS0, but had no discernable impact on retention in CS1. For Fall 2010, the department is taking a step backwards to C++ in CS0 and CS1, and Java in CS2. At the same time, a course using Alice is being proposed for general education.

ABOUT THE PANELISTS

Dr. Adel Abunawass is a Professor and Chair of Computer Science at the University of West Georgia. The Department of Computer Science offers a Bachelor of Science (BS) in Computer Science (CS) and a Master of Science in Applied Computer Science programs, as well as four graduate certificates in computing. The BS in CS program received its initial accreditation by ABET in 2002. Dr. Abunawass has led the department through the program's initial accreditation in 2002, and re-accreditation in 2008. He has been instrumental in leading the assessment efforts in the department as well as in designing the assessment system which uses open-source software tools to support the development and analysis of course portfolios. Dr. Abunawass has published papers on accreditation and assessment. He also serves as a program evaluator for ABET.

Dr. Frances Bailie is an Associate Professor, Assistant Chair and Chair of the Assessment Committee in the Department of Computer Science at Iona College. She has been teaching introductory programming for over thirty years using a wide variety of languages. One of her research interests has been novice programmers and how they learn. She has been a major contributor to the department's assessment efforts from our initial ABET accreditation to our recent application for reaccreditation, taking
responsibility for the collection and analysis of data. The use of assessment results for program improvement has resulted in numerous changes in our approach to introductory programming.

*Dr. Smiljana Petrovic* is an Assistant Professor at Iona College in the Department of Computer Science, where she has been teaching a variety of courses, including introductory programming courses in Java and Python. She taught introductory programming courses in very different environments: at a college in Serbia, as an Adjunct at CUNY, and AP and IB courses at the high school level.

*Dr. Deborah Whitfield* is a Professor and Chair of the Computer Science Department at Slippery Rock University. She spearheaded the collection and analysis of the assessment data that was used to justify the modifications to CS0 and CS1. She has taught both CS0 and CS1 numerous times in her 20 years at Slippery Rock and has published and presented at regional and national levels regarding assessment, accreditation, and pedagogical topics. As a faculty member, she has assisted the department with creating and revising student learning outcomes, developing and revising assessment methods, analyzing data, and implementing a feedback loop that includes constituent feedback. As a result of assessment, the department has made numerous changes to the degree program including increasing the credit hours of CS1.
ABSTRACT

Ensemble is a Pathways project of the NSF NSDL (National Science Digital Library) dedicated to supporting computing education. This tutorial will introduce Ensemble and enable faculty members to take advantage of the Ensemble facilities. The session will also provide an opportunity for the CCSC community to provide feedback about the Ensemble facilities and suggestions about the on-going development of the portal.

OVERVIEW

Ensemble is a portal for connecting computing educators. As a Pathways project of the NSF NSDL (National Science Digital Library) program, Ensemble provides collections, communities, and tools for computing educators. The collections consist of freely available computing education resources stored within Ensemble and also provide access to instructional materials stored 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 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 has been developed as a way to provide access and visibility to collections of computing education material curated by a broad range of computing faculty and groups. The success of Ensemble over the long term depends on the owners of these collections choosing to make them visible through Ensemble.

This tutorial will introduce facilities that Ensemble provides to enable faculty members to take advantage of the Ensemble facilities. The session will also provide an opportunity for the CCSC community to provide feedback about the Ensemble facilities and suggestions about the on-going development of the portal.

TUTORIAL CONTENT

The topics covered in the tutorial will include:

Overview - The session will begin with a short discussion about the history of NSDL and Ensemble, other NSDL collections of interest to computing educators, and existing collections of computing education resources.

Ensemble Collections - The coverage of Ensemble collections will highlight some examples of collections of computing education resources within Ensemble and collections that are not part of Ensemble but that can be accessed via the portal. The emphasis for this discussion will be for participants to become familiar with Ensemble services and facilities from the following perspectives

• Access to instructional materials - Ensemble provides facilities for locating and evaluating useful instructional material. This includes a federated search across collections, and ability to comment on resources, provide ratings, and tag items of interest.

• Existing collections - Collections that are and will remain external to Ensemble can become visible within the portal by connecting with Ensemble. The connections may include being part of the federated search by using the Open Archives Initiative Protocol for Harvesting Metadata. In addition, a collection may make use of Ensemble features such as the instructional resource commenting, rating, and tagging. This is accomplished by exposing these Ensemble facilities within the site that hosts the collection. The tutorial will explain what is needed in order for an existing collection to take advantage of these Ensemble facilities.

• New collections - The Ensemble team is also reaching out to faculty interested in creating new collections of computing education instructional materials and sharing them via the Web. The intention is to lower the barrier to entry so that faculty can share small targeted collections that they create and maintain.

Communities - Ensemble provides facilities to create online communities to support interaction among computing educators via facilities such as discussion forums, posting of working papers, and connections to venues such as Twitter and Facebook. At present, 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. The tutorial will discuss some of the open communities already started and explain facilities that may be of use to other computing education efforts that would find this venue useful.

**Tools** - Ensemble also contains several tools that provide access to more advanced facilities to help instructors 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. The tutorial will briefly introduce these tools.

**Future direction** - The Ensemble Computing Portal is still evolving, and the team is interested in input from the CCSC community about future direction. The tutorial will include time for participants to provide input and suggestions about features they would find useful to support the community of computing educators.

**ACKNOWLEDGEMENT**

This material is based upon work supported by the National Science Foundation under Grant Numbers 0534762, DUE-0840713, 0840719, 0840721, 0840668, 0840597, 0840715, 0511050, 0836940 and 0937863.

**PRESENTER BIOGRAPHIES**

**Gregory W. Hislop** holds a joint appointment in Information Science and Technology and Computer Science at Drexel University. He has broad involvement in computing education and online education. In addition to working on Ensemble, Dr. Hislop currently directs NSF projects to re-design first year computing courses, to involve students in humanitarian open-source projects, and to explore vertically integrated student teams.

**Lillian (Boots) Cassel** is a faculty member in the Department of Computer Science at Villanova University. She is lead PI of the Ensemble project. Dr. Cassel is extensively involved in computing education including being a past chair of ACM SIGCSE and current member of the ACM Education Board.
Game development and mobile computing have been successfully used to increase student motivation. However, instructors with no background in mobile computing, computer graphics, and/or game development may find it difficult to develop or adopt course materials on these topics. This workshop is designed to address these concerns. Using Java Micro Edition, we have developed several project-based course modules focused on mobile game development and designed to study fundamental programming principles (e.g. loops) while also exposing students to more advanced concepts (e.g. databases). Using a mobile phone emulator, participants will test-drive one of our modules and develop a simple game, which can then be transferred to and played on a mobile device. A Windows or Mac laptop is recommended.

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

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

A number of free software packages will be used during the workshop. These include Java Platform Micro Edition Software Development Kit 3.0, J2SE SDK version 1.6 or later, and Mappy Editor for tiled maps. The workshop participants will be provided with handouts describing how this software can be downloaded, installed, and used. Participants will also have access to the source code, data, and relevant course materials for the currently existing mobile game development modules. Workshop attendees will also be invited to register on the project website that hosts all current versions of the modules, along with a discussion board for instructors interested in or those who have already adopted our course materials.

**PRESENTERS' BIOGRAPHY:**

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

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

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

FACULTY POSTER

Bridget Baird and Ozgur Izmirli
Department of Computer Science, Center for Arts & Technology
Connecticut College
New London, CT 06320
860 439-2008
bbbai@conncoll.edu and oizm@conncoll.edu

Motion capture is a field that presents many opportunities in a CS curriculum, both for coursework and for research. As motion capture has developed and also become more accessible, placing it in a CS curriculum has become more advantageous. Some of these advantages include the possibility of writing interesting algorithms for manipulation, the wide range of applications, the interdisciplinary nature of the field, the introduction of 3D techniques and the "pop" culture relevance. Applications include character animation, dance analysis and synthesis, avatar animation, movement analysis for athletes, gestural control and study, and tools for persons with disabilities. In this abstract we talk about some of the ways motion capture can be incorporated into courses and then detail some of our own faculty/student research with dance.

Optical motion capture uses multiple cameras to record movement, which is then transformed by software into 3D animation of the movement. Our own (fairly typical) set-up has eight cameras placed in four corners of a space; each corner has two cameras at different heights. A person wearing black clothing has approximately 32 sensors placed on joints and various parts of the body. As the person moves the cameras record the positions of all visible sensors. Upon completion of the movement the software produces a 3D animation of a stick figure performing these same motions. Oftentimes this animation has to be cleaned; for example, if there is much twisting and crossing of limbs that contain sensors, the software can become confused about which sensor goes with which location. Once the animation is complete, it can be manipulated, edited, placed onto characters, and used in a variety of circumstances and for a variety of purposes.

There are several places where we have incorporated motion capture into our courses. We teach a course in graphics and 3D environments (intermediate level) and we now include a section on motion capture. Students are intrigued by using this technology and are able to perform and to capture the movements. They learn how to manipulate the motion sequences and then, with an introduction to programs such as Autodesk Maya or 3Ds Max, they can bring the motion capture sequences into these programs for further
manipulation and eventual use in 3D worlds. This adds an intriguing dimension of realism to these worlds and forms part of the discussion of interface design and navigation choices in virtual worlds. Another place where we are using motion capture is in our animation workshops. Connecticut College has an interdisciplinary Center for Arts & Technology and we teach an intensive workshop in 3D modeling and animation using Maya. Often we also offer a more advanced workshop on the topic of character animation. We are now incorporating motion capture into these workshops. A third coursework area that we are exploring (but have not yet implemented) is in the introductory CS course. Our introductory course uses Python and has a problem-solving, interdisciplinary bent to it. The software that is used for manipulation of motion capture sequences (Motion Builder) allows for the writing of algorithms in Python. We are looking for some good examples where the students can do some interesting programming that will manipulate the animations; this would be an intriguing application of object-oriented programming and algorithm development.

In addition to inclusion in courses, we are doing some faculty/student research in this area. Once again, for all the reasons stated above, this is an attractive area for student research. It is also a field where extremely interesting questions can be examined. We are currently working with members of the dance department to first develop a pedagogical tool for some of their classes and then to provide a creative way to conceive of and study choreography. Motion capture has been used in this field; for example see [1][2]. One of the reasons that it is becoming so important is that the field of dance is so visual. Thus it is difficult to describe with text either what one is seeing or what one would like to produce. One method to document and explain dance has been through video, but that often lacks the tools of analysis and precision that are needed for more universal descriptions. Laban notation bridges this gap somewhat, but is a purely descriptive tool that lacks the visual component. Thus motion capture, and a process that uses small sequences of motion capture as building blocks, offers a fruitful approach. After extensive consultation and collaboration with our colleagues in the dance department we decided on a process that uses descriptive and visual components. Our motion capture research begins with distilled and essential movements that take their roots in Laban notation. We have captured dozens of these snippets using motion capture on dance students who perform very specific kinds of gestures. The short sequences are then tagged with Laban notation descriptors. The goal is for dance students in the classroom to be able to combine these sequences in order to study, understand and choreograph longer dances that illustrate specific concepts in dance. The algorithmic, computer science end of this collaboration involves interface design and, most importantly, writing code that will combine discrete sequences of movements in a smooth manner that also respects the underlying dance concepts, producing a very valuable pedagogical tool. Upon completion of this part of the project (during this spring semester) we intend to then move to an analysis and synthesis of dance using motion capture data. All phases of this research have involved students and it has been a truly cross-disciplinary experience that demonstrates the relevancy and versatility of computer science in the undergraduate curriculum.
The authors would like to acknowledge our computer science student Ajjen Joshi, our colleagues David Dorfman and Lisa Race in the dance department and many dance students for their participation in this research.

REFERENCES


This research investigates the status of computer science (CS) education in the secondary schools of Western New York (WNY). Nationally, enrollment in computer related majors is declining in secondary schools. Prior studies show that (1) most state education departments do not certify CS as an academic area in K-12, (2) K-12 teachers who teach computer courses are most commonly trained in other disciplines, and (3) too few opportunities exist for professional development for such teachers. The primary purpose of the current study is to design professional development opportunities for secondary teachers teaching CS or related courses. Thus, secondary school teachers teaching CS or computer related courses will constitute the target sample for this research. Data about the current status of CS education in high schools will be collected from the teachers through an online survey. The survey has 3 sections: (1) demographic questions related to the teachers and their schools; (2) types of CS or related courses taught in the schools; and (3) self-identified areas of need and interest for professional development among the teachers. The survey is comprised of a combination of items we deem relevant to the study and questions derived from the pertinent literature [1,2] on this topic. An initial survey will serve as a pilot study to test receptiveness of teacher to participate in the online survey, as well as the suitability of the questions. Soon thereafter, the survey will be administered to teachers at all high schools in the WNY area. Results from analysis of this data will be available for and featured at the conference. The primary significance of the results of this study lies in the analysis of the current status of Computing Education in WNY High Schools; result will help inform the design and development of professional development opportunities to benefit WNY high school computing teachers and provide information to help broaden participation and diversity in computing fields. Future research will include a more comprehensive survey of computer teachers across the entire state of New York.
REFERENCES


Many colleges and universities have programs of general education - where courses are grouped into different categories and students have to select at least one course from each category in order to get a well-rounded education. These categories often include subjects like language studies, art, history, and mathematics, but very rarely is there a category in which computer science courses are included. In this work, we argue that computer science courses develop mathematical skills no worse than mathematics courses, and can therefore be included in the mathematics category of general education. We have assessment data which shows that certain computer science courses developed mathematical skills to an even higher extent than introductory mathematical courses such as statistics and pre-calculus. Our investigations can be used when computer science courses are proposed for the mathematics category of the general education requirements; we have not found any previous research or papers supporting this fact.

Computer science not only requires a mathematical background, but actually employs and develops mathematical models, methods, tools, denotations, reasoning, and writing. It is listed as a mathematical category by the American Mathematical Society. Computer science is interrelated with branches of mathematics such as number theory, numerical analysis, logic, geometry, discrete mathematics, statistics, and many others.

We should note that the notion "computer science" is deceiving; it is perceived that computers play a central role in the field. However, there is much more to computer science than just computers; as one of the founders of computer science, Donald Knuth says "If I had a chance to vote for the name of my own discipline, I would choose to call it algorithmics." [1] Informally speaking, an algorithm is an effective method for solving a problem expressed as a finite sequence of steps. Examples of algorithms studied in general college mathematics up to Calculus II and linear algebra are the algorithms for
addition, subtraction, multiplication and division of multi-digit numbers, solving linear and quadratic equations and inequalities, calculating determinants, finding the derivative of a function, integrating, finding the local maxima and minima of a function, performing arithmetic operations on complex numbers, finding the transpose and inverse matrix, etc.

In computer science courses, problem solving requires more creativity than in mathematics courses, because the instructor does not formally describe the algorithms, but rather leaves this task to the students. Thus, students first have to formulate an algorithm, then describe it, and finally plug in several sets of numbers to see if it gives the desired output. In this process, special attention is paid to the limitations of the algorithm and all "special cases" are carefully elaborated - something that in mathematics is usually skipped under the term of "exceptions." While a computer quickly calculates the result, the student would solve the problem on paper according to the devised algorithm and compare that answer with the computer's answer to see whether the computer solved the problem correctly. When the answers differ, the student would revise the algorithm and repeat these steps. Therefore, computer science courses generally develop problem-solving skills and creativity much better than mathematics courses. The student actually acts as an instructor for the computer, and teaches it how to solve problems. This process not only requires a mathematical background and implementation of quantitative methods from arithmetic, algebra, or geometry, but actually entails the formulation of original mathematical models and rules, representing them symbolically, and drawing inferences. Most importantly, the limits of mathematical methods are identified - something that is usually omitted in traditional mathematics courses.

In 2010, at SUNY Fredonia, certain introductory computer science courses were assessed together with other courses in the Mathematics category. Out of 1045 students, a random sample of 209 was selected. For each student, 3-5 assignments were examined. The courses were split into 4 groups: Pre-calculus, Calculus, Statistics, and Computer Science, and the results were generalized based on expected learning outcomes for that category of general education. The results show that all the learning outcomes were met to a higher extent in computer science courses than in the other courses in this category. Specifically, the difference for the outcome of recognizing the limits of mathematical and statistical methods is 17.3%; furthermore, the outcome of estimating and checking mathematical results for reasonableness is met completely (100%) [2].

Computer science courses can be included in the mathematics categories of general education program, because they not only satisfy the learning outcomes of mathematics courses, but in some cases meet them to a higher extent. As further work, we are planning to accumulate more assessment data from other colleges and institutions, and perform a thorough review of the subject.

We would like to thank Dr. Mary Carney from SUNY Fredonia for providing us with assessment data. This work is supported in part by the Cooperative Research Project of the Research Institute of Electronics, RIE, Shizuoka University, Japan and by NSF Grant No 0802994.
REFERENCES


dLife is a Java library developed for teaching and research in robotics, artificial intelligence and computer vision. While a significant number of platforms now exist for using robotics in the CS classroom (e.g. Lego, Pyro, Myro, MS Robotics Studio, Player, Tekkotsu, Webots, and others…) dLife presents a unique combination of features that will be attractive to some educators. dLife allows students to program robots in Java, is free and open-source, runs on Mac OX and Linux and supports a variety of popular educational robots both physically and in simulation.

The centerpiece of dLife is the ControlCenter, a GUI application designed to facilitate the use of dLife in the classroom. The ControlCenter allows students to observe robot sensor values, manipulate robot effectors and experiment with the effects of image filters all without writing any code. When it is time to write code, dLife provides access to all of its features through an API that has been designed to be simple enough for use in introductory CS while also being powerful enough for use in upper level courses and research projects. In typical classroom use, students use the ControlCenter's integrated Java editor/compiler to create Controller objects (Java classes that use the dLife API to interact with the robot). With just a few mouse clicks the ControlCenter will compile, load and execute a Controller. The ControlCenter manages all of the details of connecting to the robot, sending commands and retrieving sensor values, allowing students to focus on programming the control logic. For advanced applications, the dLife API can be imported into any Java project and can be used more flexibly without the aid of the ControlCenter.

The dLife API includes packages for robot control, computer vision, neural networks, genetic algorithms and grid computing. The robotics packages currently provide support for the following robots: MobileRobots Pioneer 3DX; K-Team...
Hemisson, Khepera 2 and Khepera 3; and Sony Aibo. Robot simulations are supported through the free open-source Player/Stage system. dLife's computer vision package provides a collection of basic image filters: color matching; blobification; blurring; edge detection; etc. Images for processing can come from a variety of sources: image file (jpg, gif, etc.); sequence of image files (e.g. img1.jpg, img2.jpg, …); streaming video from the Aibo; and streaming video from any V4L device (Linux server required). The neural network package includes feedforward, backpropagation, Elman and CMAC neural networks. The genetic algorithms package provides a generational genetic algorithm with a variety of selection mechanisms and reproduction operators. Grid computing is supported through an interface to Apple's xgrid technology. All of dLife's packages have been designed with extensibility in mind facilitating the creation of new types of neural networks, genetic algorithms, robots, robot devices, image filters and image sources.

dLife currently runs on Mac OS X and Linux. It is available for free under the GNU General Public License (GPLv3). For additional information or to download dLife please visit: http://www.dickinson.edu/~braught/dlife.

Acknowledgements: I would like to acknowledge the contributions that many of my students at Dickinson have made to dLife. Russell Toris developed the Khepera 3 support and the interface to the Player/Stage simulation system. Kent Carmine, Nate Mitchell, Alex Diehl, Luke Maffey and Mike Olasin all contributed to the Aibo support. Lewis Flanagan developed the Elman Neural Network.
AN UPDATED TAXONOMY OF WRITING IN COMPUTER SCIENCE EDUCATION

FACULTY POSTER

Mark E Hoffman
Department of Mathematics and Computer Science
Quinnipiac University
Hamden, CT 06518
203-582-8449
mark.hoffman@quinnipiac.edu

Analyses of the computing education literature reported by Hoffman et al. [4] and Dugan and Polanski [1] agree on a taxonomy of writing in Computer Science education where writing can be categorized as Writing to Learn, Academic, or Professional. Writing to Learn (WTL) writing is used to develop or reflect on understanding that is informal and frequently ungraded. Examples of WTL writing include reading response questions or short informal explanations. Academic writing is used to demonstrate knowledge that is formal and typically graded. Examples of academic writing include article summaries, research papers (e.g., CS pioneer biography), and formal explanations. Professional writing is used to apply knowledge in a professional context that is formal and typically graded. Examples of professional writing include technical papers and software development documents.

Dugan and Polanski use the taxonomy to guide Computer Science educators in overcoming reluctance to use writing in their teaching. They provide several examples of writing for each category and offer advice to mitigate Computer Science educators’ anxiety and resistance. Hoffman et al. use the taxonomy to offer a strategy to "bridge" writing across the taxonomy from WTL to Academic and Professional writing. They present three writing assignments based on the strategy. More recently, Giangrande [3] reviewed faculty objections to including writing in Computer Science courses and offered strategies to integrate communication skills including a variety of writing types across lower- and upper-level courses.

Curriculum standards, accrediting agencies, and universities acknowledge and support the importance of writing for Computer Science students. Dansdill et al. [2] found support for the importance of writing among Computer Science educators.

* Copyright is held by the author/owner.
However, the papers cited above, and many others like them indicate that there is still general resistance. Other strategies need to be explored to overcome the resistance.

An interview study with Computer Science educators at a variety of schools provide new insights into the types of writing that are used in Computer Science education and provide an opportunity to update the taxonomy developed by Dugan and Polanski [1] and Hoffman et al. [4]. In addition to the Type dimension, the updated taxonomy includes an Audience dimension. Audience emerged from the interviews as an important characteristic of writing in Computer Science education. Computer scientists write for audiences ranging from a machine (programming) to non-technical (customers). Table 1 shows the updated taxonomy with examples for each Type-Audience category.

<table>
<thead>
<tr>
<th>Audience</th>
<th>Type</th>
<th>Machine</th>
<th>Technical</th>
<th>Non-Technical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Writing to Learn</td>
<td>Code to test concept (i.e.,</td>
<td></td>
<td>Develop pseudocode</td>
<td>Short explanation of test code to customer</td>
</tr>
<tr>
<td>Develop and</td>
<td>learn)</td>
<td></td>
<td>Reading response</td>
<td>Creative exercise</td>
</tr>
<tr>
<td>reflect on</td>
<td></td>
<td></td>
<td>journal</td>
<td></td>
</tr>
<tr>
<td>understanding</td>
<td></td>
<td></td>
<td>Short explanation</td>
<td></td>
</tr>
<tr>
<td>Academic -</td>
<td>Code for a course</td>
<td></td>
<td>Research paper</td>
<td>Final project report</td>
</tr>
<tr>
<td>Demonstrate</td>
<td>assignment</td>
<td></td>
<td>Proof writing</td>
<td></td>
</tr>
<tr>
<td>knowledge</td>
<td></td>
<td></td>
<td>Code comments</td>
<td></td>
</tr>
<tr>
<td>Professional -</td>
<td>Code for a software product</td>
<td></td>
<td>Technical paper</td>
<td>Customer specs</td>
</tr>
<tr>
<td>Apply knowledge</td>
<td></td>
<td></td>
<td>Design specs</td>
<td>User manual</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Bug report</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code comments</td>
<td></td>
</tr>
</tbody>
</table>

The updated taxonomy includes types of writing that are native to Computer Science education and that may not be widely accepted as writing outside the discipline. For example, program code and program comments are considered writing by many interview participants. In this case, the audience is a machine. The updated taxonomy retains WTL type writing that is used to developing understanding. WTL writing for a machine captures short programming assignments where students learn by developing and testing short sections of code. The updated taxonomy captures more traditional forms of writing that are typically used to fulfill writing requirements.
The updated taxonomy offers new opportunities for Computer Science educators to use writing that is native to the discipline and link it to other forms of writing. For example, a linked WTL assignment could start with testing a concept by coding (WTL-Machine) that leads to writing pseudocode (WTL-Technical) and culminates in a short written explanation for grammar school students (WTL-Non-Technical). Each "translation" reinforces understanding. This type of assignment can overcome resistance by capitalizing on writing that is native and familiar to Computer Science educators.

REFERENCES


Growing a community of students in a commuter college is a difficult task. Students at commuter schools are disadvantaged with respect to resident students in many ways. Many resident colleges have dormitory floors or sections which group students of various disciplines together. Meeting after class to work on class projects is easier when your classmate lives within walking distance of your dorm room. Resident students don't have the financial and societal pressures commuter students have that come with the responsibilities of living at home. Gersting and Young note that student computer science campus organizations, such as chapters of the ACM or IEEE and even more informal computer science clubs, offer students a multitude of social, professional, and learning opportunities. [2] At the College of Staten Island (CSI), we have leveraged our student organization's Computer Science Club to create an environment that helps students form a computer science community outside of the classroom.

One of the most important contributions to the creation of a CS community at CSI is the allocation of a permanent space for the computer science club. The need for space stemmed from a club project to create and maintain a web server for all student clubs. Club project participants needed an accessible space to house these servers. The college required that the servers sit outside the school's network, "just in case..." The first club room was a faculty office sized space, equipped with web servers, college networked computers and some desk space. Because of concerns about too many students "hanging out" in such a small space, the space allocated to the club has grown into a classroom sized space, that has more student computers, reference books, tutoring space, video game equipment, Lego® robots, and desk space. Importantly, students have open access to the...
club room. Students use the room as a place to "hang out" between classes. What is also significant is that the club room is located in the middle of our faculty offices, offering students many opportunities to interact with faculty outside of the classroom. Many club members were able to find paid research work because of these interactions.

The club maintains its own web server and creates its own web site. The club's web site has had many iterations. Each new generation of students feels that they can improve and enhance and at times redo, the previous generation's work. Past club web masters have parlayed their experience on the club's web site into "real" web related jobs.

Besides maintaining a web server, the club offers a number of ongoing computer science projects for members to participate in. One such project is the "Autonomous Robotic Barbie Jeep". Based on a project first proposed by Zachary Dodds [1], students build an autonomous robotic vehicle based on a PowerWheels platform. Students are involved in many different aspects of this project. Some students are more hardware oriented and became involved in engineering the Jeep to drive autonomously. Others were involved in programming the Jeep. One student even decided to write a simulator using "Blender" a graphics program with which he was familiar. What is important to note is that students were not bounded by any formal class assignment but allowed total creative freedom. They were free to modify, experiment and enhance as long as those involved in the project agreed to the changes.

The club meets weekly with different activities planned for various meetings. Guest speakers, some former club alumni, have given talks. The club runs an annual computer science film festival which is open to all students on campus. Some members participate in service learning projects such as tutoring, sponsoring video game competitions for charity, and creating web pages for other clubs. Meetings serve a social function as well. Suffice it to say there are many opportunities for students to become involved.

The success of our computer science club has not only been beneficial to building a student community but has benefitted our department as well. Our feeling is that club members are more likely to continue and finish the computer science major as opposed to students who don't participate in the club. We are investigating whether the or not computer science club contributes to student retention. Our feeling is that this is most probably so.

The contribution of this poster will be to highlight the importance of building a student community via a computer science club. It will focus on how it benefits not only students but faculty as well. The author has been advisor to the computer science club for the past 11 years and has seen it grow into an integral part of CSI's computer science experience.

REFERENCES


The newly created Capstone course in Computer Information Systems (CIS) at UNH Manchester starting in spring 2011 offers students a faculty guided project. The course will expose students to the rigors of speech processing, giving them the opportunity to engage in solving real world problems, gaining invaluable experience along the way. By combining the Capstone course with an active research agenda in speech the course will be used as a pilot program for a National Science Foundation (NFS) Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics (TUES) grant proposal.

The purpose of the NFS TUES program is to improve the quality of science, technology, engineering and mathematics education at the undergraduate level. One way to excite students about science is to involve them in direct scientific discovery. Speech provides a rich area which encompasses all those aspects required by the TUES program. For a wide area of technological fields, speech can be applied to solve real world problems. Both its solution and applications are engineering problems that rely heavily on mathematics. To create a viable solution, good scientific principals need to be applied.

As the Capstone course is a one semester program, having only 16 weeks requires a staged approach. The inaugural project will focus on developing the platform and processes to enable active research. Specifically, students will learn with the Carnegie Mellon University Sphinx open source toolkit for speech recognition, building tools to use on a set of servers running Linux. The goal will be to develop baseline models and capture the techniques in creating them by crafting tools in the form of scripts and utility programs to ease the recreation of those models. Each subsequent course will expand on the previous, enabling more sophisticated research as the program matures.
Additionally, creating an experiment paradigm along with all tools required to generate a viable system is an important exercise that will expose students to all facets of the speech process. Not only will the aim be to capture the techniques used to create the models, but to track and document performance results of those models for future comparison. This is a critical element in any experimental research field.

As the Capstone course matures the aim is to partner with small high tech firms to solve real world speech projects. Think of it as a low cost alternative to an expensive speech consulting firm. Students would be key resources in solving problems towards specific project goals with expert guidance. This would allow partners, both external as well as within UNH a viable and cost effective solution to speech related problems.

Close ties with industry will enable placement opportunities for graduating seniors with high tech companies specializing in the field of speech. For non-seniors the possibility for summer internships in industry is also a very likely prospect. Along with that, a summer program as a follow up on the Capstone course will also be created to allow interested student a more in-depth and focused experience.

**OPTIONAL REFERENCES**

A WEB SERVICE TO CONNECT DATA ANALYSIS AND VISUALIZATION

FACULTY POSTER

Thomas Jyh-Cheng Liu
Department of Computer Science
New Jersey City University
Jersey City, NJ 07733
201-200-3140
tliu@njcu.edu

This poster presents the preliminary results of software that MSI (Minority Serving Institution) summer faculty-student team developed[1]. The software is a designated web service to facilitate cyber security research: It connects multiple data sources with analysis software and visualization tools for the Department of Homeland Security and the homeland security community. This project has demonstrated the effectiveness of a faculty-student team throughout the development of this software. The outcome has been successful thus far (the project is ongoing) as well as inspiring to the participating students; It has been a precious learning experience in computer science education as well as a practice in enterprise-level software development.

As an illustration of the preliminary outcome, this paper presents the analysis of a large dataset: email exchanges between Enron employees. The data visualization software adopted for this web project is Graphviz[2], an open source graphics project. The graphical user interface was designed using Adobe Flash Builder (Flex). The second phase of this study will employ XML, WSDL and SOAP to facilitate inter-application communication. The design will draw upon Mirage[3], an open source Java-based visualization tool. This framework will provide a "one-stop shop" to integrate data analysis and visualization tools for applications of homeland security community.

This project was initiated during summer research at the Command, Control and Interoperability Center for Advanced Data Analysis (CCICADA) of the Department of Security at Rutgers University. A faculty-student team was formed and mentored by researchers at CCICADA.

As of now, only the basic framework of this summer project has been developed and tested. One of major accomplishments has been the user interface, developed by writing Flex MXML. MXML is a rich Internet application markup language for building Web
applications that deploy on all major browsers, desktops and other platforms (it compiles to Flash actionscript). On the server side, the XAMPP package was downloaded and installed on both Linux and Windows XP.

The applications installed on the server-side include: igraph [4], GraphViz, and the industry-standard R [5] programming language for statistical computing and graphics. In addition, Perl and PHP are adopted as server-side scripting languages for implementation of the application backend. Adobe Flash player is the only requirement on the client-side.

Once a user provides an input data file and selects relevant parameters using given front end, the back-end Perl scripts are executed to conduct calculations (via other programs written in the R statistical language). In addition, data is communicated to Graphviz, which generates the intended graphs, histograms and relevant outputs. XMLHttpRequest objects, the generated graphs, diagrams and other outputs are then delivered to the front-end and displayed on the user's browser window.

The second phase of this project will be continued during the summer of 2011. The expected result includes a fully operational web service to deploy visualization tools like Mirage (a Java-based open source software developed in Bell Labs), and a model for similar tools and software to be deployed on the same framework.

ACKNOWLEDGEMENTS

Dr. William Pottenger, Research associate professor of Rutgers University and Director of transition, CCICADA in Department of Homeland Security.

Dr. Tin Ho, head of the Statistics and Learning Research Department of Bell labs in Alcatel-Lucent co.

Dr. Tami Carpenter, Associate Director of CCICADA in Department of Homeland Security and coordinator for MSI (Minority Serving Institution) faculty student team summer research program.

Monty Bains, a student member of the summer research team and a current graduate student of NJIT.

REFERENCES


USING PENNY AUCTIONS TO INTRODUCE NON-CS MAJORS TO MODELING AND SIMULATION*

FACULTY POSTER

David Toth
Department of Computer Science
Merrimack College
North Andover, MA 01845
978-837-3405
david.toth@merrimack.edu

Computation has become an accepted technique for scientists, mathematicians, economists, and others to learn about complex phenomena that might otherwise be impractical or impossible to study. Computational science uses modeling to create a simplified view of the phenomenon being studied and simulation to see how the phenomenon will behave over time under given conditions [1]. Computational science is used for pharmaceutical development, weather forecasting, predicting the behavior of the stock market, studying disasters like the oil spill that occurred in the summer of 2010, and many other things. A number of colleges and universities have majors in computational science [2, 3, 4, 5, 6, 7, 8].

We introduce the idea of modeling and simulation in our service course, Introduction to Information Technology, in order to give non-STEM students an idea of how computing can be useful for solving complex problems. One desired learning outcome of introducing modeling and simulation in this course is that the students will be able to take a new problem and analyze it to determine what they need to know in order to solve the problem. The required knowledge includes how a particular system works as well as the input parameters that affect the behavior of the system and the outcome of the simulation. The other desired learning outcome is that students learn how to explain how the system works in prose, incorporating the parameters, so that another person can read their description and understand how the system works. In the Introduction to Information Technology course, we have the students construct their simulations using Scratch because of the ease of converting English text into Scratch code [9]. The introduction to modeling and simulation also provides the transition in that course to the section on parallel and distributed computing, motivating the need for multi-core computers, clusters, supercomputers, and networking technologies.

* Copyright is held by the author/owner.
We are also planning on introducing a new course, Introduction to Computational Science, this fall for biology, chemistry, and physics majors. For the Introduction to Computational Science course, in addition to the learning outcomes for the modeling and simulation activities from our Introduction to Information Technology course, we also want the students to learn to build complex models and simulations with some of the standard tools.

We have used several different simple problems to motivate students in our Introduction to Information Technology course in the past [10]. We had mixed results and decided to develop a new problem that the students would find more interesting. Because many of the students taking the Introduction to Information Technology course are business majors, we decided to develop a problem on penny auctions. Penny auctions have become popular on the Internet in the last couple of years. These auctions differ from traditional auctions because it costs some amount of money to make a bid on an item. In addition to this, bids increase by one cent. Thus, the auctions are able to sell an item at a fraction of its retail cost and still make a profit, while the winning bidder is able to purchase the item at a fraction of its cost. We believe that this problem will also work well for our new Introduction to Computational Science course because of the simplicity of the problem and the ability to expand the problem and make it much more complex.

In this poster, we present an explanation of how penny auctions work and discuss how the company running the auction and the participants in the auction view the financial ramifications of running and participating in the auctions respectively. We also discuss the parameters that are involved in modeling the auctions so an accurate simulation can be created and present a simple simulator that we expect our students to be able to create. We will use this example in our courses next year, gauging interest with an anonymous survey and assessing the effectiveness with a quiz in Introduction to Information Technology and a project in Introduction to Computational Science.

REFERENCES


The purpose of this project was to use the McCabe IQ tool to analyze source code complexity of the Light-up Puzzle program. The McCabe IQ tool is software used in quality assurance for code analysis. It can allow people to determine if any parts of the source code are unreliable or unmaintainable without the user having to look at the source code. This can save both time and money as well as improve the quality of software code in businesses. Basically what the tool does is convert the source code into a graph, which it then uses to analyze the source code. The features of the tool that were used in this project are the Battlemap, System Complexity metrics, method flowgraphs, scatter diagrams, Halstead metrics, Class metrics, and Object-oriented metrics.

The McCabe IQ tool provides two main benefits to students. The first benefit is that students can learn to debug large volumes of code that is not their own. This is a task students rarely do in school, but may have to do in a work environment. The McCabe IQ tool can be used to quickly identify areas of code that have a high probability of errors. It is unlikely that the student would have to search through source code to find bugs. The second benefit is getting a better idea of what good programming is. Although good programming practices are taught, specific cut-off points for function size and complexity are usually not mentioned or followed. Several functions that are thought to be well-coded are really not. Students who use the McCabe IQ tool can get a better idea of this by programming to the standards of the McCabe IQ software.

The code chosen for analysis is my program of the puzzle "Light-Up". The program consists of roughly 4,000 lines of code with around 20 functions among six classes. It has a GUI for users to actually play the puzzle and a heuristic for attempting to automatically solve any puzzle. The main volumes of code are the GUI setup and event
handlers, the algorithms for manipulating the puzzle, and the heuristic that attempts to automatically solve any puzzle.

The source code was analyzed by the McCabe IQ tool and the data was interpreted to identify problems in the code. A new version of the program was coded to fix the problems found in the old version. Data for both versions were compared to each other to identify whether there was improvement between the two versions.

The Battlemap of the original version had 22 (63%) green modules, 1 (3%) yellow module, and 12 (34%) red modules, for a total of 35 modules. The Battlemap of the new version had 101 (81%) green modules, 1 (1%) yellow module, and 23 (18%) red modules, for a total of 125 modules. Although the number of red modules almost doubled from the old version to the new version, the new version was an improvement. The percentage of green modules increased from 63% in the old version to 81% in the new version and the percentage of red modules decreased from 34% to 18%. This indicates significant improvement in the source code between the old and new version. The reason why the numbers are deceptive is because 90 new less-complex modules were created by breaking up a few complex modules.

There was also significant improvement in the System Complexity metrics. The old version had a highest Cyclomatic complexity value of 281, highest Essential complexity value of 125, and highest Design complexity of 98. All three values are awful. The new version had a highest Cyclomatic complexity value of 31, highest Essential complexity value of 13, and highest Design complexity of 18. All three values are slightly bad. Considering that the worst values in the new version are only slightly high, that means the rest of the modules and the program as a whole is good. The problem occurred with the Class Metrics. The original version had no problems at the class level because there were not many functions. When 90 new functions were created from breaking up big functions, it introduced problems for the Class metrics. The RFC and WMC values for the "Dougrithm" class (the class with all the puzzle algorithms) jumped from 12 to 102.

There were two key discoveries made in this project. The first is that blindly using the McCabe IQ tool is not a good way to analyze the code. Users must know how to interpret the data correctly in order to gain the benefits of the tool. The Battlemap was a clear example where the Battlemap of the new version looked much worse than the Battlemap of the old version. At first glance people notice how there is many more red modules in the new version than the old version. They also notice the increase in the number of modules. The percentage of red/green modules is usually unnoticed. Therefore someone might think that the new version is worse when it is really much better. The second discovery is that changes to fix one problem could have unforeseen consequences on other metrics of the program. The changes made to fix the System Complexity metrics had a negative effect on the Class metrics. When there were only a few functions there was no problem. When those functions were broken down into several new functions, the communication among functions dramatically increased. For future work the functions of the "Dougrithm" class could be grouped together by what task they carry out. Then all the function groups can be separated into different classes to fix the problem that was introduced at the class level.

I would like to acknowledge Tim Houle for contributing to the analysis.
REFERENCES

Case studies have been a subject of much study and research about their effectiveness in teaching and learning [1, 2, 3]. Although the use of case study teaching has become a proven and pervasive method of teaching about professional practice in such fields as business, law, and medicine, it is yet to be adopted in any significant way in the computing education. One of the reasons for the paucity in the use of the case-study approach is the lack of sufficient material for this purpose. For example, among the hundreds of case studies in science at the SUNY Buffalo web site [4] only three concern computer science or software engineering.

Digital Home [11, 12] is a comprehensive case study that can be used throughout a computing curriculum. The case study material include complete software development artifacts as well as teaching case modules to be used to teach different topics (i.e., software inspections, object oriented design and implementation…) throughout a computer science or software engineering curriculum.

There are many software textbooks that use case studies (as examples to illustrate concepts and techniques): [5] (several, including an Airline Reservation System and a Household Alarm System), [6] (Fireworks Factory), [7] (Picadilly Television and Ariane-5), [8] (SafeHome). Additionally, there has been an increased effort in introducing case studies in computing courses as reported in the literature [9, 10].

Although these cases seem to serve a specific purpose (e.g., discussing planning, requirements, design, risk analysis, or construction issues by using simple examples that are quasi-realistic), they often lack the following:

- Realistic artifacts (often space does not allow providing a complete requirements or design document)
- Completeness (covers only a portion of the life-cycle, and not an end-to-end), with a focus on design and implementation
- Ability to decouple from the text and apply in ways not intended by the author
- Techniques for integration into course activities or into the curriculum as a whole
• A scenario format that motivate students' engagement in problem identification/solution.
• Guidance to the instructor on how to use the case study to teach a course topic or concept.

The DigitalHome case study aims to address these shortcomings by providing a complete set of artifacts associated with software development as well as providing case modules that can be used by faculty in teaching different subjects in a computing curriculum. Each case module represents a mini case study and is associated with a specific teaching subject (e.g. requirements inspection, object oriented design, testing…) and learning objectives. Case modules also include instructions and desired class setup to assist the instructor in teaching the session. A current list of the Digital Home artifacts, case modules, and exercises can be found at: http://www.softwarecasestudy.org/index.html

The goal of this work is to advance the understanding of case study based teaching and to introduce the audience to a comprehensive case study in software development. The goals of the poster are as follows:
1. Describe the background and effectiveness of case study techniques in professional education
2. Introduce the DigitalHome software development case study material
3. Advertise the use of case study material (artifacts and case modules) to faculty in a variety of computing programs and courses.
4. Invite audience of faculty to contribute to the development and assessment of the quality and utility of the case study material.

ACKNOWLEDGMENT

Initial work on the DigitalHome case study was funded as part of the NSF project: "The Network Community for Software Engineering Education" (SWENET) (NSF 0080502. The current work on the case study is funded through NSF's (DUE- 0941768) "Curriculum-wide Software Development Case Study".

REFERENCES


This poster presentation reports on classroom experience using the logic/math puzzles known by the generic name Mathdoku[1] for student programming projects in a Data Structures course. Using these puzzles encourages students to employ an object-oriented approach to the two major components of the project (1) a user or computer puzzle player component, and (2) a puzzle generation component. This project affords students the chance to develop a non-trivial application, to re-use data structures developed during the semester, and to leverage existing data structures from framework libraries. It also offers ample opportunity for customizations, so that no two projects should end up the same.

Students were provided with starter code in small sections. The first iteration simply presented an existing puzzle, read in from a file. The second iteration looked at possible ways to support user play, and looked ahead to computer play. Project development next concentrated on puzzle generation, in three separate steps: First, entirely manual creation, with the user specifying both the numbers for each square as well as the cage layouts and operations. Second, a semi-automatic creation mode in which the program generates the numbers for the squares and the user then specifies the cages. The third and final mode is a fully-automatic mode, in which the program completely generates the puzzle. We follow [2] in classifying our puzzle generation as implementing a Monte Carlo algorithm, since "a Monte Carlo algorithm is a randomized algorithm whose running time is deterministic, but whose output may be incorrect with a certain (typically small) probability." We allow that a puzzle is incorrect if it has more than one solution.

At this stage, development switched back to the task of computer-generated solutions. This is where the "methodology" took part, in that the game tree concept was employed, using either a standard depth-first or breadth-first search though the generated

* Copyright is held by the author/owner.
tree to find a solution. One of the students in the class did a very nice comparison of the number of nodes explored finding a solution via the two standard search methods.

The last step of the project was to create a third application that would spawn a process to execute either the puzzle-playing application or the puzzle-generation application. This gave students experience with the process concept, and using it in a program.

REFERENCES


MINING FORUMS FOR ASYNCHRONOUS LEARNING IN DISTANCE EDUCATION

FACULTY POSTER

Songmei Yu, Sofya Poger, Alberto LaCava
Department of Computer Information Systems
Felician College
262 South Main Street
Lodi, NJ 07644
201-559-6000
yus@felician.edu, pogers@felician.edu, lacavaa@felician.edu

I. INTRODUCTION TO AWK ALGORITHM

Distance Education is a process of delivery of instructional resources online. It is a flexible form of education where a student can study from home, work, or wherever else is convenient. Computer based teaching materials, online conferencing, discussion boards, e-mail, and other techniques are used in distance education. The information can be delivered synchronously and asynchronously. The example of synchronous method of delivery is Web conferencing where all participants are "present" online at the same time. The discussions boards, e-mail and recorded video are examples of asynchronous method of delivery, where participants access course material at their own schedule [1].

Online discussions and forums allow students and instructors to exchange the ideas, research and information. The aim of this research is to apply data mining techniques to discussions boards and forums to measure the student participation and performance. In this research, we plan to introduce and implement the AWK (Author Weighted Keywords) Algorithm as a tool to measure the student performance and learning capabilities by monitoring their participation in distance learning forums, extract discussion topics for later comers and help instructors to guide the online discussions.

The AWK algorithm is based on the following Web Mining Techniques:

• Data/Information extraction
• Web information integration and schema matching
• Opinion extraction from online sources
• Knowledge synthesis
• Segmenting Web pages and detecting noise
An alternative to web searching can be applied to students participating in online courses using a course management system (CMS). A CMS uses a database program to store the information of the course, including forum entries. As an example, Moodle [2] frequently uses a MySQL database engine and stores the information in a database file. This file and the files in the system can then be searched using the database engine, and the AWK algorithm can be applied directly to the database contents.

By using data mining techniques we extract the keywords from each thread or topic. The weight for each student is calculated based on number of threads initiated, number of posted comments and responses, and content of posted comments and responses. The content of posted comments and responses is defined by the several factors of extracted keywords, such as the distance from keywords to other words in a post. The student performance is based on the weight - the higher the weight, the better performance. The AWK algorithm helps to understand the dynamic behavior of students in the Web systems and improve the teaching process in more efficient and effective ways, compared to other Web mining techniques in current distance learning field [3].

II. OUR FUTURE PLAN

- Build source data from the real online materials of Computer Information Systems courses from a live distance learning forum (e.g. under Moodle)
- Construct tree structure by simulating distance learning forum scheme
- Extract keywords using text mining techniques and natural language processing
- Apply Author Weighted Keywords (AWK) Algorithms
- Compare manual calculations of the student performance with the result obtained using AWK algorithms.

III. POSTER CONTENT

The poster will provide an overview the Web Mining techniques, challenging issues in mining distance learning forums, Author Weighted Keywords Algorithm, and our implementation plan.

REFERENCES


[2] A description of Moodle is given in the community page http://moodle.org/about/

## INDEX OF AUTHORS

<table>
<thead>
<tr>
<th>Author</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abunawass, A</td>
<td>156</td>
</tr>
<tr>
<td>Bailie, F</td>
<td>156</td>
</tr>
<tr>
<td>Baird, B</td>
<td>165</td>
</tr>
<tr>
<td>Banerjee, S</td>
<td>168</td>
</tr>
<tr>
<td>Barneva, R</td>
<td>170</td>
</tr>
<tr>
<td>Blake, J</td>
<td>124</td>
</tr>
<tr>
<td>Braught, G</td>
<td>51, 173</td>
</tr>
<tr>
<td>Brinkov, B</td>
<td>170</td>
</tr>
<tr>
<td>Brusilovsky, P</td>
<td>160</td>
</tr>
<tr>
<td>Cassel, L</td>
<td>160</td>
</tr>
<tr>
<td>Chua, M</td>
<td>10</td>
</tr>
<tr>
<td>Decker, A</td>
<td>7</td>
</tr>
<tr>
<td>Defoe, D</td>
<td>163</td>
</tr>
<tr>
<td>Delcambre, L</td>
<td>160</td>
</tr>
<tr>
<td>Dickson, P</td>
<td>136</td>
</tr>
<tr>
<td>Dziallas, S</td>
<td>10</td>
</tr>
<tr>
<td>Eckmann, M</td>
<td>117</td>
</tr>
<tr>
<td>Fischbach, A</td>
<td>53</td>
</tr>
<tr>
<td>Fischer, A</td>
<td>61</td>
</tr>
<tr>
<td>Fitzgerald, S</td>
<td>133</td>
</tr>
<tr>
<td>Fox, E</td>
<td>160</td>
</tr>
<tr>
<td>Furuta, R</td>
<td>160</td>
</tr>
<tr>
<td>Garcia, D</td>
<td>160</td>
</tr>
<tr>
<td>Goldschmidt, D</td>
<td>78</td>
</tr>
<tr>
<td>Grammer, K</td>
<td>44</td>
</tr>
<tr>
<td>Gudmundsen, D</td>
<td>23, 107</td>
</tr>
<tr>
<td>Hanebutte, N</td>
<td>101</td>
</tr>
<tr>
<td>Harrison, W</td>
<td>101</td>
</tr>
<tr>
<td>Hilburn, T</td>
<td>13, 190</td>
</tr>
<tr>
<td>Hislop, G</td>
<td>10, 160</td>
</tr>
<tr>
<td>Hoffman, M</td>
<td>175</td>
</tr>
<tr>
<td>Imberman, S</td>
<td>178</td>
</tr>
<tr>
<td>Ivanov, L</td>
<td>142</td>
</tr>
<tr>
<td>Izmirl, O</td>
<td>165</td>
</tr>
<tr>
<td>Jackson, S</td>
<td>1</td>
</tr>
<tr>
<td>Joiner, D</td>
<td>71</td>
</tr>
<tr>
<td>Jonas, M</td>
<td>180</td>
</tr>
<tr>
<td>Kurkovsky, S</td>
<td>163</td>
</tr>
<tr>
<td>Kussmaul, C</td>
<td>10, 135</td>
</tr>
<tr>
<td>LaCava, A</td>
<td>195</td>
</tr>
<tr>
<td>LeBlanc, M</td>
<td>130</td>
</tr>
<tr>
<td>Liu, T</td>
<td>182</td>
</tr>
<tr>
<td>MacDonald, I</td>
<td>78</td>
</tr>
<tr>
<td>MacKellar, B</td>
<td>93</td>
</tr>
<tr>
<td>Madigan, E</td>
<td>64</td>
</tr>
<tr>
<td>Meinke, J</td>
<td>X</td>
</tr>
<tr>
<td>Milonovich, B</td>
<td>78</td>
</tr>
<tr>
<td>Moll, R</td>
<td>55</td>
</tr>
<tr>
<td>Morreale, P</td>
<td>71</td>
</tr>
<tr>
<td>O'Rourke, J</td>
<td>78</td>
</tr>
<tr>
<td>Olivieri, L</td>
<td>23, 107</td>
</tr>
<tr>
<td>Peluso, E</td>
<td>12</td>
</tr>
<tr>
<td>Peters, S</td>
<td>6</td>
</tr>
<tr>
<td>Petrovic, S</td>
<td>156</td>
</tr>
<tr>
<td>Poger, S</td>
<td>195</td>
</tr>
<tr>
<td>Prey, J</td>
<td>5</td>
</tr>
<tr>
<td>Proulx, V</td>
<td>149</td>
</tr>
<tr>
<td>Riabov, V</td>
<td>86</td>
</tr>
<tr>
<td>Rilett, D</td>
<td>130</td>
</tr>
<tr>
<td>Russo, J</td>
<td>130</td>
</tr>
<tr>
<td>Salamah, S</td>
<td>13, 190</td>
</tr>
<tr>
<td>Sarawagi, N</td>
<td>23, 107</td>
</tr>
<tr>
<td>Selent, D</td>
<td>187</td>
</tr>
<tr>
<td>Stevenson, C</td>
<td>37</td>
</tr>
<tr>
<td>Stewart-Gardiner, C</td>
<td>16</td>
</tr>
<tr>
<td>Stolerman, J</td>
<td>44</td>
</tr>
<tr>
<td>Stone, J</td>
<td>64</td>
</tr>
<tr>
<td>Teresco, J</td>
<td>58</td>
</tr>
<tr>
<td>Torrey, L</td>
<td>110</td>
</tr>
<tr>
<td>Toth, D</td>
<td>184</td>
</tr>
<tr>
<td>Towhidnejad, M</td>
<td>13, 190</td>
</tr>
<tr>
<td>Trees, F</td>
<td>7</td>
</tr>
<tr>
<td>Werner, M</td>
<td>130</td>
</tr>
<tr>
<td>Whitfield, D</td>
<td>156</td>
</tr>
<tr>
<td>Wilkens, L</td>
<td>193</td>
</tr>
<tr>
<td>Wu, H</td>
<td>130</td>
</tr>
<tr>
<td>Wurst, K</td>
<td>1</td>
</tr>
<tr>
<td>Yi, B</td>
<td>44</td>
</tr>
<tr>
<td>Yu, S</td>
<td>195</td>
</tr>
</tbody>
</table>