The Journal of Computing Sciences in Colleges

Volume 25, Number 3 **January 2010**

The Journal of Computing Sciences in Colleges

Papers of the Twenty-Fifth Annual CCSC Eastern Conference

October 30-31, 2009
Villanova University
Villanova, Pennsylvania

John Meinke, Editor
UMUC Europe

George Benjamin, Associate Editor
Muhlenberg College

Susan T. Dean, Associate Editor
UMUC Europe

Volume 25, Number 3 **January 2010**

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.

TABLE OF CONTENTS

THE CONSORTIUM FOR COMPUTING SCIENCES IN COLLEGES
BOARD OF DIRECTORS vii

CCSC NATIONAL PARTNERS viii

FOREWORD viii
John Meinke, UMUC Europe

WELCOME — CCSC EASTERN 2009 1
Don Goelman, Villanova University, John Lewis, Virginia Tech

CCSC: EASTERN REGION STEERING COMMITTEE 2

CCSC : EASTERN 2009 CONFERENCE COMMITTEE 2

REVIEWERS — 2009 CCSC EASTERN CONFERENCE 3

ESCCC / CCSC EASTERN CONFERENCES AND HOST INSTITUTIONS 4

AFTER THE OPEN SOURCE REVOLUTION — KEYNOTE ADDRESS 5
Eric Raymond

GAME2LEARN: CREATING AND EVALUATING EDUCATIONAL GAMES
FOR COMPUTING — SIGCSE INVITED PRESENTATION 5
Tiffany Barnes, University of North Carolina – Charlotte

THE MAGIC OF COMPUTER SCIENCE — BANQUET ADDRESS 5
Tom Way, Villanova University

COMPUTER SCIENCE UNPLUGGED — CONFERENCE WORKSHOP 6
Thomas Cortina, Carnegie Mellon University

PUZZLE-BASED LEARNING — CONFERENCE WORKSHOP 7
Raja Sooriamurthi, Carnegie Mellon University, Nickolas Falkner, Zbigniew
Michalewicz, University of Adelaide

TEACHING A FEMALE FRIENDLY RPRCC (REAL PROJECTS FOR REAL CLIENTS COURSE) INTRODUCTION TO SOFTWARE DEVELOPMENT AT THE MIDDLE SCHOOL, HIGH SCHOOL OR FRESHMAN COLLAGE LEVEL — CONFERENCE WORKSHOP 8
David Klappholz, Stevens Institute of Technology

CERTIFICATION AND STANDARDS FOR COMPUTING EDUCATION IN PENNSYLVANIA — PANEL DISCUSSION 9
Jean Griffin (Moderator), University of Pennsylvania, John P. Dougherty, Haverford College, Tammy R. Pirmann, School District of Springfield , Rita Powell, University of Pennsylvania

COMMUNITY EMPOWERMENT AND SERVICE LEARNING PRACTICES THROUGH COMPUTER SCIENCE CURRICULA OF A MAJOR METROPOLITAN UNIVERSITY — TUTORIAL PRESENTATION 10
James Lawler, Jean Coppola, Susan Feather-Gannon, Jonathan Hill, Richard Kline, Pauline Mosley, Andrea Taylor, Pace University

SELECTING AND USING VIRTUALIZATION SOLUTIONS – OUR EXPERIENCES WITH VMWARE AND VIRTUALBOX 11
Peng Li, East Carolina University

USING SUBVERSION AS AN AID IN EVALUATING INDIVIDUALS WORKING ON A GROUP CODING PROJECT 18
Curt Jones, Bloomsburg University

REAL-TIME COLLABORATION TOOLS FOR DIGITAL INK 24
Steven Lindell, Haverford College

IMPROVING BELIEVABILITY OF SIMULATED CHARACTERS 32
Jere Miles, Wilkes Community College, Rahman Tashakkori, Appalachian State University

BEYOND “NOT-INVENTED-HERE”: DEVELOPMENT ENVIRONMENTS FOR A MULTIMEDIA COMPUTATION COURSE 40
Stephen P. Carl, Sewanee: The University of the South

EBAY, ITUNES, AND PROPOSITIONAL LOGIC: COMPARING EXPRESSIVENESS OF DIFFERENT QUERY LANGUAGES 47
Ian Barland, Radford University

INCORPORATING ETHICS INTO THE COMPUTER SCIENCE CURRICULUM: MULTIPLE PERSPECTIVES — PANEL DISCUSSION 53
Frances Bailie (Moderator), Keitha Murray, Smiljana Petrovic, Iona College, Deborah Whitfield, Slippery Rock University

METHOD ASSUMPTIONS IN OBJECT-ORIENTED PROGRAMMING 54
Robert Noonan, College of William and Mary

REVITALIZING CS HARDWARE CURRICULA: OBJECT ORIENTED HARDWARE DESIGN 60
Ganesh R. Baliga, John Robinson, Leigh Weiss, Rowan University

THE ESSENCE OF OBJECT ORIENTATION FOR CS0: CONCEPTS WITHOUT CODE 67
Raja Sooriamurthi, Carnegie Mellon University

PERSPECTIVES CONCERNING THE UTILIZATION OF SERVICE LEARNING PROJECTS FOR A COMPUTER SCIENCE COURSE 75
Richard A. Scorce, St. John's University

MOTIVATING PROGRAMMERS VIA AN ONLINE COMMUNITY 82
Poul Henriksen, Michael Kölling, University of Kent, Davin McCall, Deakin University

CYBER-POLITICS: DEVELOPING AN INTERDISCIPLINARY LEARNING COMMUNITY IN AN ELECTION YEAR 94
Michael Ruth, Adrian Ionescu, Wagner College

IMPLEMENTING A BACCALAUREATE PROGRAM IN COMPUTER FORENSICS 101
Jigang Liu, Metropolitan State University

MAKING SERVICE ORIENTED ARCHITECTURE RELEVANT USING A MULTIDISCIPLINARY APPROACH 110
Thomas Way, Vijay Gehlot, Villanova University

GREENFOOT – INTRODUCTION TO JAVA WITH GAMES AND SIMULATIONS — TUTORIAL PRESENTATION 117
Michael Kölling, University of Kent

INTRODUCING MULTI-CORE PROGRAMMINGINTO THE LOWER-LEVEL CURRICULUM: AN INCREMENTAL APPROACH — TUTORIAL PRESENTATION 118
Timothy J. McGuire, Sam Houston State University

A STATE DIAGRAM CREATION AND CODE GENERATION TOOL FOR ROBOT PROGRAMMING 120
Sambit Bhattacharya, Bogdan Denny Czejdo, Fayetteville State University

ROBOTS IN THE CLASSROOM … AND THE DORM ROOM 128
Jennifer S. Kay, Rowan University

INDUSTRIAL ROBOTIC GAME PLAYING: AN AI COURSE 134
Sebastian van Delden, University of South Carolina Upstate

INTERNATIONAL COMPUTING ISSUES AS A FRESHMAN SEMINAR . . . 143
Christopher A. Healy, Furman University

USING VIDEO TO EXPLORE PROGRAMMING THINKING AMONG UNDERGRADUATE STUDENTS 149
Carol Wellington, Rebecca Ward, Shippensburg University

FACTORS IMPACTING STUDENT PERCEPTIONS OF COMPUTING AND CIS MAJORS 156
Jeffrey A. Stone, Pennsylvania State University Schuylkill Haven, David P. Kitlan, Pennsylvania State University Middletown

USING PEER LED TEAM LEARNING TO ASSIST IN RETENTION IN COMPUTER SCIENCE CLASSES 164
Carolee Stewart-Gardiner, Kean University

STUDENT EVALUATION IN MONITORED TEAM PROJECTS 172
Joo Tan, Kutztown University of Pennsylvania

INTRODUCTION TO CRYPTOGRAPHY — TUTORIAL PRESENTATION . . 180
Seth D. Bergmann, Rowan University

COOPERATIVE LEARNING FOR CS1 AND BEYOND: MAKING IT WORK FOR YOU — CONFERENCE WORKSHOP 181
Leland Beck, Alexander Chizhik, San Diego State University

THE ANIMATED DATABASE COURSEWARE (ADbC) — CONFERENCE WORKSHOP 182
Mario Guimaraes, Meg Murray, Kennesaw State University

TURNING A 14 WEEK NON-MAJOR CLASS INTO A 7 WEEK FAST FORWARD CLASS — CONFERENCE WORKSHOP 183
Barbara Zimmerman, Villanova University

POSTER SESSION 184

INDEX OF AUTHORS 185

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:

Myles McNally, President (2010), Webmaster, Professor of Computer Science, Alma College, 614 W. Superior St., Alma, MI 48801, (989) 463-7163 (O), (989) 463-7079 (fax), mcnally@alma.edu.

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

Jim Aman, Membership Chair (2010), Assoc. Professor, Computer Science, Saint Xavier University, Chicago, IL 60655, (773) 298-3454 (O), (614) 492-1306 (H), (630) 728-2949 (cell), aman@sxu.edu.

Bill Myers, Treasurer (2011), Dept. of Computer Studies, Belmont Abbey College, Belmont, NC 28012-1802, (704) 461-6823, (704) 461-5051, (Fax), myers@crusader.bac.edu

John Meinke, Publications Chair, (2012), Collegiate Associate Professor, UMUC – Europe, US Post: CMR 420, Box 3668, APO AE 09063; (H) Werderstr 8, D-68723 Oftersheim, Germany, 011-49-6202-5 77 79 16 (H), 011 - 49-6221- 31 58 71 (fax), meinkej@acm.org.

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

Elizabeth S. Adams, Eastern Representative (2011), James Madison University - Mail Stop 4103, CISAT - Department of Computer Science, Harrisonburg VA 22807, 540-568-1667 (O), 540-568-2745(fax), e-mail: adamses@jmu.edu.

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

Scott Sigman, Central Plains Representative (2011), Associate Professor of Computer Science, Drury University, Springfield, MO 65802, (417) 873-6831, ssigman@drury.edu

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

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

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

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), Department of Computer Science, Sam Houston State University, Huntsville, Texas 77341-2090, (936)294-1571, mcguire@shsu.edu

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

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

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

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

Susan Dean, Associate Editor, Collegiate Professor, UMUC – Europe,Europe, US Post: CMR 420, Box 3669, APO AE 09063; (H) Werderstr 8, D-68723 Oftersheim, Germany. 011-49-6202-5 77 82 14, sdean@faculty.ed.umuc.edu.

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

Paul D. Wiedemeier, National Partners Program Coordinator, The University of Louisiana at Monroe, Computer Science and Computer Information Systems, Monroe, LA 71209, (318) 342-1856, wiedemeier@ulm.edu.

CCSC NATIONAL PARTNERS

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

Microsoft Corporation

Cengage Learning

The Shodor Education Foundation, Inc.

John Wiley and Sons

Turing's Craft

RidgeSoft

FOREWORD

A Silver Jubilee! Wow, Eastern celebrates a quarter century! I can remember vividly those first planning meetings in Scranton back in the early 1980's. It's somewhat hard to believe that more than a quarter century has passed since those first meetings. As I look at the list of previous conferences I am impressed at the geographical range. I am also impressed with the geographical range represented by the presenters for this conference. While we consider ourselves a regional conference participation is international. When I look at this year's program I know that it will be a productive couple days for the participants.

It's not too early to already be thinking about next year's conference. Planning is already taking place for October 2010 hosted by Juniata College in Huntingdon, Pennsylvania. The call for participation will be out shortly The success of any conference is totally dependent on the quality of the presentations. Be a part of that quality by starting work on your presentation now.

Many thanks to those who have contributed to this issue of the *Journal*. It starts with the steering committee and the conference committee. The associate editors are gems to work with, and, of course, many thanks to the folks at Montrose Publishing who assemble the final product.

Enjoy the conference.

John Meinke
UMUC Europe
CCSC Publications Chair

WELCOME — CCSC EASTERN 2009

Welcome to CCSCE 2009, the 25th Eastern Regional Conference, and to Villanova University! This year's presenters have submitted an amazing array of high-quality papers, workshops, tutorials, panels, posters, lightning talks, and nifty ideas. There's a lot going on this year, including the highly-anticipated programming contest, and we hope you will make the most of it.

This year, twenty-three papers were accepted, resulting in an acceptance rate of 58%. Each paper was reviewed by 3 or 4 reviewers. Twelve workshops, panels, and tutorials were also accepted for inclusion in this year's program.

Eric Raymond, our keynote speaker on Friday, is a renowned author and leader of the open-source revolution. Appropriate for our conference theme (Treats and Some Tricks in Computer Science Education), our banquet speaker is Tom Way, who will show us how magic tricks can be used to illustrate computing concepts. And the rest of the program is packed with other great events and presentations!

Obviously, an undertaking like this conference is a team effort. In addition to this year's presenters, we want to thank the reviewers, session chairs, contest judges, and student assistants. Thanks also to the CCSC Board and the CCSCE Steering Committee. Special thanks go to Liz Adams, the Eastern Region Steering Committee Chair, George Benjamin, who worked so hard on these Proceedings, and to Sr. Jane Fritz, Liz Chang and Gary Gillard, chairs of the most recent CCSCE conferences. Their insight and advice were invaluable.

The CCSCE 2009 Conference Committee is a wonderful group of people. Thanks for making this such a productive and enjoyable process! We appreciate the Department of Computing Sciences at Villanova University for providing such a great venue. We're also grateful to ACM SIGCSE for their support of Tiffany Barnes' invited talk, and to ACM UPE for their contribution to the student prizes. Finally, we are extremely grateful to our National Partners, vendors, and sponsors. Please visit them and thank them for their generosity!

We hope you enjoy this year's conference!

Don Goelman (Villanova University) and *John Lewis* (Virginia Tech)
CCSCE 2009 Conference Co-Chairs

CCSC: EASTERN REGION STEERING COMMITTEE

Elizabeth Adams . . . . . James Madison University, Harrisonburg, VA
Steven Andrianoff . . . . . St. Bonaventure University, St. Bonaventure, NY
Karen Anewalt . . . . . University of Mary Washington, Fredericksburg, VA
Jack Beidler . . . . . University of Scranton, Scranton, PA
George Benjamin . . . . . Muhlenberg College, Allentown, PA
Elizabeth Chang . . . . . Hood College, Frederick, PA
Dorothy Deremer . . . . . Montclair State University, Montclair, NJ
Sister Jane Fritz . . . . . St. Joseph's College, Patchogue, NY
Amruth Kumar . . . . . Ramapo College, Mahwah, NJ
John Meinke . . . . . UMUC Europe, Heidelberg, D
Jennifer Polack-Wahl . . . . . University of Mary Washington, Fredericksburg, VA
Onkar Sharma . . . . . Marist College, Poughkeepsie, NY
James R. Sidbury . . . . . University of Scranton, Scranton, PA
Pat Woodworth . . . . . Ithaca College, Ithaca, NY

CCSC : EASTERN 2009 CONFERENCE COMMITTEE

Don Goelman, Co-Chair . . . . . Villanova University, Villanova, PA
John Lewis, Co-Chair . . . . . Virginia Polytechnic Institute and State University, Blacksburg, VA
Pete DePasquale, Papers . . . . . The College of New Jersey, Ewing, NJ
Amruth Kumar, Papers . . . . . Ramapo College, Mahwah, NJ
Joe Chase, Workshops, Panels, Tutorials . . . . . Radford University, Radford, VA
Ian Barland, Workshops, Panels, Tutorials . . . . . Radford University, Radford, VA
Leigh Ann Sudol, Nifty Ideas & Lightning Talks . . . . . Carnegie Mellon University, Pittsburgh, PA
Thomas Cortina, Programming Competition . . . . . Carnegie Mellon University, Pittsburgh, PA
Najib Nadi, Programming Competition . . . . . Villanova University, Villanova, PA
Sara Miner More, Programming Competition . McDaniel College, Westminster, MD
Dave Hovemeyer, Posters . . . . . York College of Pennsylvania, York, PA
Vijay Gehlot, Registration . . . . . Villanova University, Villanova, PA
Tom Way, Local Arrangements . . . . . Villanova University, Villanova, PA
Paula Matuszek, Local Arrangements . . . . . Matuszek Consulting and Villanova University, Villanova, PA
Frank Klassner, Vendors . . . . . Villanova University, Villanova, PA
Daniel Joyce, Web Site . . . . . Villanova University, Villanova, PA
Victoria Suwardiman, Web Site . . . . . Villanova University, Villanova, PA
JD Dougherty, K-12 . . . . . Haverford College, Haverford, PA
Jerry Kruse, 2010 Conference Co-Chair . . . . . Juniata College, Huntingdon, PA
John Wright, 2010 Conference Co-Chair . . . . . Juniata College, Huntingdon, PA

REVIEWERS — 2009 CCSC EASTERN CONFERENCE

Elizabeth Adams . . . . . James Madison University
Sambit Bhattacharya . . . . . Fayetteville State University
Stephen Bloch . . . . . Adelphi University
Steven Bogaerts . . . . . Wittenberg University
Esmail Bonakdarian . . . . . Franklin University
John Boon . . . . . Hood College
Tom Briggs . . . . . Shippensburg University
David Bunde . . . . . Knox College
Melanie Butler . . . . . Mount St. Mary's University
Mohsen Chitsaz . . . . . Frostburg State University
Lawrence D'Antonio . . . . . Ramapo College of New Jersey
Mary Jo Davidson . . . . . DePaul University
Molisa Derk . . . . . Dickinson State University
Manpreet Dhillon . . . . . George Washington University
Priscilla Dodds . . . . . Georgia Perimeter College
Aijuan Dong . . . . . Hood College
Peter Drexel . . . . . Plymouth State University
Barbara Edington . . . . . St. Francis College
Frank Friedman . . . . . Temple University
Alessio Gaspar . . . . . University of South Florida Polytechnic
John Gray . . . . . University of Hartford
Phillip Heeler . . . . . Northwest Missouri State University
Catherine Helwig . . . . . Villanova University
Dwight House . . . . . Fayetteville State University
Daniel Joyce . . . . . Villanova University
Tom Kelliher . . . . . Goucher College
James Lawler . . . . . Pace University
Peter Liu . . . . . Seneca College
Xinlian Liu . . . . . Hood College
Kate Lockwood . . . . . Northwestern University
Jeff Martens . . . . . University of Maryland Baltimore County
José Carlos Metrôlho . . . . . Instituto Polytéchinico de Castelo Branco
Ruth Michael . . . . . Wagner College
Robert Neufeld . . . . . McPherson College
Jeff Parker . . . . . Harvard Extension School
Jeffrey Popyack . . . . . Drexel University
Veeramuthu Rajaravivarma . . . . . Farmingdale State Collge, SUNY
Ahmed Salem . . . . . Hood College
Ian Sanders . . . . . University of the Witwatersrand
Richard Scorce . . . . . St. John's University
Barry Soroka . . . . . Cal Poly Pomona

ESCCC / CCSC EASTERN CONFERENCES AND HOST INSTITUTIONS

1. ESCCC 1985 . . . . . University of Scranton, Scranton, PA
2. ESCCC 1986 . . . . . University of Scranton, Scranton, PA
3. ESCCC 1987 . . . . . Marist College, Poughkeepsie, NY
4. ESCCC 1988 . . . . . Eastern College, St. David's, PA
5. ESCCC 1989 . . . . . Ithaca College, Ithaca, NY
6. ESCCC 1990 . . . . . Allentown College of St. Francis de Sales, Center Valley, PA
7. ESCCC 1991 . . . . . Marymount College, Tarrytown, NY
8. ESCCC 1992 . . . . . Muhlenberg College, Allentown, PA
9. ESCCC 1993 . . . . . Monmouth College, West Long Branch, NJ
10. ESCCC 1994 . . . . . St. John Fisher College, Rochester, NY
11. ESCCC 1995 . . . . . Iona College, New Rochelle, NY
12. ESCCC 1996 . . . . . Marywood College, Scranton, PA
13. ESCCC 1997 . . . . . Richard Stockton College of New Jersey, Pomona, NJ
14. ESCCC 1998 . . . . . Marist College, Poughkeepsie, NY
15. ESCCC 1999 . . . . . St. Bonaventure University, St. Bonaventure, NY
16. ESCCC 2000 . . . . . University of Scranton, Scranton, PA
17. ESCCC - CCSCE 2001 . . . . . Shepherd College, Shepherdstown, WV
18. CCSCE 2002 . . . . . Bloomsburg University, Bloomsburg, PA
19. CCSCE 2003 . . . . . Montclair State University, Upper Montclair, NJ
20. CCSCE 2004 . . . . . Loyola College in Maryland, Baltimore, MD
21. CCSCE 2005 . . . . . Iona College, New Rochelle, NY
22. CCSCE 2006 . . . . . University of Mary Washington, Fredericksburg, VA
23. CCSCE 2007 . . . . . St. Joseph's College, Patchogue, NY
24. CCSCE 2008 . . . . . Hood College, Frederick, MD
25. CCSCE 2009 . . . . . Villanova University, Villanova, PA

KEYNOTE ADDRESS

Friday, October 30, 2009

AFTER THE OPEN SOURCE REVOLUTION

Eric Raymond

Author and Troublemaker

SIGCSE INVITED PRESENTATION

Friday, October 30, 2009

GAME2LEARN: CREATING AND EVALUATING EDUCATIONAL GAMES FOR COMPUTING

Tiffany Barnes

Department of Computer Science

University of North Carolina – Charlotte

BANQUET ADDRESS

Friday, October 30, 2009

THE MAGIC OF COMPUTER SCIENCE

Tom Way

Department of Computing Sciences

Villanova University

.

COMPUTER SCIENCE UNPLUGGED*

CONFERENCE WORKSHOP

Dr. Thomas Cortina
Carnegie Mellon University

Computer Science Unplugged is a book containing a set of fun activities that teachers from K-12 (and beyond) can use to illustrate computer science principles without using a computer. This workshop will allow participants to learn how to run a number of the activities with additional discussion on variations that can be used and issues to be aware of when these are presented to students. Audience participation will be a major focus of this workshop. After a number of activities are demonstrated and discussed, participants will work together to create a new activity suitable for the Unplugged theme.

PUZZLE-BASED LEARNING*

CONFERENCE WORKSHOP

Raja Sooriamurthi
Carnegie Mellon University

Nickolas Falkner
University of Adelaide

Zbigniew Michalewicz
University of Adelaide

Puzzle-based learning (PBL) is a new and emerging model of teaching critical thinking and problem solving. In this interactive workshop we will examine a range of puzzles, brainteasers, and games. What general problem solving strategies can we learn from the way we solve these examples? A learning goal of PBL is to distill domain independent transferable heuristics for tackling problems. In the past year we have created and taught new courses on PBL in three countries under different academic settings. In this hands-on workshop we will introduce PBL and discuss our goals and experience.

TEACHING A FEMALE FRIENDLY RPRCC (REAL PROJECTS FOR REAL CLIENTS COURSE) INTRODUCTION TO SOFTWARE DEVELOPMENT AT THE MIDDLE SCHOOL, HIGH SCHOOL OR FRESHMAN COLLAGE LEVEL*

CONFERENCE WORKSHOP

David Klappholz
Stevens Institute of Technology

Middle/high-school teacher attendees will learn, hands-on, how to teach a Real Projects for Real Clients Course (RPRCC) in which students work in teams to perform the interpersonal-interaction-intensive activities involved in doing the pre-programming work of designing/prototyping software to provide services to or solve problems for socially relevant clients. They will also learn how to recruit project clients, how to manage clients' expectations, and how to have the software implemented in later courses at their institutions, or by college-level teams involved in the RPRCC Initiative. For more information about the attractiveness of RPRCCs to girls and young women, see The PRCC Initiative Site (http://sites.google.com/site/therprccinitiative/).

CERTIFICATION AND STANDARDS FOR COMPUTING EDUCATION IN PENNSYLVANIA*

PANEL DISCUSSION

Jean Griffin (Moderator)
University of Pennsylvania

Tammy R. Pirmann
School District of Springfield Township

John P. Dougherty
Haverford College

Rita Powell
University of Pennsylvania

The goal of this panel is to continue (and to extend to other interested parties) a discussion of the certification practices and curricular standards for computing education in Pennsylvania. Specifically, we will present the current state of K-12 computing education (with particular emphasis at the secondary level), identify issues that distract or interfere with the proper delivery and transfer of knowledge in computing needed by students as they leave high school (and enter university), and list some of the options available to best prepare students for careers in the 21st century.

COMMUNITY EMPOWERMENT AND SERVICE LEARNING PRACTICES THROUGH COMPUTER SCIENCE CURRICULA OF A MAJOR METROPOLITAN UNIVERSITY[*]

TUTORIAL PRESENTATION

James Lawler
Jean Coppola
Susan Feather-Gannon
Jonathan Hill
Richard Kline
Pauline Mosley
Andrea Taylor
Pace University

SELECTING AND USING VIRTUALIZATION SOLUTIONS – OUR EXPERIENCES WITH VMWARE AND VIRTUALBOX*

Peng Li
Department of Technology Systems
East Carolina University
Greenville, NC 27858
252 328-9669
lipeng@ecu.edu

ABSTRACT

Virtualization provides a cost-effective solution for delivering hands-on education remotely. Various virtualization software packages are currently available. In this paper, we present our experiences with selecting and using virtualization technology for information and computer technology courses. VMware and VirtualBox are both viable choices with their advantages and disadvantages.

1. BACKGROUND

The Information Technology (IT) field is one of the most dynamic disciplines to emerge in recent years. The speed of technological advances, along with shrinking budgets makes it very difficult for academic institutions to maintain IT curricula in step with the changes. Additionally, there is a significant push to provide hands-on experiences that help face-to-face and distance education (DE) students learn what would have been abstract concepts in science, technology, and engineering. Our Information and Computer Technology (ICT) program at East Carolina University is not exempt from these challenges. Our lab equipment has not been upgraded in a timely manner due to budget constraints. However, the student enrollment, especially in the distance education section, has gone up rapidly. As a result, the demands for remote labs have been increasing and the lab resources have been stretched to their limits. To deal with these challenges, we started incorporating virtualization technology into our courses in 2006 in hope of finding a cost-effective way of delivering remote hands-on labs.

Virtualization enables users to run one or more virtual machines simultaneously, each with its own (guest) operating system, on a single physical computer. Consequently, the resources can be shared more efficiently. The technology was initiated at IBM in the 1960s [1]. However, little progress was made on x86 platforms until the late 1990s, when the first x86 virtualization software was released by VMware [2].

Virtualization software was used in college computer labs as early as 2002 [3]. However, due to the high cost and the unproven reliability of x86 virtualization technology, it has not been widely deployed in education or in industry until recent years. In 2006, VMware released VMware Server 1.0, free of charge, for personal use. To compete with VMware, Microsoft offered Virtual Server 2005 R2 for free download.

We realized that free virtualization solutions would provide us the opportunity to deal with the challenges that our program was facing. Virtual machines were simple to set up, easy to maintain and now essentially free. Our goal was to migrate from a centralized physical lab approach to a decentralized virtual lab approach in order to reduce hardware and maintenance costs and to deliver remote labs efficiently and effectively.

We started experimenting with virtual machines in the course Intrusion Detection Technologies (ICTN4200/4201) in fall 2006. The results of the study were published elsewhere [4]. We also integrated virtualization technology into other courses, including, among others, Network Environment II (ICTN 2530/2531) and Scripting for Information Technology (ICTN 2732). In this paper, we are going to focus on the selection of virtualization software and our experiences with VMware and VirtualBox.

2. VIRTUALIZATION OPTIONS

2.1 Selecting Virtualization Solutions

Since the late 1990s, many different virtualization projects have been developed. Commercial virtualization software offerings include, but are not limited to: VMware Workstation, Server and ESX Server, Parallels Workstation and Desktop, and Microsoft Virtual PC, Virtual Server, and Hyper-V. Notable open-source virtualization projects include, among others: QEMU, Xen, OpenVZ, Kernel-based Virtual Machine (KVM) and VirtualBox.

Some of our faculty members had been playing with virtualization since 2002. From our observations, not all options were suitable for class use. When selecting virtualization solutions, a few factors needed to be considered:

- The virtual machines would be installed on the students' personal computers. The software must be low-cost and easy to use. The software should not require high-end hardware to run.
- The majority of our students ran Windows XP or Windows Vista operating systems on their personal computers. A small percentage of students had Mac OS X. The virtualization software must be compatible.

- The virtualization software must support different guest operating systems (Windows, Linux, UNIX etc.) to make it possible to create a virtual lab with diversified platforms.

The virtual machines were not intended for game playing or word processing. So we did not care much about factors such as 3D acceleration or office benchmarks. A little overhead would be tolerable.

Among so many different virtualization solutions, some, such as Xen, OpenVZ and KVM, were excluded because they did not support Windows as host operating systems or were not easy to use; some, such as Microsoft Virtual PC and Server, were not chosen because they supported only a very limited number of Linux distributions as guest operating systems; others, such as non-free Parallels Workstation and Desktop, were not considered due to cost reasons. These applications might perform well as virtualization software in certain benchmark tests, but they did not meet our needs.

Eventually we selected VMware Server and Player in 2006, used VMware Workstation in 2007 and adopted Sun xVM VirtualBox in 2008. Both VMware and VirtualBox were easy to use. Both supported multiple flavors of Windows, Linux and UNIX as host and guest operating systems, as listed in Table 1 [5].

Table 1. VMware vs. VirtualBox (Features)

Feature	VMware Server/Workstation/Fusion	Sun xVM VirtualBox
Supported host operating systems	Windows 2000, XP, 2003, Vista, Linux, Mac OS X (Fusion only)	Windows 2000, XP, 2003, Vista, Linux, Mac OS X, Solaris
Supported guest operating systems	Windows 3.1, 95, 98, NT, 2000, XP, Vista, Linux, FreeBSD, Solaris	Windows 3.1, 95, 98, NT, 2000, XP, Vista, Linux, OpenBSD, FreeBSD, OS/2, Solaris
License costs	Workstation and Fusion free for VMware Academic Program members with restrictions, Server free for end users, not redistributable	Free for personal and educational use
Size of the installation file (32bit/64bit)	463 MB (VMware Workstation 6.5.2 for Windows)	63 MB (VirtualBox 2.2.2 for Windows)
Other features	USB support, 3D acceleration, API, AMD-v, Intel VT-x and more	USB support, 3D acceleration, API, AMD-v, Intel VT-x and more

2.2 VMware versus VirtualBox

VMware offers a variety of virtualization products geared toward different markets. VMware Workstation is probably the most popular desktop virtualization solution for

Windows and Linux. VMware Fusion supports Mac OS X as the host operating system on Intel-based Macs.

VirtualBox was originally developed by Innotek, a company later acquired by Sun Microsystems in 2008. VirtualBox has different builds for Windows, Mac OS X, Linux and Solaris hosts.

VirtualBox is more lightweight than VMware Workstation. As shown in Table 1, the size of the installation file is 63 MB for VirtualBox 2.2.2 for Windows and 463 MB for VMware Workstation for Windows. VirtualBox can run virtual machines created by VMware. To get a rough idea about the memory usage, a Red Hat Enterprise Linux (RHEL) virtual machine was launched with VMware Workstation 6.5.2 and VirtualBox 2.2.2 separately on the same physical computer. The vmware-vmx.exe process, the main process to run the virtual machine for VMware, used about 370 MB of memory while the VirtualBox.exe process, the main process to run the virtual machine for VirtualBox, used only about 46 MB of memory. When multiple virtual machines were launched on the computer, VMware used even more memory than VirtualBox. The high memory usage of VMware may not be a major issue if the virtual machines are to be hosted on high-end servers on campus. However, if the students need to run multiple virtual machines concurrently on their personal computers, VirtualBox may be a better choice because it is less resource-demanding.

More pros and cons of VMware and VirtualBox are summarized in Table 2. VirtualBox is dual-licensed and has an open source edition. The software is free for personal and educational use. VMware Server is free for personal use but it does not officially support Windows XP or Vista as host operating systems. VMware offers free licenses of VMware Workstation, Fusion and other products to degree-granting academic institutions through VMware Academic Program [6].

Table 2. VMware vs. VirtualBox (Pros and Cons)

	VMware	VirtualBox
Pros	1. supports multiple host and guest operating systems 2. user-friendly 3. better snapshot support 4. more widely adopted in industry 5. networking easier to set up 6. management tools such as ACE 7. proven reliability	1. supports multiple host and guest operating systems 2. user-friendly 3. lightweight 4. less restrictive license. 5. configuration file in XML
Cons	1. more restrictive license. 2. resource demanding.	1. not well-known in industry 2. new product and supposedly more buggy

It is easy to set up bridged, host or NAT networking in VMware. For VirtualBox, it is different. Before version 2.1, bridged networking in VirtualBox had to be set up

using command line tools or by editing the configuration file. While this was sometimes regarded as a problem for VirtualBox, we deemed it as an advantage. In some networking labs such as ICTN 4201, we intend to create a private virtual network which is "isolated" from the host operating system. By default, the VirtualBox internal network is isolated. The network settings can be changed by modifying the configuration file in XML. It is relatively easy for the instructor to create an isolated virtual network consisting of two or more virtual machines with VirtualBox. The virtual machines are then packed into one zip file and delivered to the students. The students unpack the virtual machines on their own computers to perform the labs. This is convenient for deployment in a large class.

VirtualBox is relatively new and is not as widely adopted as VMware in production environments. However, we have not had major issues in class with either application. The community support for both applications has been adequate for us.

3. INTEGRATING VIRTUALIZATION INTO COURSES

Virtual machines have been used in different courses for students to complete hands-on projects.

3.1 ICTN 4200/4201

ICTN 4200/4201 - Intrusion Detection Technologies, is a 3-credit hour undergraduate course (2 hour lecture and 2 hour lab). We started incorporating virtual machines first in this course in fall 2006. The students used VMware Player or VMware Server to run a single Linux virtual machine on their personal computers. The virtual machine acted as the server/target, on which Intrusion Detection Systems such as Snort were installed. The host computer was used as the client/attacker. In 2007, we changed to VMware Workstation and tentatively introduced a second virtual machine as the client/attacker. The student feedback was generally positive but some students reported that the virtual machines had slowed down their computers significantly. In 2008, we switched to Sun xVM VirtualBox, which had lighter hardware demands. The students were provided with a CentOS Linux virtual machine as the server/target and a Debian Linux virtual machine as the client/attacker, as shown in Table 3. An anonymous class survey was conducted at the end of each semester. In 2006, 38% of surveyed students complained VMware Server used too much resource; 50% of surveyed students said the same about VMware Workstation in 2007 when the second virtual machine was introduced. The percentage dropped to 22% after we switched to VirtualBox in 2008. More details about our experiences in this course were published elsewhere [4].

Table 3: Virtualization software used in the three courses in fall 2008 and spring 2009

Course	ICTN 2530/1	ICTN 2732	ICTN4200/1
Virtualization Software	VMware Workstation	VMware Workstation	Sun xVM VirtualBox
Hardware Requirements	Pentium IV CPU, 512 MB RAM minimal, 2	Pentium IV CPU, 512 MB RAM minimal, 2	Pentium IV CPU, 768 MB RAM minimal, 2 GB RAM recommended, 10 GB free space

	GB RAM recommended, 10 GB free space	GB RAM recommended, 10 GB free space	
Pre-built VM1	Red Hat Enterprise Linux 5.1	Fedora 10 as Linux, Apache, MySQL, PHP (LAMP) server	CentOS Linux 5.2 as server/console/target
Pre-built VM2	None	None	Debian 4.0 as client/sensor/attacker

3.2 ICTN 2732

ICTN 2732 - Scripting for Information Technology is a 3-credit hour undergraduate course, covering the web scripting language PHP and the Linux, Apache, MySQL, PHP (LAMP) server administration. The information and computer technology (ICT) students need to know not only how to code but also how to manage a web/database server. Traditionally, in this type of courses, students completed programming projects and submitted the codes to a central server. On the central server, they maintained their own home directories and websites but usually had very limited administrative privileges. Now a central server was still available for student use. In addition, a prebuilt Fedora 10 Linux virtual machine was provided to the students to complete the hands-on projects, as shown in Table 3. With virtual machines on their own computers, the students had full control over the LAMP servers. This presented them some "real world" experiences as system administrators, database administrators and programmers.

3.3 ICTN 2530/2531

ICTN 2530/2531 - Network Environment II, is a 3-credit hour undergraduate course (2 hour lecture and 2 hour lab), covering Linux/UNIX Essentials and System Administration. The course is hands-on and lab-intensive. Before the virtual machines became available, the lab was offered in a traditional computer room. Each student used a physical computer with a removable hard drive to perform hands-on exercises during the assigned lab time. The virtual machines made things much easier for both face-to-face students and distance education (DE) students. At the beginning of the semester, a pre-built Linux virtual machine was provided. Each student installed the virtual machine with VMware Workstation on her/his personal computer. On the virtual machine, the student ran a script to establish a VPN tunnel [7] with a central server on campus. Python scripts could be downloaded from the central server through the VPN tunnel to grade the hands-on labs. The instructor was able to log into a student's virtual machine through the VPN tunnel to check her/his work and to provide assistance. In the second half of the semester, students installed Linux on the virtual machine and finished other exercises. With virtual machines, students could work on the labs anytime and anywhere.

4. SUMMARY

Both VirtualBox and VMware are viable virtualization solutions for educational purposes. VirtualBox is free, lightweight, but somewhat feature-limited. If the students

need to run multiple virtual machines concurrently on their personal computers in a decentralized fashion, VirtualBox is a better choice because not everyone has a high-end machine with a lot of memory. VMware is reliable, feature-rich but resource-demanding. For centralized labs hosted on college campuses, VMware is a better choice due to the availability of management tools such as VMware ACE. If the students run only one virtual machine at a time on their own computers, VMware is also a good option. We have been using VMware and VirtualBox in different courses with success.

REFERENCES

[1] Creasy R. J., The origin of the VM/370 time-sharing system, *IBM Journal of Research and Development*, 25 (5), 483–490, 1981.

[2] Rosenblum M., The reincarnation of virtual machines, *ACM Queue*, 2 (5), 2004.

[3] Stockman, M., Creating remotely accessible virtual networks on a single PC to teach computer networking and operating systems, *Proceedings of the 4th Conference on information Technology Curriculum*, Lafayette, Indiana, 2003.

[4] Li, P. and Mohammed, T., Integration of virtualization technology into network security laboratory, *Proceedings of the 38th Annual Frontiers in Education (FIE) Conference*, Saratoga Springs, NY, 2008.

[5] VirtualBox, http://www.virtualbox.org/wiki/VBox_vs_Others, retrieved April 20, 2009.

[6] VMware Academic Program, http://www.vmware.com/partners/academic, retrieved April 20, 2009.

[7] Toderick, L. W. and Lunsford II, P. J., Using VPN technology to remove physical barriers in Linux lab experiments, *Proceedings of the 8th ACM SIGITE Conference on Information Technology Education*, Destin, Florida, 2007.

USING SUBVERSION AS AN AID IN EVALUATING INDIVIDUALS WORKING ON A GROUP CODING PROJECT*

Curt Jones
Department of Mathematics, Computer Science, and Statistics
Bloomsburg University
400 East Second Street
Bloomsburg, PA 17815
(570) 389-4500
cjones@bloomu.edu

ABSTRACT

Evaluating individual students working on a group project can be a difficult task. Students should be rewarded proportionately to their accomplishments on the project, but it is difficult to determine the exact accomplishments of each individual in the group. In this paper we show how to use a version control system as an aid in determining the accomplishments of each individual in a group programming project.

INTRODUCTION

We offer an Object-Oriented Software Engineering course at Bloomsburg University where the students work in groups on a large programming project. The class is broken down into teams of two to five students each. Each team elects its own team leader and is given a primary area of responsibility for the semester. The teams are assigned tasks to complete each week. We evaluate the students on their individual and group accomplishments for each weekly assignment. Our problem was trying to accurately determine the contributions of each team member.

As an aid to maintaining our code and document base in a project course, we use the Subversion version control system [1]. Subversion is a popular open source version control system [2]. It is an improved version of CVS that integrates into NetBeans, Eclipse and Visual Studio among other IDEs and various operating systems. A server is

needed to host the Subversion repository (our document base) and each student needs a Subversion client or an IDE with a Subversion plugin.

Originally we wanted a convenient method to allow students to work on a large project and learn how to merge their code with code modified by other students [3]. We give all students their own login credentials and enforce requirements on how they use the Subversion system. We found that Subversion was not only an excellent version control system, but contained features that allowed us to conveniently evaluate the accomplishments of each member of the class. We can now accurately determine the accomplishments of each team member working on a large group project without relying solely on self-reported accomplishments.

SUBVERSION CONTROL SYSTEM FEATURES

Subversion has many great features. We use it to allow multiple students to change documents of many different types, but in this paper we will only concentrate on those features that aid in evaluating individual students working on a group coding project.

In our environment we are currently using NetBeans version 6.5 with a Subversion client version 1.4.3 plugin. Subversion is a standard plugin for NetBeans, so the students have no difficultly configuring NetBeans to work with our Subversion server. Subversion requires users to login to a repository of code. Students are given their own login credentials and shown how to use the system during the first week of class. Subversion can be used from a Linux/Unix shell, the Mac OS or Windows Explorer, but our students predominately work right from the NetBeans IDE.

Students obtain a copy of our group coding project by *checking out* the current code base to their own local storage. Students are not *locking* a file. They all have a copy of the files in the project. They can modify files, delete files or add new files to the project. Once they have this working version of the project they can *commit* and *update* their code. An *update* action will replace any files on their local storage with new files from the server. A *commit* action will allow the user to place updated versions of files on the server.

Students are given the following set of guidelines in using our Subversion system.

(1) Update your code immediately before you start to work on the project. We want the students to have the latest versions of any files before modifying the code. This also helps minimize conflicts if two students happen to work on the same file at roughly the same time. Subversion can help manage conflicts, but not having any simplifies the task and saves time.

(2) Normally students should commit often. Students should work on one file, compile and run the program to ensure they have not introduced any obvious errors and then commit their changes. We want all students to have the latest version of all files in the project. We also want to minimize conflicts. When a student has a major change, or has a significant refactoring of the project code to committed, the student will ensure that the refactored code is thoroughly tested before the code is committed. We inform the entire class when the major revision will be committed. Students will then avoid code updates until the major revision is committed. We could lock the files to ensure we minimize conflicts, but have not found this to be necessary.

Students who introduce compiler errors or bugs into the project are penalized. Students all understand they are not to commit code that has not been compiled and thoroughly tested.

(3) Students must provide accurate messages with all commits. Subversion allows users to commit one or more files at a time and allows them to place a message with each commit. The commit message should describe what changes were made to the files. For example, students could place a message stating they changed the documentation of a class; they fixed an error in the systems or added some new functionality. When other students update their code, not only do they see the files that were modified, added or deleted in the project, they can view the messages to see what changes were actually made. Even two students working on the same team can see what changes were made to the sections of the program under their responsibility.

Subversion also allows all users to search the revision history. If the students in a class follow the guidelines above, the search history screen will show an instructor: (1) how many posts a student made for this assignment; (2) when those posts were made (for example the night before the due date); (3) what files were modified, added or deleted; (4) what lines of code were changed and exactly how the lines of code were modified.

WEEKLY SUBMISSION EVALUATION

As mentioned above, we give weekly assignments to each group. Groups have primary areas of responsibility and students within a group tend to work on the same areas of functionality each week. This limits file modification conflicts with other students in the class. Students are told that they are expected to complete 12 hours of work each week. To help them monitor how much effort they are actually committing to the course, we require all students to complete a work summary for the week. This work summary shows for each day the hours they worked and a summary of what they accomplished. Each student provides this work summary among other documents as part of their weekly submission.

Evaluating students now involves using the *Subversion Search History* window in NetBeans. This screen allows the user to specify the starting and ending revision number to include in the search. You can also limit the search by dates. This allows the instructor to limit the changes to the project to the modifications made during the week being evaluated. The instructor can also have Subversion display only the posts by a particular student for the date range being evaluated. For each post the instructor can view the post message submitted by the student and the list of files modified, added or deleted. Figure 1 shows a sample display of this screen. In this example, we are displaying a summary of the changes made by one student. We can see the commit message and the files modified by each commit.

We found that limiting the display to show the posts committed by a particular student for each assignment aids in evaluating that student. We compare the work summary document for a student with the posts a student commits. Students are now more likely to accurately create their work summary document. They know that listing three hours of work and having only one post that changes two lines of one file does not help their evaluation. Once we compare the work sited on the work summary document

with the posts committed by the student, we then look at the changes actually made to each file.

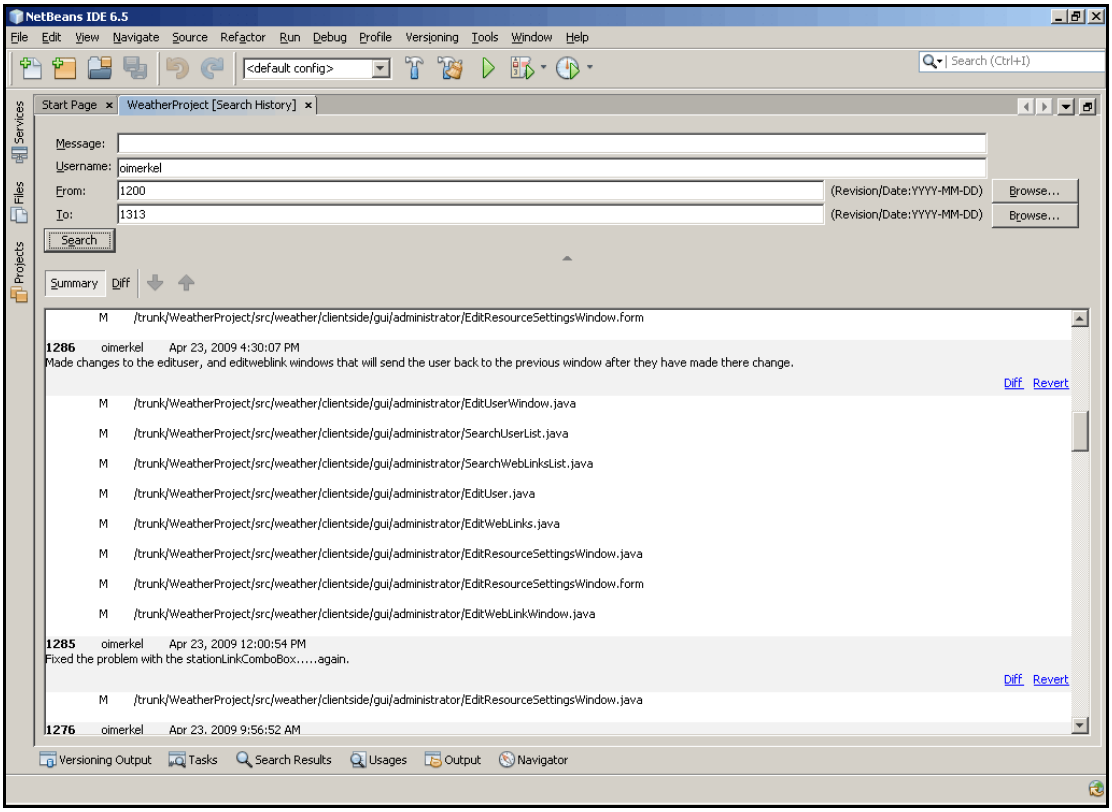


Figure 1 Search Screen History Window

Subversion allows us to see the difference between this version of the file and the one right before this post. The *file difference screen* has color codes placed on the right hand side of the display that indicates the type of changes committed. The most common colors are red, blue, and green. Red indicates that these lines of code were deleted from the last file version to the current file version. Blue indicates that these lines of code have been modified in the new version and green indicates the lines of code have been added for this version.

Instructors can now quickly get an accurate impression of the modifications made to each file. The instructor can also look at the exact lines of modification. The instructor can check the lines of code modified each week to verify the students did not violate any style or formatting conventions. Correctness issues can also be determined by being able to hone in on the exact lines of code modified in each file. Figure 2 shows an example of this technique. In this example we have show the difference between revision 1285 and 1286 of a file. We can see that two lines of code were added between lines 302 and 303 of version 1285 of the file. Because of other changes in the file, these lines are now numbered 314 and 315. We can also see that several lines of code were added between lines 304 and 305 of version 1285. The color coded slider bar to the right of version 1286 allows us to immediately locate the changes in the file.

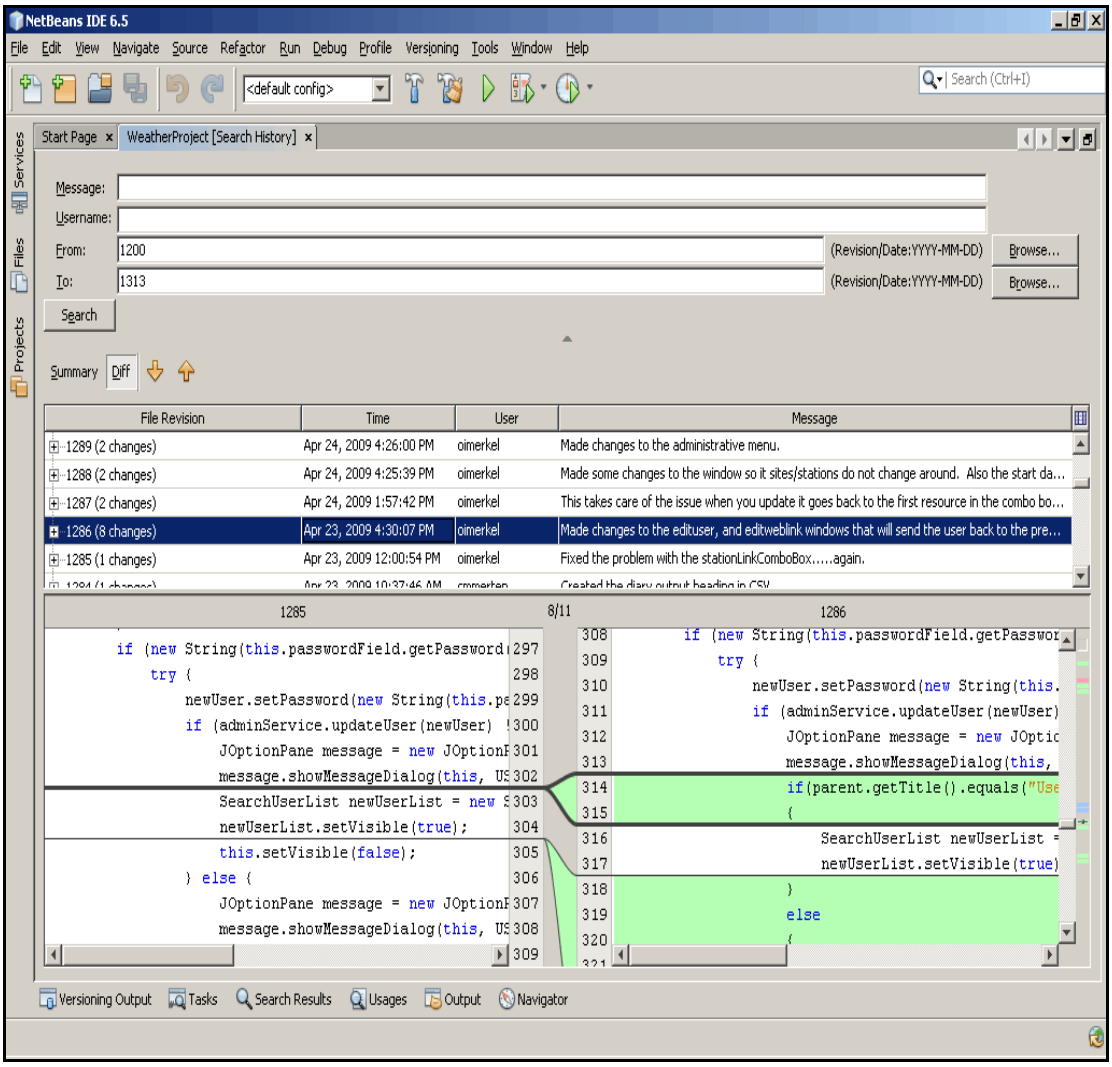


Figure 2 File Difference Screen Example

CONCLUSION

Subversion is version control software with many practical classroom usages. It is easy to use and simple to install. It allows students to conveniently work on a large programming project and enables an instructor to efficiently evaluate the contributions made by each student in the class. It also allows students to be involved in open source projects. We strongly encourage its use in courses with group programming projects.

REFERENCES

[1] Subversion is available at http://subversion.tigris.org/.

[2] Reid, K., and Wilson, G., Learning by Doing: Introducing Version Control as a way to Manage Student Assignment, *Proceedings of SIGSE*, 272 - 276, 2005.

[3] Clifton, C., Kaczmarczyk, L., Mrozek, M., Subverting the Fundamentals Sequence: Using Version Control to Enhance Course Management*, Proceedings of SIGSE*, 86 - 90, 2007.

REAL-TIME COLLABORATION TOOLS FOR DIGITAL INK*

Steven Lindell
Haverford College, Haverford, Pennsylvania
(610) 896-1203
slindell@haverford.edu

ABSTRACT

The importance and use of digital ink in formulas and diagrams is well recognized, and currently utilized by scientific and technical users of pen-enabled computers. Similarly, there is a growing recognition as to the utility of collaborative tools that allow multiple users to edit a document, either simultaneously or asynchronously. Finally, more people are using real-time communication tools to enhance problem-solving teamwork and productivity in commercial and educational environments. But the ability to perform all of these tasks simultaneously is poorly supported by current software solutions. In this paper, we explain how real-time collaboration tools for digital ink should provide the potential for increased technical productivity, along with the results of functional software testing that was performed by several students and a teacher in efforts to identify those features most necessary for their effective use.

INTRODUCTION

Typically, in mathematics classes, problem-solving sessions replace the usual notion of a science laboratory for the purposes of hands-on learning. In those sessions, an instructor (often a teaching assistant) helps students with solutions and facilitates a discussion. However, more recently, it has been strongly recommended that the joint activity of solving problems in small groups is a superior method of learning, often called active learning for obvious reasons. The skills and knowledge obtained in this type of social interaction often seem to stick in a more permanent fashion [4].

In our investigation, we specifically focused on problems in science and mathematics that would typically be solved by drawing a diagram and/or writing a formula. The usual method of interaction between students in this case would be to write

---

using pencil and/or pen on a piece of paper laid horizontally on a table, or chalk or dry-erase marker vertically on a blackboard or whiteboard respectively. While these forms of interaction are well understood as providing a high degree of bandwidth and interactivity, they require the participants to be present in the same room (disallowing remote collaboration), and do not provide a convenient way of saving work from one session to the next (unless an electronic scanning board is utilized). Although electronic whiteboards do exist to provide remote collaboration, their size, expense, and lack of portability make them difficult to use in the typical classroom.

During the summer of 2006, I was involved in supervising a project that I initiated to study collaborative problem solving, and investigative the utility (if any) of utilizing tablet computers to assist in this activity. Its purpose was to investigate the potentials and limitations of using tablet computers for collaborative problem solving and classroom presentations, and the project was supported by the Cascade Mentoring Program at my institution, funded by the Howard Hughes Medical Institute.

In order to study the potential advantages that electronic communication and storage might provide, we employed a high school teacher and two high school students, working with a combination of portable and desktop tablet computers. As mentioned, the types of problems chosen were specifically designed to take advantage of the ability to draw diagrams and write formulas directly on the screen, and the hardware we used specifically supported this natural form of interaction.

BACKGROUND

The project started with the premise that all types of problem solving are facilitated by two fundamental principles: the ability to communicate ideas with other interested parties in real-time; and the ability to store the results of those interactions for long-term reference. In this paper, we will refer to these fundamental principles as *collaboration* and *persistence*, and study the ways in which these principles are enhanced (or hindered) by the hardware and software technologies that were tested.

But before we go into the details of our preliminary results, it would be useful to describe from a more abstract point of view the ideas of collaboration and persistence. We will assume that the primary aspect of problem solving (at least in science and mathematics) involves the building of mental models. Transmitting those models through time and/or space is a fundamental challenge to any individual or group working on a technical solution. Those models often take the form of a combination of textually annotated diagrams and formulas, together with the ancillary oral explanations that people often add to such forms of written communication. This information is most readily transmitted by pen and voice, though it can also be captured on a tablet computer equipped with a microphone.

REAL-TIME COLLABORATION

Communication can be defined as the transmission of information from one point in space (here) to another (there). Typically, participants engage in the same room, where voice transmission takes place face to face. The sharing of written information can occur through a (mobile) piece of paper, a (fixed) board, or a combination of these two

instruments. However, neither permits remote collaboration between participants that cannot directly see or hear each other. Communication allows information transfer between people separated in space.

LONG-TERM PERSISTENCE

Storage can be defined as the transmission of information from one point in time (now) to a later point (then). When someone records written information on a piece of paper or board, those marks are retained until the piece of paper is lost or destroyed, or the board is erased. On the other hand, oral information is not retained in a conversation unless a specific technology for voice recording is employed, such as a microphone attached to an electronic recorder or computer. But even in the ideal situation where the paper is not misplaced and the board is not erased, it is difficult for each participant to retain a written copy of those interactions. Storage makes possible access to information separated by time.

PAPER

One of the supreme advantages of paper is that it combines both the options of long-term persistence and real-time collaboration. People can annotate an existing document, carry that information from place to place, and even collaborate with someone else using the same piece of paper, which retains a record of that information interaction. However, editing information (such as moving things around) or erasing is tedious at best (with pencil) and impossible at worst (with pen).

Moreover, this "technology" does not permit remote collaboration, nor does it allow each of the participants to each retain a written copy of their interactions (without a copy machine). Using a tablet computer could solve both of these problems, allowing multiple participants to collaborate from separate locations, and in addition allow for the long-term storage and organization of notes related to specific problem solutions obtained as the result of that collaboration.

SIMULTANEITY IN SPACE AND TIME

The final ingredient that needs to be specified is known as *deixis* -- the proof or demonstration of an idea by simultaneously speaking about it and pointing to it -- a ubiquitous technique in nearly all forms of instruction. What is really going on here is that in order to transmit information about a mental model from one individual to another, the teacher (the one who is doing the transmitting) must indicate which aspect of a diagram or formula he or she is currently talking about. By pointing to the object under consideration, the teacher is cementing a correspondence between the temporal information being transmitted (by voice) and the spatial information being transmitted (by pen). This is why teaching at a board has been such a successful format, and why most professors (particularly in math and science) will refuse to teach in a classroom without ample board space. Instructors typically use their hand, a piece of chalk, or a long stick as a pointer to draw attention to the specific area of the board that is under consideration at a given moment. As an aside, the best scientific articles utilize a similar

technique to connect the written text of a discussion (which conveys the temporal information) with specific numbers or letters called out on an illustration (conveying the spatial information). Scientific American magazine has become famous for its colorful illustrations which, when describing a process, often include specifically numbered areas with individual captions. The importance of deixis in online mathematics education is well recognized [5].

In what follows, we will outline the approach we took to analyzing various hardware and software configurations, and how they assisted or fell short of the goals articulated above. We found that of all the implementations, it was most difficult to perform both deixis (for the teacher) and organization (for the student), and that these items were the most challenging to achieve from a performance point of view.

PRIOR WORK

One of the first equipment designs to incorporate nearly all the important ideas required for remote collaboration using digital ink was developed by Hewlett-Packard. Originally known as the DeskSlate [6], this device permitted two parties (and no more) to collaborate with voice and digital ink over an ordinary telephone line. In real-time, each person could hear the other's voice (like an ordinary telephone conversation), and simultaneously see each other's scribbling on a jointly shared screen. Both parties had their own individual pens, which when they weren't inking served as pointers. When a session was over, each participant could save the resulting screen images, organize them, and even print them for archival purposes. The machine was marketed as the Omnishare [7], but it did not catch on because of its very high price, limited market appeal, and poor penetration (remember that you could only use its capabilities with the person you were calling if they also had one of these hard-to-find and expensive devices). However, in terms of persistence and collaboration, together with the simultaneous voice and support for dual deixis pointers, this is the design to beat.

Hewlett-Packard's Omnishare [8]
(reproduced by permission of photographer)

HARDWARE

Digital inking requires an absolute positioning device, so special hardware is required because a mouse is a relative positioning device (i.e. it senses motion). The least expensive way to do this is to use a graphics tablet which senses the position of an RF pen above its surface, and translates that into a cursor position on the screen. Although this is a perfectly functional method, these can take quite a bit of getting used to since they demand a fair degree of hand-eye coordination. A simpler, but more expensive solution is to use an integrated tablet monitor, in which the digital pen "writes" directly on the screen. These are most commonly available as special notebook computers known as Tablet PCs, and also as specialized desktop monitors used in graphics applications.

We chose to use a pair of tablet computers made by Sony around 2002, model LX-920. These unique desktops [9] included an integrated pen-tablet monitor, and ran the ordinary version of Windows XP (not the special Tablet PC version included with notebooks).

Sony's PCV-LX920 desktop tablet computer

Because the monitor allows the screen angle to be adjusted from vertical to nearly horizontal, this hardware provided an almost ideal platform to evaluate digital inking.

We also used one of the first generation Tablet PCs made by Hewlett Packard, the TC1000.

The HP TC1000 Tablet PC

This hybrid (as it's known in the industry) is convenient because the tablet can be detached from its keyboard for carry-around portability, or perched on its keyboard for use on a table. Its small size and wireless capability was also important because it can be used by the instructor to see what students are doing on their computers while moving about the classroom.

Apple currently provides little support for digital ink on their computers, despite the fact that their portable Newton MessagePad was one of the first devices to support digital ink in a robust way across its platform. In its last incarnation before being discontinued, the handwriting recognition was reportedly excellent. But despite this, Apple has never since provided substantial support for incorporating pen-based ink.

SOFTWARE

A fair number of pieces of software are ink-enabled, such as the well-known Microsoft PowerPoint, making it useful in a one-way mode that would typically be used in a lecture-style classroom for teacher-to-student interaction. Also, because only one computer is being used, the cursor serves well as a pointer for deixis. A lesser-known piece of ink-enabled software called Microsoft Office Document Imaging allows one to electronically annotate any document that can be printed or scanned, but it too does not provide a mechanism for collaboration. However, both pieces of software allow annotations to be saved, and therefore do permit persistence. In the context of teaching, this permits the instructor pick up where he or she left off last. This is a significant difference between ink-enabled overlays (which work with virtually any piece of software) and ink-enabled software in which the capability to store annotations digitally is built into the underlying file format.

Since our work requires remote collaboration between two computers, we focused on ink-enabled software that could work in a two-way mode to allow for peer-to-peer interaction required in problem-solving sessions between students. Identifying such software and determining which features were crucial formed the focus of our work. We examined five programs: OneNote (an optional part of Microsoft Office); DyKnow; Classroom Presenter; Conference XP; and NetMeeting (a little known part of Microsoft Windows). The DyKnow software has been the focus of many articles highlighting its usefulness in classroom presentation and group work, e.g. [1, 2].

The two high school students worked together solving many sets of mathematics and physics problems, including logic puzzles and brain teasers (selected by the college student mentor). Their experiences formed the basis of our findings. Experiences using Tablet PCs in a variety of other scenarios are the subject of an entire book [3].

FINDINGS

Multiple panels (i.e. virtual whiteboards) were extremely helpful, especially if there was an option to synchronize them (ensuring that students were literally on the same page). Sitting across from each other, students had to rely on a remote pointer for deixis. Since this was the most poorly implemented feature in general, they preferred to sit side by side (though it was pointed out that there may be a gender basis for this, since both students were women).

With the exception of OneNote, which had a vast feature set and capability for organization, none of the other pieces of software provided a way for students to retain their handwritten work in a coherent fashion (i.e. organized persistence). Support for remote pointers was nonexistent in Conference XP, though it did allow for voice communication. In Classroom Presenter, panels were not editable during a presentation, but it did allow inking from multiple people at once. NetMeeting featured both application sharing and voice communication, but only between two users (i.e. no conference calls). However, there was whiteboard support for multiple users, including simultaneous inking and text editing, with movable hands that served as repositionable remote pointers (though they often were buggy). As mentioned, OneNote featured powerful organizational tools, and a unique file sharing mode where multiple users could simultaneously text edit and ink to the same open file, as long as they didn't modify the same information simultaneously. But highly inked pages had long latency and refresh delays. Remote pointing was implemented with disappearing ink – information could be circled or underlined, and seconds later that annotation would be gone. This was acceptable, but certainly not ideal for proper communication in complicated problems. DyKnow was the most capable piece of software – not surprising since it was specifically designed for collaborative instruction. Its strengths were in its ability to manage the instructor-to-student interaction, but the remote pointer displayed a very high degree of latency.

CONCLUSIONS

NetMeeting is a free piece of software included on every Windows computer, and can perform some of the most basic tasks related to remote collaboration, including application sharing, voice communication, and basic digital inking with pointers. Although multiple users can take advantage of the whiteboard features, there's no facility for instructor management, nor is there ability to organize session notes. This software was most useful for taking advantage of real-time interactivity.

OneNote had superior organizational features, but its difficulty with lots of ink on a page, and the use of disappearing ink instead of a remote pointer, limited its utility for high bandwidth real-time interactivity required in solving technical problems. This software seemed better suited for asynchronous collaboration in which organized persistence was far more important than real-time communication.

DyKnow provided a strong suite of tools for instructor-student and student-student interaction, making it the most promising piece of classroom-based software. Remote pointer lag and the limited ability to organize session notes made this product more difficult to use for remote collaborative work requiring high interactivity or information management. But, it was the only piece of software evaluated that was designed by a company specifically dedicated to educational technology.

The most challenging (and in many ways the most important) feature was to facilitate deixis, with individual pointers representing remote cursors (something the OmniShare had no trouble with even though it was developed and based on a technologically challenged platform compared with today's devices). People naturally need to point to items within equations or figures in order to convey their "point". Until this feature can be implemented with a high degree of reliability and speed (enough to

keep up with a person's voice), remote collaboration involving digital inking will remain noticeably inferior to live contact between parties at the same board or screen.

ACKNOWLEDGEMENTS

The author appreciates financial support for this project from HHMI, along with the participation of high school teacher Alan Bronstein together with his students Angel Feng and Katherine Sioson from Central High School in Philadelphia, along with the mentoring from Haverford College student Michael Jablin. Additional support from DyKnow, allowing us to use the software gratis, is also appreciated.

REFERENCES

[1] Berque, D., Bonebright, T., Whitesell, M., Using pen-based computers across the computer science curriculum, *SIGCSE Bull.,* 36, (1), 61-65, 2004.

[2] Berque, D., An Evaluation of a Broad Deployment of Dyknow Software to Support Note Taking and Interaction Using Pen-Based Computers. *J. Comput. Small Coll.*, 21, (6), 204-216, 2006.

[3] Berque D.A., Prey, J.C., Reed, R.H., *The Impact of Tablet Pcs and Pen-Based Technology on Education 2007: Beyond the Tipping Point*, Purdue University Press, 2007.

[4] Bonwell C.C., Eison, J., Active Learning: Creating Excitement in the Classroom, ASHE-ERIC Higher Education Report No. 1, George Washington University, Washington, D.C. 1991.

[5] Cakir, M.P., Sarmiento, J., Stahl, G., Shared deictic referencing in online mathematics discourse, *27th Urban Ethnography in Education Research Forum*, Philadelphia, PA, 2006.

[6] O'Conaill, B., Geelhoed, E., Toft, P., Deskslate: a shared workspace for telephone partners, *Conference Companion on Human Factors in Computing Systems*, 303-304, ACM CHI, 1994.

[7] Hunter, A., Pen-Based Interaction for a Shared Telephone Workspace, *Advances in Human-computer Interaction* Jakob Nielsen, Intellect Books, 1995.

[8] Sharpe, W.P., Stenton, S.P., Information appliances, *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, Erlbaum Associates, 2003.

[9] Sony Corp., *VAIO LX Series Developers' Stories*, originally published on Sony's Japanese website, available at www.haverford.edu/cmsc/slindell/VAIO%20Story.pdf, retrieved June 19, 2009.

IMPROVING BELIEVABILITY OF SIMULATED CHARACTERS*

Jere Miles
Division of Business and Public Service Technologies
Wilkes Community College
Wilkesboro, NC 28697, USA
828-719-1844
jere.miles@wilkescc.edu

Rahman Tashakkori
Department of Computer Science
Appalachian State University
Boone, NC 28608, USA
828-262-7009
rt@cs.appstate.edu

ABSTRACT

In recent years the video game industry has experienced rapid expansion developing virtual environments that accurately mimic a real-world setting. However, the industry almost entirely relies on finite state machines for deploying computer-controlled characters within these environments. This has resulted in simulated inhabitants that lack the same degree of believability as their surroundings. As part of this research a simulation was developed using Java in which an agent was placed. In a survey students were asked to rank the believability of different artificial intelligence techniques employed by the simulation. The genetic algorithm developed for this simulation provided for an agent whose decisions were more believable than the decisions generated by a finite state machine or random selection process.

INTRODUCTION

The notion of believability is explored within this paper through the implementation of a virtual environment that simulates a popular consumer video game inhabited by a computer-controlled character. The research is particularly interested in the role that different decision-making mechanisms may play in the observed believability of the character.

It is hypothesized that incorporating evolutionary computing techniques, such as a genetic algorithm, within the agent's decision-making process may provide an increased believability of the agent over those utilizing more traditional video game artificial

---

intelligence approaches. In order to test this hypothesis a simulator was developed with three separate decision-making mechanisms: a random selector, a finite state machine, and a genetic algorithm. These mechanisms were observed in operation by human viewers who rated each on their believability as a human-like character within the virtual environment.

This research is significant as there is little or no publication exploring the potential differences between these artificial intelligence techniques in terms of perceived believability. Many researchers have documented the improvement that evolutionary techniques provide in a competitive or problem solving environment. There have also been documented efforts that have discovered some of the required characteristics to improve the believability of a computer-controlled character [3][5]. However, these efforts have not focused on the decision-making mechanism employed by the agent. This paper will address the potential usability of evolutionary artificial intelligence techniques within a video game environment. While exploring the usefulness of these methods, it is expected to demonstrate that these approaches are not only achievable, but that they will provide an increase in believability of the computer-controlled character to a human observer. This increased believability may provide for a more engaging interactive product that can be delivered by the video game industry.

1.0 BACKGROUND

The artificial intelligence systems of video games have historically relied on a finite state machine to provide a decision-making mechanism for the computer-controlled characters. These finite state machines are implemented in virtually all computer games that have been developed and released [1]. Recent additions have allowed scripting techniques to be used as a method to fine-tune the state machines that actually generate the logic for the agents [7]. It has been considered necessary for the video game to employ these techniques in order to ensure a stable and testable product. Also, finite state machines require little computational overhead in order to reach decisions [1].

In recent years, however, human players have begun a migration towards games that involve playing with other people rather than playing with a computer. This trend seems to be a result of human players finding the computer-controlled characters to be too predictable in their behaviors. Other human players were also preferred for their ability to be flexible and adapt their responses to dynamic situations [6]. In recent years, there has been a renewed interest in the development of believable video game agents with the success of Electronic Arts' The Sims [2]. This series of games is considered to have raised the expectation of artificial intelligence within a video game and many developers are attempting to replicate a similar degree of believability [7]. The premise of The Sims is that the player plays the role of a virtual human making the decisions of this character's life [4]. A character that is not directly under the control of a human player generates its actions based upon the behaviors that are present in all of the objects [8].

2.0 IMPLEMENTATION

2.1 System Overview

In order to test the impact that genetic algorithms may have on the believability of a computer-controlled character, a virtual environment was created to mimic many of the capabilities and displays of Electronic Arts' The Sims [2]. The simulator was named StudentLife and was populated by a computer-controlled student character which was then observed in operation by human volunteers. The virtual environment did not allow these observers to interact with the system in any way other than through observance. This insured that the observers were focused on the behaviors of the agent being surveyed. The system allowed the human observers to have a full view of the virtual environment and all of the internal conditions of the agent.

The system queries the computer-controlled character to discover whether the agent is in need of changing its current activity during each cycle of execution, as depicted in Figure 1. If a change is required, the system requests this new activity from the logic engine. The system contains three separate logic implementations allowing for decisions to be made for the agent through different mechanisms. Whether a new activity has been returned by the logic engine or not, the simulation must update all of the screen objects and the internal conditions of the agent.

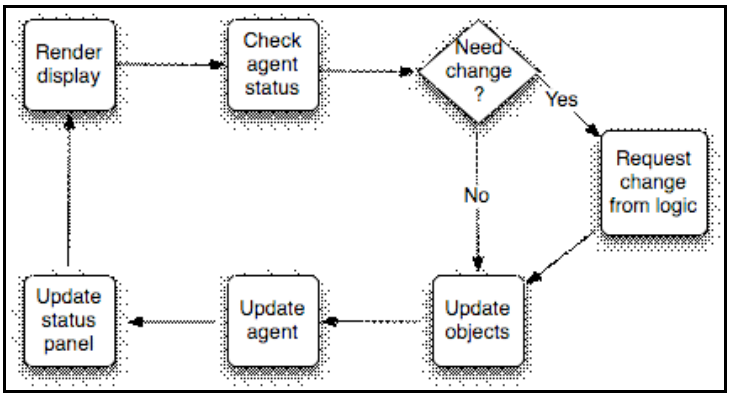


Figure 1: Standard operation of the *StudentLife* simulator

2.2 Graphical Representation

The visual representation of the StudentLife simulation, as seen in Figure 2, presented the virtual environment in the top portion and the agent's information in the lower section. The display and animations were designed in a manner intended to mimic the display of The Sims. The objects could be updated on the screen to depict changes resulting from agent behavior. Character animations depicted the various activities.

Figure 2: Screenshot of StudentLife

The bottom portion was divided into two distinct sections. The left section was utilized to present textual information as to what the agent was doing. This was done to insure that there was no confusion caused by animation sequences that did not match an observer's mental image of such activities. The section on the right-side was used to display the current needs of the agent. A graphical representation of these needs allowed for the observer to be able to quickly analyze the condition of the agent. A need that was mostly full was displayed with a green color. A yellow color represented a need that was more pressing and red was utilized for the needs that were in urgent need of attention.

2.3 Agent Architecture

The student agent was designed with nine essential needs that dictated the internal conditions of the agent. These needs were gathered from those that were utilized within The Sims and could range in value from 0 to 20. An agent's overall happiness is derived from a combination of these internal needs. A student character that has met its needs would be considered to be happier than a student who has one or more needs not being addressed. The needs of the agent are dynamic in nature and are influenced in either negative or positive manners by activities that the character engages in.

The agent has been developed with thirty-three individual activities that it is capable of performing. Each of these activities contains positive or negative modifiers for the needs of the character. The combination of modifiers is unique for each activity. These activities were gathered from the many activities that are utilized within The Sims and modified for use within this project. The modifiers of the available actions were determined through observation of similar actions as they were executed within The Sims. Due to the dynamic nature of the needs of an agent, the simulated student is required to continually alter the actions that are performed in order to maintain a higher level of contentment based on the values of the internal needs. The character was given full knowledge of these activities as well as how and where to perform them. This was to insure that the agent did not attempt to perform unrealistic actions.

2.4 Agent Logic Engine

The logic engine is intended to provide mechanisms for the character to independently select an activity in which it participates. The needs of the agent were converted into states, such that an agent that had a pressing hunger need could be considered to be in a hungry state. These states were utilized by all of the logic implementations. Three distinct techniques for providing the character with a decision making process were developed; a random selection process, a finite state machine, and a genetic algorithm.

3.0 RESULTS

3.1 Overall Believability

Ninety-seven surveys were administered to the observers. In Figure 3, a comparison is provided of the highest-ranking values from each of the observers. Of the human observers, 43% found that the genetic algorithm behaved the most believably of the options. The remainder was split with the finite state machine preferred by 27% and the random selector rated the highest by 11%. The remaining 18% of the observers gave a similarly high value to more than one of the logic engines on their survey.

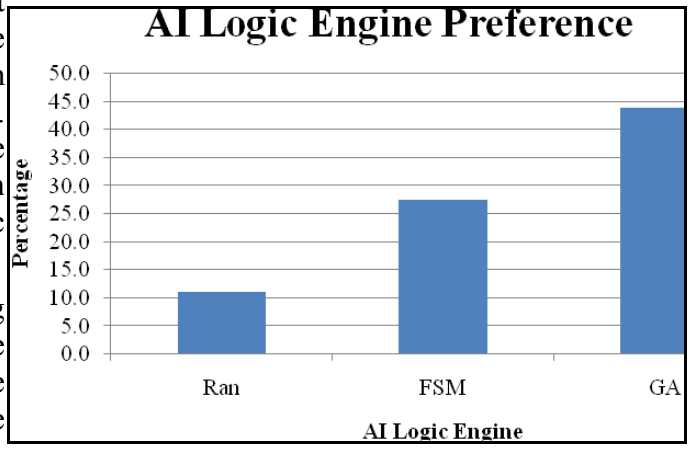


Figure 3: Logic engine preferences overall

It is worth noting that among the respondents, those who gave the random engine a higher score did so due to the variations in the activities that were performed. The observers expressed an interest in the believability of the random student because of the different actions that were performed, not because of the condition in which the agent was. This implied that for 11% of the respondents, the state of the agent's needs and happiness had no bearing on the believability of the behavior of the computer-controlled character.

3.2 Believability by Field of Study

This research was also interested to determine whether an observer's educational background in computer science resulted in any major differences in the perceived believability of a computer-controlled character. To find if there was any significance, the believability results of the surveys were compared based upon the major field of study of the observer. The observers were sorted into two education related groups: computer science and non-computer science. The computer science group consisted of twenty-four respondents, while the non-computer science category contained seventy-three observers. Of those that were not in computer science, the respondents ranged from construction to biology, with the largest group being health promotion majors at thirteen members.

Figure 4 displays the overall preferences of these two groups of observers. As can be seen in the charts, the genetic algorithm was preferred by both groups with 39% of the non-computer majors and 58% of the computer science majors ranking the genetic algorithm higher than the other

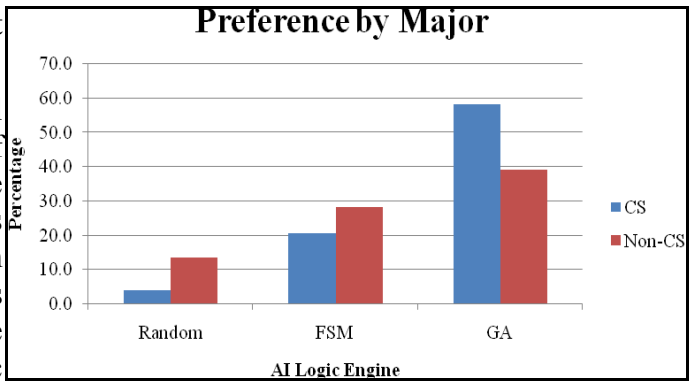


Figure 4: Logic engine preferences by major

options that they observed. In both groups there were over 15% of the respondents that ranked more than one of the engines equivalently.

The random engine was preferred by more of the non-computer science majors, however, even at 13.5%, the random engine did not score well enough to be comparable to the other techniques that were employed in this study. The computer science majors preferred the genetic algorithm by 28% while the non-computer science group preferred the same algorithm by only 11% over the finite state machine.

Table 1 depicts the correlation values of field of study with logic engine preference. These results demonstrate that there was no relationship between the ratings that observers provided and the field of study that observers possessed, with the exception of the random selector. Both the finite state machine and the genetic algorithm contained no correlation with the observer's current field of study, indicating that those engines will receive consistent ratings across the study group.

Table 1: Logic engine rating and field of study

Logic Engine	Absolute Value of Correlation
Random selector	0.2496
Finite State Machine	0.0493
Genetic Algorithm	0.0652

The principal complaint against the finite state machine by all the observers, as indicated in the observer's written comments within the survey, was that it provided a predictable behavior. Observers, regardless of educational background and field of study, noted the consistency of the computer-controlled character when it encountered similar situations. This consistency led to patterns in the behavior of the agent that the observer's commented on. Also, this predictability is in line with the complaints that had been noted in previous works in regards to the artificial intelligence of a video game system [7].

3.3 Believability by Gaming Experience

It was anticipated that there would be a difference in opinion between those who play video games on a daily basis and those who do not play at all. The potential relationships between how often an observer played video games and the perceived believability score that each agent received were an aspect by which this research was intrigued. The study group provided an even distribution across the categories of video gaming experience.

The preferences of the observers are displayed in Figure 5. It is worth noting that the random engine performed well among the groups that played video games once a week or less. In those categories, 18%, 15%, and 22% of the observers preferred the random selector, respectively. However, for those who played games two to three times a week or more, the random selector was preferred by only 4% of the respondents. The genetic algorithm was preferred by 63% of those who played two to three times a week. 47% of those who played video games every day found the genetic algorithm to be the

most believable. It is also significant to note that the genetic algorithm outperformed the other techniques in the opinions of those who had never played video games.

Table 2 displays the correlation values that were derived to determine the relationship between the rating given to a logic engine and the amount of video gaming experience the observer possessed. It is evident from these findings that there is a weak relationship between the ratings that observers provided for the logic engine and the video gaming experience of the observer.

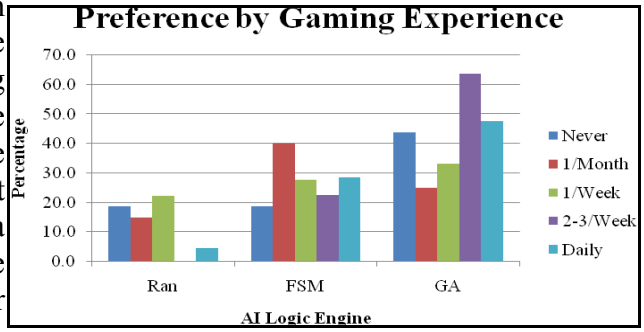


Figure 5: Gaming experience preference

Table 2: Logic engine rating and gaming experience.

Logic Engine	Absolute Value of Correlation
Random selector	0.1397
Finite State Machine	0.2075
Genetic Algorithm	0.1053

4.0 CONCLUSIONS

From the results that were collected, it has been concluded that a genetic algorithm could be developed that has the ability to govern the decision-making process for a computer-controlled character within a simulated environment. The concerns of a dumb intelligence evolving [8] were adequately addressed through the strength of the fitness function in combination with the mutation capability to introduce new genes to an evolving set of solutions. As many games currently employ finite state machines, a genetic algorithm could be developed utilizing the previously designed states, activities, and cycles of the finite state machine to provide a more believable computer-controlled character. These results could be applicable to all video games that utilize synthetic agents for the purposes of story advancement or cooperative game play. Genres that could benefit from this type of artificial intelligence technique include role-playing games and simulations.

REFERENCES

[1] BUCKLAND, MATT, 2005, *Programming Game AI by Example*, Woodware Gamed Developer's Library.

[2] ELECTRONIC ARTS, 2008, http://thesims.ea.com.

[3] LAIRD, JOHN, E., DUCHI, JOHN, 2000, "Creating Human-Like Synthetic Characters with Multiple Skill Levels," *AAAI 2000 Fall Symposium Series: Simulating Human Agents.*

[4] MACEDONIA, MICHEL, 2000, "Using Technology and Innovation to Simulate Daily Life," *Computer*, V. 33, No 4.

[5] PAIVA, ANA, DIAS, JOAO, SOBRAL, DANIEL, AYLETT, RUTH, SOBREPEREZ, POLLY, WOODS, SARAH, ZOLL, CARSTEN, HALL, LYNNE, 2004, "Caring for Agents and Agents that Care: Building Empathic Relations with Synthetic Agents," *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*.

[6] SWEETSER, PENELOPE, JOHNSON, DANIEL, SWEETSER, JANE, WILES, JANET, 2003, "Creating Engaging Artificial Character for Games," *Proceedings of the Second International Conference on Entertainment Computing.*

[7] WOODCOCK, STEVEN, 2001, "AI Roundtable Moderator's Report," *2001 Game Developer's Conference.*

[8] WRIGHT, WILL, FORBUS, KENNETH, 2001, "Some Notes on Programming Objects in The Sims," *Qualitative Reasoning Group.*

BEYOND "NOT-INVENTED-HERE": DEVELOPMENT ENVIRONMENTS FOR A MULTIMEDIA COMPUTATION COURSE*

Stephen P. Carl
Department of Mathematics and Computer Science
Sewanee: The University of the South
Sewanee, TN 37383
(931) 598-1305
scarl@sewanee.edu

ABSTRACT

Multimedia computation has emerged as a promising way for educators to motivate student interest in computer science and to build interdisciplinary bridges to the art, music, and graphic design worlds. This paper introduces two software development environments created by practitioners from the world of digital art, graphic design, and electronic music – practitioners who have leveraged training in computer science to develop systems well suited not only to media content developers, but to teaching multimedia programming in the CS curriculum as well. We describe *Processing*, a Java-based IDE designed and used by digital artists, and *Pure Data*, a graphical development environment primarily for audio and video processing, and discuss classroom experiences with each in a sophomore-level multimedia programming course.

MOTIVATION

The recent challenging climate of falling enrollments and problems with retention in computer science programs has motivated innovations by researchers seeking to bring new energy into the curriculum, particularly in the first and second year course sequences. One theme that has emerged is that of *contextualization* [5], which seeks to place computational thinking in contexts more relevant to students within the greater technological world. Multimedia computation places programming in the context of

---

manipulating digital media, appealing to the student through use of technologies that have become part of their everyday lives, such as webcams, digital cameras, sound synthesis, and compressed audio file formats such as MP3.

Multimedia computation also allows us to build collaborations with artists, musicians, and designers working in new media. An important goal for our media computation course, designed in 2007, was to support both the CS curriculum and the needs of new media artists in other departments. For this reason, we chose to base the course around software development environments *already* being used by faculty in the fine arts, even though these systems were not developed specifically with computer science pedagogy in mind [4]. The course, Multimedia Programming and Design, is a sophomore-level course intended as a second course in the computer science curriculum, with a prerequisite of either Introduction to Computer Science (a course for non-majors) or Introduction to Programming (our CS 1 course). Our department goals for this course include offering an attractive elective for students interested in a minor in Computer Science, providing a bridge between CS 1 (offered every term) and CS 2 (offered only in the Spring term), and attracting CS students to independent studies or projects in digital media.

Open-source and collaborative technologies provide viable and affordable options for departments wanting to incorporate digital media in their programs. In the rest of this paper, we first discuss the *Processing* and *Pure Data* systems, then describe how they are used in our media computation course.

PROCESSING

The Processing system is targeted at artists, digital media designers, and others interested in learning to write programs for digital media [9]. The base system features a simplified development environment designed to be accessible to students and practitioners new to programming, together with an API geared toward drawing, 3-D graphics, image processing, and simple physical modeling.

The Processing language is essentially a subset of Java with a few special wrinkles. A typical Processing program, or *sketch*, is made up of a **setup** method, a **draw** method, and any other methods needed by the program. The **setup** method contains code to run once at startup, not unlike a class constructor, and **draw** contains code to be executed *continuously* so that a drawing window is updated at a default rate of 10 frames per second (the programmer can specify another rate, or turn off continuous drawing entirely). This behavior makes it quite easy to conceive of and implement simple animations and interactive graphical programs.

Figure 1 shows an example of a simple Processing sketch displayed in the Processing IDE. The design goals of the system include functional simplicity in the user interface: along with just five menus, there are six buttons representing the common actions **Run**, **Stop**, **New**, **Open**, **Save** and **Export**. The area below the editor window displays messages from the system; in this case, a runtime error is being displayed.

Running a sketch opens Processing's ubiquitous *drawing window*, which is always present, even for sketches that produce only text. The drawing window can be displayed in full-screen mode or (using the appropriate libraries) saved as a PDF file. Animations

can be saved as a series of image files in the PNG format. The system allows the programmer to export a completed sketch as a JAR-based application or a Java applet for presentation on the Web.

The entire Processing IDE is a Java application running on the Java Virtual Machine. When the programmer clicks the **Run** button, the files making up the program are converted internally into inner classes of the main Processing system class, PApplet, which is then compiled and run. This process is completely opaque to the programmer, who can use the system without ever seeing or understanding what is happening "under the hood."

The Processing API includes a number of methods for event handling that the programmer can redefine, such as what to do when the mouse is pressed or dragged, a key is pressed, and so on. The position of the mouse is continually tracked and available in variables **mouseX** and **mouseY**, updated every time the **draw** method is executed.

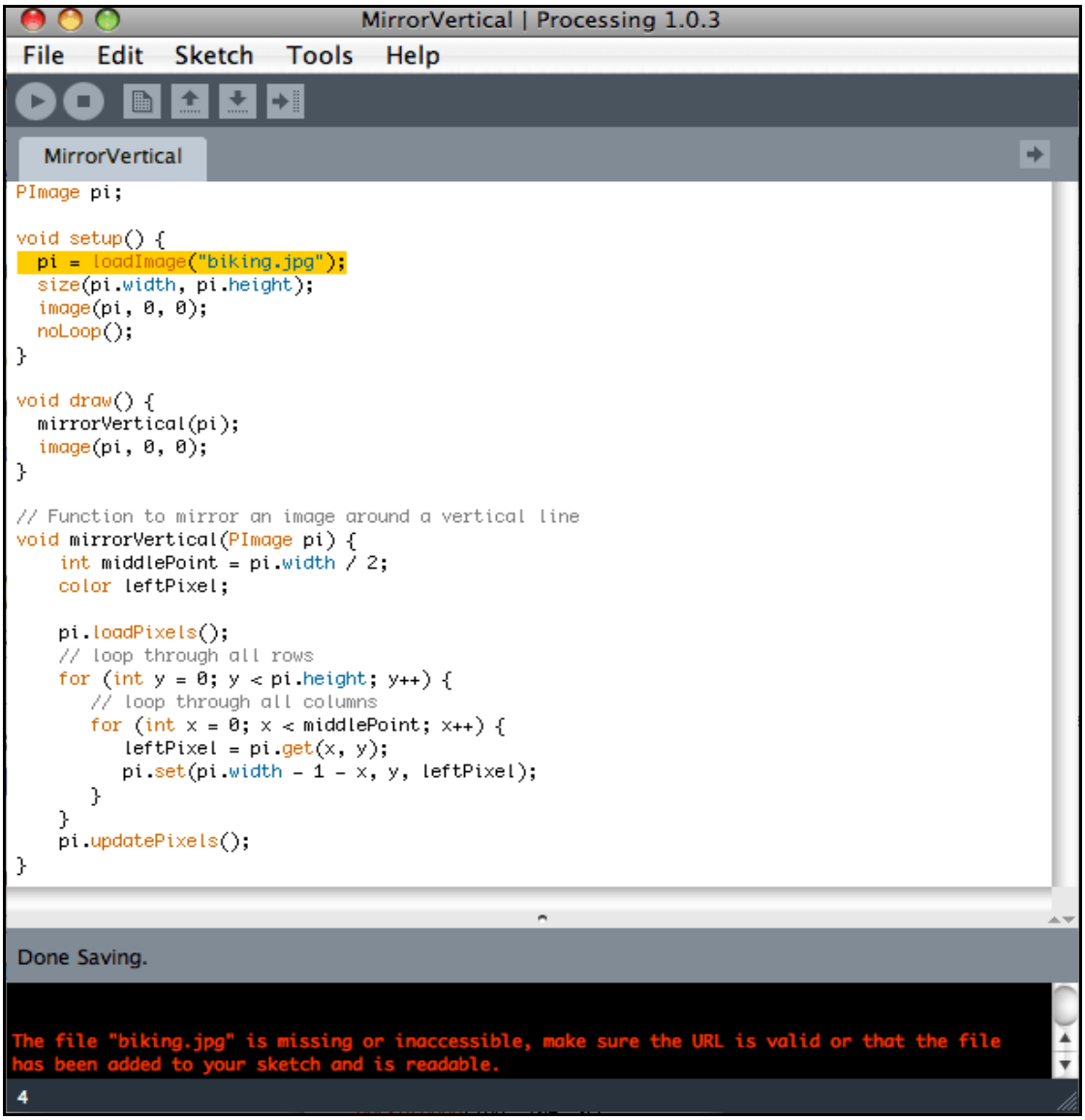


Figure 1: The Processing IDE in action

The Processing community has contributed a large number of libraries that are available from the main website [7]. There are libraries for extending the range of media to be handled to webcam video, movie file formats, digital audio, GUI elements, advanced graphics formats such as SVG, and networking. Libraries are used in a sketch simply by adding a Java import statement to the sketch for each library used (classes from the Java class library can also be imported).

PURE DATA

Pure Data, or just Pd, is a graphical programming system for creating and manipulating audio, video, and graphics in real-time [1]. Functional units for generating and manipulating media, called *objects* in this system, are represented as simple icons. Connections between objects represent the flow of data through the units; the resulting network is called a *patch*. Pd supports subprograms through *abstractions*, in which a patch can be given its own name and input and output objects, and then used as an object in its own right in other patches.

Figure 2 demonstrates a patch for loading and rendering a video onto a three-dimensional shape (in this case, a cube) while producing an audible tone. The pitch of the tone and the rotational position of the cube can be controlled using the horizontal sliders. This patch uses a library called GEM to provide objects for working with video and graphics.

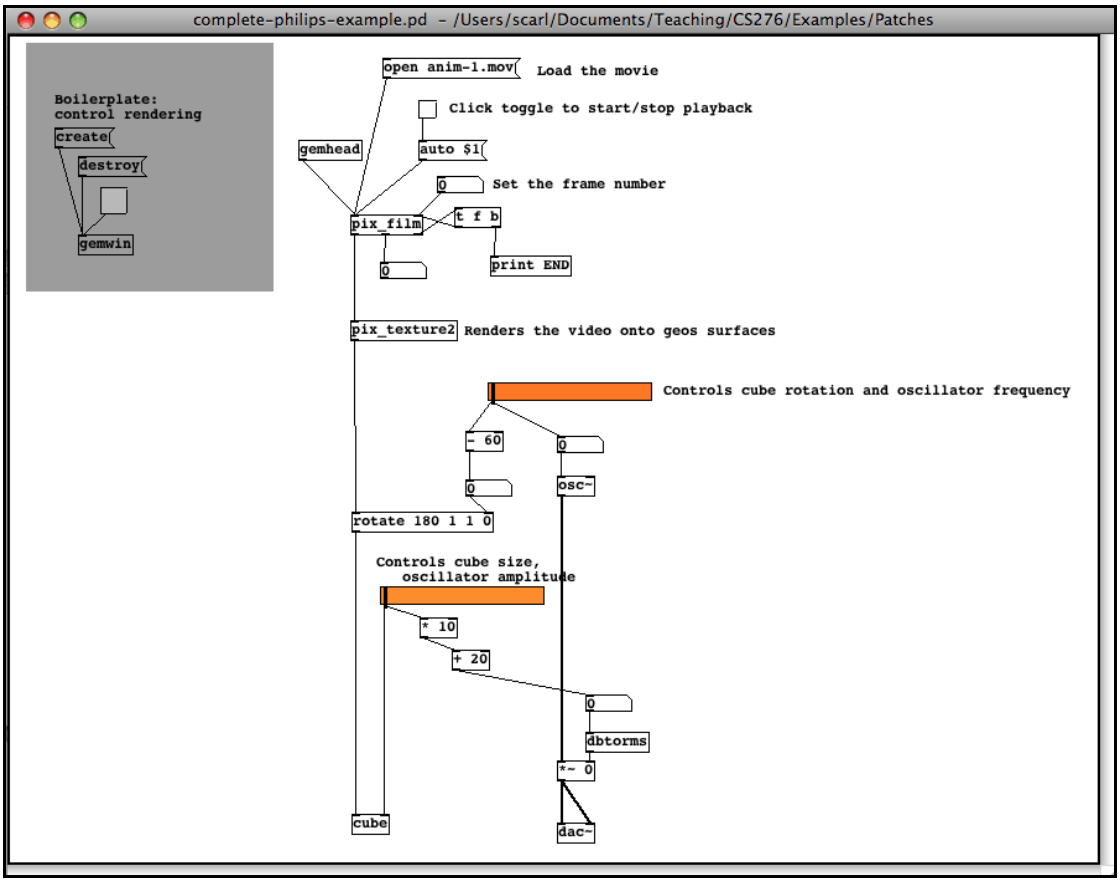


Figure 2: A Pure Data patch combining audio and video processing

OUTLINE OF THE COURSE

The general outline of the course is to study the manipulation and creation of digital images, digital audio, and video. We first use Processing to study algorithms for simple pixel-based manipulations of images taken from digital cameras or the web, often from the students' own Facebook pages. We then briefly cover techniques for hiding secret data in images (steganography), ways of incorporating text and font manipulations in graphics programs, and time-based animations. Students are given weekly assignments to practice each technique and are assigned a midterm project. In the midterm project they apply several algorithms to images of their choice to produce artwork in poster format on any topic of interest to them. As an example, a student studying Russian manipulated examples of Soviet propaganda as commentary. The students are asked to share the results of their explorations on the class website.

After learning the basics of digital audio, students use Processing to generate musical sounds by combining *unit generators* (which produce sine waves, pink noise, and other waveforms), filters, and envelope generators using algorithms such as *additive synthesis* and *wavetable sampling*. They thus gain experience with the basic tools of classical electronic music [8].

The Pure Data system is introduced after covering these simple sound synthesis techniques in Processing. We start by recreating the synthesis algorithms in Pure Data to introduce the system and important objects. It is useful to have the students develop a Processing sketch and a Pd patch for performing more or less equivalent operations and compare the text-based program to the graphical one. The patch often allows them to better visualize what is happening in ways that the syntax of the textual language sometimes obscures.

We return to Processing to show how easy it is to extend algorithms for manipulating pixels to video by viewing video as streams of images. While this can be done by loading and manipulating Quicktime-compatible video files, we prefer to work with video feeds from computers with cameras, either built-in or connected through a USB or Firewire port. When students are comfortable manipulating video, we introduce basic techniques for tracking different types of objects in the feed as described in the *Processing* textbook [9]. After a single lecture on video processing in Pd, every student was able to create a unique video-processing patch in that system.

We also study rudimentary inter-process communication, using the Open Sound Control (OSC) networking protocol to send information from a sketch in Processing to a Pure Data patch for further manipulation. Since this involves communicating between processes or across machines, we are able to discuss networking. A typical application is sending data produced by a sketch to Pd for visualization. Another would be sending musical data across networks to be performed by different software "instruments."

The final project in the course asks the student to produce a work combining at least two of the three media types studied, presenting a story, idea, or argument, in as concrete or abstract a fashion as desired. This open-ended strategy has been very effective as students were uniformly ambitious in their ideas. For example, one student created a visualization while playing an MP3 audio file using frequency data (we did not cover the Fast Fourier Transform class that made this possible in lecture). Another created a "light

graffiti" system for writing "graffiti" on a wall using a projector, a laser-pointer, and a computer with a built-in camera.

DISCUSSION

Students in our course find the simplicity of Processing enabling. Because they have already used Java or JavaScript, they quickly move past the language syntax to the features of the API and begin doing creative work in the first two to three weeks of class. This motivates them to go beyond the lecture and learn the features of the API that they find interesting on their own. For example, each time the course has been taught some students learned the coordinate-system transformation operations on their own and used them in sketches in advance of the material being covered in class. Freely mixing Java classes with Processing sketches is a more advanced use of the environment, but when one student wanted a dynamically changing bar for showing progress or variation, he simply coded it up as a Java class and included it in the sketch. Processing code can create Java objects and invoke their methods, and the Java code can call the Processing API, which simplifies coding the GUI elements.

Seeing students push beyond the basic course requirements in turn motivates the instructor, who can spend less time talking *about* useful features of the language and more time showing interesting ways to use them. It then becomes easy to incorporate more active learning into the classroom by showing a particular technique, having the students experiment with it, and asking them to demonstrate their sketches to the class.

There are some disadvantages to using the Processing environment. For example, there is no interactive evaluation such as one might find in IDEs such as JES [6] or DrJava [3]. This means that sketches must be written to operate on a specific media file; adding support for selecting a file requires working with the Java Swing dialog box classes, which we don't cover in this course. However, we have begun looking into developing our own library to provide this capability.

The basic concepts needed to create patches in Pure Data is very easy for students to grasp, but because of the way the documentation is structured, students find it more difficult to figure out how to use more advanced objects. The system is not partitioned into groups of related objects; every object must either be discovered on its own, bereft of context, or the student must work through an extensive help system before understanding how some objects work together. This situation is addressed by providing tutorials showing students how to develop a personal library of useful patches using the most common objects.

OUTCOMES AND FUTURE DIRECTIONS

Our media computation course has attracted students majoring in physics, economics, and computer science, an Academic Technology Center staff member, and a math professor. Enrollment remains modest, which is typical of our upper level courses, but has increased every year. While our goals for reaching more students from the fine arts have not yet been realized, we have had two students pursue independent projects in multimedia programming.

Currently, Processing is being taught in two sections of our introductory (CS 0) course as an experiment. This fall it will be taught to students new to programming in Art 231: Digital Media. We are developing a project requiring collaboration between students taking this art course and our media computation course. We continue to pursue ways to use these cross-discipline tools in our curriculum and to build interdisciplinary collaborations that benefit our students.

REFERENCES

[1] About Pure Data - PD community site, puredata.info, retrieved October 15, 2007.

[2] ActionScript.org Flash, Flex and ActionScript Community - Tutorials, Support, Open Source & More, www.actionscript.org, retrieved May 07, 2009.

[3] Allen, E., Cartwright, R., Stoler, B., DrJava: a lightweight pedagogic environment for Java, ACM SIGCSE Bulletin, 34, (1), 137-141, 2002.

[4] Carl, S., Pond, G., Building Collaborative Learning in the Arts and Sciences, Proceedings of Visual and Computational Teaching and Learning: a conference for educators, www.cofc.edu/~thinking/docs/T=vc%5e2.2007.Proceedings.pdf, retrieved October 17, 2008.

[5] Forte, A., Guzdial, M., Computers for Communication, Not Calculation: Media as a Motivation and Context for Learning, Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), January 05-08, 2004.

[6] Guzdial, M., Forte, A., Design process for a non-majors computing course, Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, February 23-27, 2005.

[7] Processing 1.0, www.processing.org, retrieved May 07, 2009.

[8] Puckette, M., *The Theory and Technique of Electronic Music*, Singapore: World Scientific Publishing Co., 2007.

[9] Reas, C., Fry, B., *Processing: a programming handbook for visual designers and artists*, Boston, MA: MIT Press, 2007.

EBAY, ITUNES, AND PROPOSITIONAL LOGIC: COMPARING EXPRESSIVENESS OF DIFFERENT QUERY LANGUAGES*

Ian Barland
Radford University
Radford,VA 24142
832-655-5474
ibarland@radford.edu

ABSTRACT

We outline a series of exercises that relate propositional logic to various straightforward query languages as used by some popular programs. The exercises naturally motivate the use of boolean algebra to translate between equivalent formulas (including a practical application of converting a formula to conjunctive normal form). They also illustrate connections between topics student see in logic/discrete math, programming topics, and topics in design.

1. INTRODUCTION

Logic plays a fundamental role in computer science, yet it is often presented to computer science students as disconnected from computing. For example, a typical problem might be "Show $(p \to q) \wedge (p \to r)$ is equivalent to $p \to (q \wedge r)$ using boolean algebra." Some exercises might also relate propositional formulas to programming, perhaps asking students to rewrite conditions found in if statements. Such exercises are important, but it is easy to see why they might not seem strongly related to any of the programming courses a student is taking concurrently.

Sections 2 through 4 present exercises motivated by how propositional logic is used as a query language in several popular applications: iTunes, an email client, and eBay. In all cases, the applications use something *similar* to propositional logic to specify their queries, but with tangible differences. As students write (essentially) propositional formulas in the dialect of these programs, they discover how apparently minor differences can lead to various shortcomings (and, perhaps, how to redress those shortcomings). In doing so, students are exposed to how formal logic relates to real-world applications, as

well as how varying syntax can affect expressibility. (In particular, eBay's syntax motivates a real need for converting a formula to conjunctive normal form.) By showing logic as a query language early, it also strengthens the connection between logic and the SQL they will use in a database class.

These exercises are suitable for a series of related homework problems, or as the basis of one or two lectures when discussing propositional logic. They augment standard topics neatly, displacing little-to-no traditional material. The topics, as homework exercises, can be freely adapted from the author's web page [2]. In Section 5 we summarize how these problems relate logic to issues such as expressiveness, language design, and user interface design.

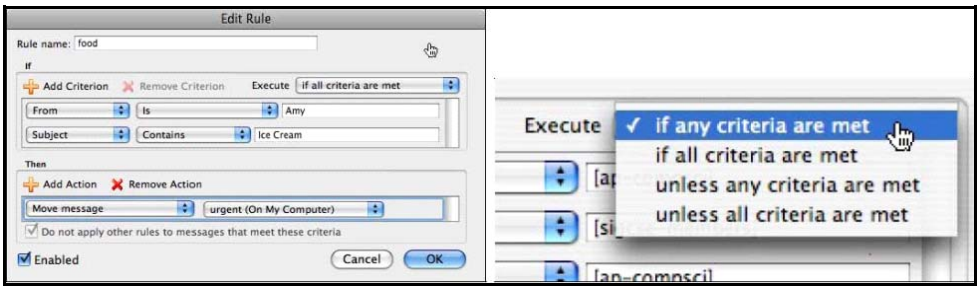


Figure 1: A gui dialog for a compound rule, in Microsoft Office's mail program. The drop-down menu (right) showing the (only) ways of combining atomic rules.

2. MICROSOFT OFFICE MAIL RULES *VS.* PROPOSITIONAL LOGIC

Many e-mail programs, such as Microsoft Office's Outlook, allow the user to specify "Rules" for automatically filing incoming emails [4]: "If the 'from' field is 'Amy' and the 'subject' field contains 'Ice Cream,' then take some action (perhaps, place the email in the folder 'urgent')". Rules are queries over the database of the user's songs. This database view lets students see propositional formulas as functions that return a set of songs, relating logic to their programming classes. Atomic queries are those like "the 'from' field contains 'Amy'." *Compound* queries can combine the atomic queries. However, compound queries are restricted to either one big conjunction, or one big disjunction (see Figure 2).

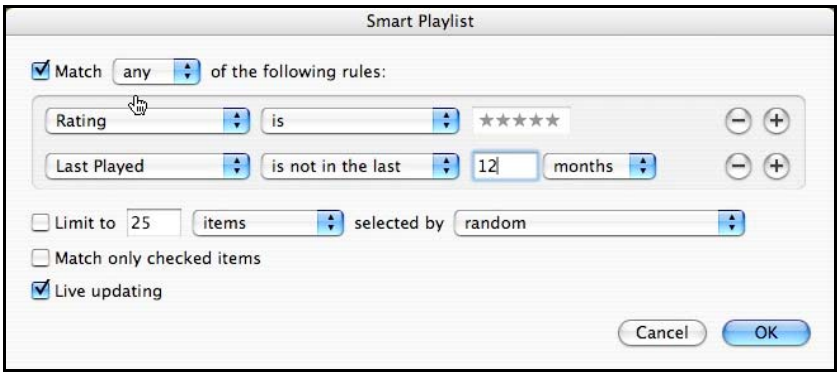


Figure 2: A gui dialog for a smart playlist in iTunes; the drop-down menu allows only "match any" or "match all" of the sub-queries.

It is not difficult to get students to want to create queries that include disjunctions *and* conjunctions. However, it turns out that *there is no way*[1] *to write a mail rule of the form* $a \wedge (b \vee c)$, in Microsoft's Outlook. The query language is strictly less expressive than propositional logic.

3. ITUNES PLAYLISTS *VS.* PROPOSITIONAL LOGIC

Music-playing programs, such as iTunes[1], let users manage their collection of songs, where songs are objects with fields such as name, artist, genre, length, last-played-date, *etc.* Such programs let users specify not only playlists (fixed sets of data), but also "smart playlists" —queries over the database of the user's songs.

In iTunes, atomic propositions can include:

- whether a song's last-played date is within the last (say) 11 days,
- whether its genre is (say) Math Rock, or
- whether it is on the existing playlist "beach music" (where the user has created that playlist themselves),

and a wide variety of other conditions. As in Outlook though, *compound* queries, are still restricted to either one big conjunction, or one big disjunction (see Figure 3). Having students think up *plausible* queries requiring both "and" and "or" is a good exercise in itself.

However, in iTunes' query language, there *is* an indirect way to create queries involving both disjunctions and conjunctions: by creating a "helper smartlist" and naming it (say) "Foo", and then creating the real smartlist and including "is on the playlist Foo" as one of its conditions. That is, to create a query corresponding to $a \wedge (b \vee c)$, we create $Foo = (b \vee c)$ and then our real query is a $a \wedge$ (playlist = Foo). Students readily recognize

[1] It is left as "a fact proven by logicians" that $a \wedge (b \vee c)$ is not equivalent to *any* possible formula involving only conjunctions, literals, and negated literals.

this as a form of programming; in fact "Foo" is essentially an auxiliary helper query. This shows that *adding auxiliary names increases the expressive power of our query language*.

This trick also reinforces the notion of writing the helper-methods (procedural abstraction) in conventional programming courses. We can also note that the language of iTunes only allows global variable names, and we must clutter our sidebar with all our helper-variables.

One sharp upperclassman, seeing playlists described this way, immediately wondered about cyclical dependencies, and whether Russell's paradox could be created: a playlist "Baz" defined as "all songs which are not on the playlist 'Baz'." Students can be asked how, as a real-world program, does iTunes *actually* prevent such a recursive playlist-definition from being created (either directly, or as a multi-step mutual recursion)? The takeaway point is that the Apple engineers, who were probably thinking mostly in terms of designing a GUI interface, must be familiar with these issues of logic if they wish to ship paradox-free products.

4. EBAY QUERIES *VS.* PROPOSITIONAL LOGIC

On the auction website eBay[3], users can write search queries for all pages which contain a certain word. They can also combine queries by using $\wedge$, $\vee$, and $\neg$. Of course, eBay's syntax is slightly different: $a \wedge b$ is written as "*ab*" (with a space for conjunction), and $a \vee b$ is written as (a, b) (with parentheses and a comma, for disjunction).

One might expect that it is trivial to convert between a propositional logic formula and eBay's syntax. However, it turns out that eBay's syntax does not actually let you compose formulas: $a \wedge (b \vee c)$ should translate to "(a,b c)",but this eBay query does not actually give the desired results.

For example, Figure4 shows a series of queries: "pilaf" returns four matches, and "superb bowl" returns 104 items. So one would expect "(pilaf, superb bowl)" to return between 104 and 108 items, instead of the 68420 it actually returns[2].

After some experimenting, it can be confirmed that while a conjunction inside a disjunction doesn't work on eBay, the other way around does. So, using the rules of boolean algebra to re-write the formula as "$(a, b)(a,c)$", we get the correct search results ("(pilaf, superb) (pilaf, bowl)" returning 108 items, in Figure 4). This is the first time this author has ever *actually used* conjunctive normal form for something practical. It is also an example of how the implementors of eBay's search page may *not* have been well-enough trained in their undergraduate discrete math or logic course.

5. FURTHER THOUGHTS

The central themes in the above examples are as follows:

[2] The screenshots in Figure4 are older; more recent eBay searches suggest that the query still doesn't work as expected, but in a way that returns too-few results rather than too-many.

- Discrete math topics permeate programming (sets, functions returning sets, recursive definitions, the use of helper functions).
- Language features influence expressiveness: In fact, some languages might be incomplete — unable to express all possible formulas (such as Outlook's mail rules).

  Conversely, are we really sure that propositional logic is complete? Are we certain that AND, OR, NOT are enough to express *any* boolean function? This question might lead to discussion of representing formulas as truth tables, representing a row of a table as a conjunction, and minimal sets of connectives.
- The design of the GUI elements used by many programs (including iTunes and Outlook) impacts what sort of queries can be made easily, or made at all. Is this trade-off worth it? Do users *want* to create queries that use both conjunctions and disjunctions, and is it worth adding complexity to the user interface?

By emphasizing how logic relates to programming and design, students can better appreciate how their discrete math class is relevant to the rest of their computing curriculum.

REFERENCES

[1] APPLE COMPUTER. iTunes product page. www.itunes.com/.

[2] BARLAND, I. Discrete math exercises. www.radford.edu/itec122/2008fall/Homeworks/Hw02/hw02.html.

[3] EBAY. eBay home page. www.ebay.com/.

[4] MICROSOFT. Tips for managing your e-mail using rules. http://office.microsoft.com/en-us/outlook/HA010173281033.aspx.

Figure3:A series of four eBay searches, with an unexpected 3rd result

INCORPORATING ETHICS INTO THE COMPUTER SCIENCE CURRICULUM: MULTIPLE PERSPECTIVES*

PANEL DISCUSSION

Frances Bailie (Moderator)
Iona College

Smiljana Petrovic
Iona College

Keitha Murray
Iona College

Deborah Whitfield
Slippery Rock University

Professional organizations in computer science demand that computer professionals adhere to a strict code of ethics. The Association for Computing Machinery (ACM) promulgates commitment to the ACM Code of Ethics and Professional Conduct for all its members and encourages the integration of computer ethics into the computer science undergraduate curriculum. ABET, Inc., the leading accreditor of computer science programs, highlights the importance of computer ethics in one of its criteria for all computing programs: An understanding of professional, ethical, legal, security and social issues and responsibilities. Therefore, it is imperative that graduates of computer science programs not only have a recognition of the conduct that will be expected of them as computer science professionals but also of the profound effects of computing in today's world. Computer science faculty members are aware of the necessity to engage students in vital discussions of the ethical issues that impact society but often find it difficult to find meaningful ways to do so. Consequently, many schools do not adequately address computer ethics in their curriculum. This panel will present specific initiatives to address the ethics of computing in a range of courses. Panelists will discuss student reaction and interest in the variety of methodologies presented and will assess the effectiveness of these initiatives. Audience members will be encouraged to share their own experiences with incorporating ethical issues into the computer science curriculum.

METHOD ASSUMPTIONS IN OBJECT-ORIENTED PROGRAMMING*

Robert Noonan
Department of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795
757-221-3465
noonan@cs.wm.edu

ABSTRACT

The paper discusses a programming assignment in a junior-level *Programming Languages* course whose main effect was to clarify the importance of explicitly stating method assumptions in object-oriented programming. We argue further that where the cost of checking a method assumption is small, it should be the responsibility of the supplier class, rather than the client class, to enforce the assumption, quietly if possible. The student assignment which clarified this issue is presented and discussed.

INTRODUCTION

For some time my colleagues and I have been dissatisfied with our students' ability to write good object-oriented programs. As such, we are always on the lookout for devices that will force our students to write better code in an easily identifiable way.

Last year we read an article on object calisthenics [1] that resonated with us. Many of proposed rules were trivially quantifiable, although their relevance to code quality was not as easily established. In the next section, we briefly explain the concept of object calisthenics as it relates to the outcome discussed here. Next we state the problem used as the assignment. The third section explores the assignment, including an algorithm for solving the problem. The fourth section examines the problems that the students encountered in implementing the algorithm. This led to a surprising result about the role of method assumptions. Finally, we summarize our conclusions.

OBJECT CALISTHENICS

According to [1], there are nine rules for object calisthenics; "if you write 1,000 lines that follow all these rules, you will find that you have created something completely different than what you expected." After some experimentation, the author modified some of these rules. Here we present only the rules that are relevant to our findings. In presenting these, we use the original numbers and the original statement. What follows is the rule as given to the students.

1. Use only one level of indentation per method.

Do not nest control structures. A loop may not contain another loop; an *if* statement may not contain another *if* nor a loop. The only exception is that a loop may contain a single *if* statement.

2. Do not use either the *else* or the *switch*.

3. Keep all entities small.

Limit your code to no more than 7 lines per method, 55 lines per class, and 10 classes per package. Note that an application such as Java NCSS will compute these values.

Adoption of the above rules had a pronounced effect on both the author's and the students' coding styles. Interestingly enough, our experiment has not yet come to any conclusions about the usefulness of object calisthenics in promoting object-oriented thinking.

Having decided what we hoped to accomplish, we then set about to find an appropriate assignment. The program had to be large enough to warrant using classes and small enough that it could be done in the last 3 weeks of the semester. We present the object-oriented assignment next.

THE PROBLEM

A Sudoku board is a 9 x 9 grid of cells. In addition to rows and columns there are nine 3 x 3 boxes. A puzzle initially has some of the cells set to a digit in the range 1:9 (1 through 9). A solution to the puzzle has each digit in the range 1:9 appear exactly once in each row, column, and box. The puzzle shown in Figure 1 has exactly 1 solution, and is thus designated a *valid* puzzle.

	1	2	3	4	5	6	7	8	9
1	7			5	8	3			6
2			6			1	4		5
3		5	2			6		8	3
4	3			2			9	5	8
5	5				7	8		6	
6	6	4	8		1		3		
7		6		8		2	5		
8			3	1	5			7	2
9	2	1	5	6				3	

Figure 1: An Easy Sudoku Puzzle

If the rows and columns are each numbered in the range 1:9, consider the cell numbered (6, 4) in Figure 1. There is a 1 in its box, a 2 in its box and column, a 3 in its row, a 4 in its row, a 5 in its column, a 6 in its row and column, a 7 in its box, and an 8 in its row, column and box.

Thus, if we store in each cell without a value a set of *candidate* values and eliminate from those candidates all of the values in the same row, column and box (the *neighborhood*) of the cell, we are left with only the value 9 (termed a *singleton*). Any non-value cell whose set of candidate values is a singleton can only be assigned that value. Similarly, if any non-value cell has no values in its candidate set, then the current puzzle has no solution. Thus, a cell either has a value in the range of 1:9 or has no value but instead contains a set of *candidate* values, which is computed by taking a set consisting of {1, 2, ..., 9} and removing any value which occurs in its neighborhood. The *neighborhood* of a cell *s* consists of all cells that are in the same row, column, or box as *s* excluding *s* itself.

THE ASSIGNMENT

We wanted the assignment to be an exercise in simple information management, so as part of the assignment, we gave the students the following *work list* algorithm:

1. Read a puzzle description and convert it to a board *x*.
2. Set the *solution list* to empty.
3. Set the *work list* to empty and add the board *x* to the *work list*.
4. While the *work list* is not empty:
    a. Choose and remove a board *b* from the *work list*.
    b. In the board *b*, find a non-value cell *s* that has a minimum size candidate set *c*. That is, for all non-value cells *t* in *b* with candidate sets *d*, size(*c*) <= size(*d*).
    c. Assume that the size(*c*) = *n*, i.e., *c* = {*c[1]*, …, *c[n]*}. Generate *n* new boards *b[i]* from *b* by:
        i. Set the value of square *s* to *c[i]*, where *c[i]* in *c*.
        ii. Update the candidate lists in all non-value cells in the neighborhood of *s*.
        iii. If board *b[i]* has no remaining non-value cells, add *b[i]* to the *solution list*.
        iv. Else if *b[i]* has a non-value cell with an empty candidate list, then discard *b[i]*, as it has no solution.
        v. Else add *b[i]* to the *work list*.

Upon termination of the loop, the solution list contains the solutions to the puzzle. The puzzle is *valid*, if the list contains exactly one solution. Otherwise the puzzle is invalid.

However, as stated here, with the addition of an algorithm to compute the neighborhood of a square (which was also given to the students), the problem is an exercise in simple information management. The assignment was used in a junior-level *Programming Languages* course, which had CS2 as a prerequisite. Many of the students struggled mightily with this assignment, a topic we explore in the next section.

MISTAKES AND LESSONS

We expected that the students would easily solve the assignment, and that the data collected would tell us something about how post-CS2 students develop an object-oriented program. Most of the students had roughly the same set of classes. Since we provided the interesting algorithms, we were very surprised with the difficulty most students encountered in developing and debugging their implementations. Student mistakes which had nothing to do with either the limits imposed by object calisthenics or method assumptions are ignored here.

Object calisthenics played a direct role in influencing the implementation of the solution. For example, the work list algorithm presented in the previous section violates both rule 1 (no nested loops) and rule 5 (only small methods). Steps 4(c)iii-4(c)v is an if-then-else, which violates rule 2. Some of these issues can be solved by refactoring [2].

By the term *method assumption* we mean any assumption that the method makes about its input. In *design by contract* [4] these assumptions would be termed *preconditions*. It is our contention that students need to be trained to explicitly state these assumptions, even when they *do not matter*.

For example, for this problem the constructor for the **Board** class had a string parameter that specified the puzzle to be solved. This constructor assumed that the string had **exactly** 81 characters, one character for each square or cell. The values 1, ..., 9 were represented by the characters 1, ..., 9. To accommodate a number of distinct Internet puzzle sites, a non-value cell could be a character chosen from a variety of characters, including a zero, a space, a period, a hyphen, etc. However, in all cases the character was a printable ASCII character. Later during program development, this routine was reused.

One of the problems that taxed the students occurs in step 4(c), where they have to clone a copy of the existing board. Many of the students quickly realized that the board's constructor could be combined with the board's **toString** method to effectively create a clone. However, the **toString** method, since it was primarily a debugging method, added an end of line character to each board row for the sake of readability. So the output of the **toString** method had 90 characters consisting of the digits 0-9 plus a linefeed character, not the 81 characters expected by the constructor. This led to a subtle bug which resulted in a running algorithm which gave the wrong answers. This bug was caught early by the instructor because his constructor method asserted that that the string length was 81.

In creating classes for both value and non-value cells, many students planned for only value cells being asked to return a value. In this case, when asking for a value, you would assume that the program filters out non-value cells, since it is not clear what value they should return.

However, our students are taught that the first thing they should do in creating a class is to write both its constructor and its **toString** method, the latter for debugging purposes. That means that both value and non-value cells had to return a value; to do otherwise would break the board's **toString** method. This requirement meant that non-value cells could be asked for a value, with the student implementations returning a zero, representing no value. This led to problems elsewhere in the implementation.

One of these problems was that a zero value could be inadvertently entered into a candidate set for a non-value cell. This was an uncommon problem, but one which was easily avoided. It resulted from the fact that a non-value cell could be legally asked for its value with most student implementations returning the value zero. The solution was to have the candidate set class provide its own **add** method, which merely rejected such values and returned false. We use the term *quietly* for this approach, since an assertion is not violated nor an exception thrown. The incorrect value is merely rejected.

A similar problem occurred in setting a value for a non-value cell *s*. This value is then removed from the candidate sets of all non-value cells in the neighborhood of the *s*. In the final stages of the work list algorithm this value was sometimes the zero value, resulting in an infinite loop. Again, our solution quietly rejects an attempt to set a cell to a zero value. Note that this eliminates an *if* statement in the board constructor.

The most common problem encountered by the students occurs in step 4(b) of the work list algorithm. The step calls for finding a non-value cell that has a minimum size candidate set. In implementing this, most students assumed that the routine would be called only if there were at least one non-value cell, an assumption that when false led to implementations that ran, but gave an incorrect answer. In this case, the non-quiet step was taken of asserting that the number of non-value cells was greater than zero.

The other, but less frequent problem in implementing step 4(b) was accessing value cells and asking for the size of their candidate sets. A quiet solution is to have value cells return a value greater than the number of legal values, e.g., 10. This approach allows you to assert that the return value is smaller than 10; otherwise, there are no non-value cells. Note that this approach eliminates a filter on the type of cell.

A host of similar problems included the following. You had to ensure that a board inserted into the work list had at least one non-value cell, and that all its non-value cells had at least one candidate value. Also, you had to ensure that a board inserted into the solution list had only value cells. The solution here is the same one used for candidate sets; each had its own class to which an appropriate **add** method was supplied.

There is a definite pattern here to solving these problems. For each method the assumptions or preconditions are explicitly stated, with the supplier class enforcing the precondition. If possible, the enforcement should be done quietly; otherwise an assertion or an exception is used. Consider the code for steps 4(c)i-4(c)v:

```
private void generateBoards(Board b) {
    String str = b.toString( ); // for efficiency
    int s = b.minCandidateCell(b);
    for (Value v : b.candidates(s)) {
        Board board = new Board(str);
        board.setCell(s, v);
        solutions.add(board);
        worklist.add(board);
    }
}
```

The code appears as though the board is added to both the solution list and the work list. However, the add methods of the two classes involved insert the board *only if* its restrictions are met. Otherwise, the add methods silently do nothing (other than returning *false*), which is actually step 4(c)iv. The if-then-else structure called for in the algorithm

does not appear as such in the code, thus satisfying both rules 1 and 2 of object calisthenics.

Note that the philosophy advocated here with regard to method assumptions disagrees with that of the text [3] used in our software development course, which is also a junior-level, post-CS2 course. Because reuse is often not foreseen, we contend that methods should always explicitly state their assumptions. Our experience is that such an approach simplifies and speeds up debugging.

We contend that students should be required to make method assumptions explicit, and where possible, prevent erroneous use, quietly, if possible. Examples included the board constructor, and the **add** methods for candidate sets, the work list, and the solution list.

Sometimes it is necessary either to assert some condition, or equivalently, throw an unchecked exception if some input condition is not met. Examples included the board constructor, finding a cell with a minimum size candidate set, and setting a value.

CONCLUSIONS

What started out as an experiment in object calisthenics wound up illuminating the need to explicitly state method assumptions about their input parameters. We discovered that having stated these assumptions explicitly, the best course of action was to have the supplier class enforce these assumptions, in most cases quietly.

Despite these added checks, the resulting program had fewer statements, and a lower McCabe complexity [5] than the student programs without such checks. Furthermore, adding these checks detected or eliminated subtle run-time bugs, resulting in a faster time to completion. We strongly recommend that instructors enforce this discipline.

REFERENCES

[1] Bay, J. Object calisthenics. *The ThoughtWorks Anthology*, Dallas, TX: Pragmatic Bookshelf, chapter 6, 2008.

[2] Fowler, M. *Refactoring*, Boston, MA: Addison-Wesley, 2000.

[3] Horstmann, C. *Object-Oriented Design & Patterns*, 2nd edition, Hoboken, NJ: John Wiley, 2006.

[4] Meyer, B. *Object-Oriented Software Construction*, 2nd edition, Englewood Cliffs, NJ: Prentice-Hall, 1997.

[5] McCabe, T. A software complexity measure. *IEEE Transactions on Software Engineering*, *SE 2*, (4), 308-320, 1976.

REVITALIZING CS HARDWARE CURRICULA:

OBJECT ORIENTED HARDWARE DESIGN*

Ganesh R. Baliga, John Robinson, Leigh Weiss
Computer Science Department
Rowan University
201 Mullica Hill Road
Glassboro, NJ 08028
(856) 256-4805
baliga@rowan.edu, robinsonj@rowan.edu, weiss@rowan.edu

ABSTRACT

This paper describes a plan to integrate current digital hardware design technology into the computer science hardware curriculum. The project will approach digital hardware design from a perspective familiar to computer science students. Techniques featuring Object Oriented Design (OOD), Object Oriented Programming (OOP), code reuse, rapid prototyping, project centered learning, and visualization will be used to facilitate learning computer hardware and embedded systems. In particular, we plan to introduce digital systems design from a programming perspective using the VHSIC Hardware Description Language (VHDL) language, Field Programmable Gate Arrays (FPGAs), and modern development environments.

INTRODUCTION

The complexity of digital designs has increased drastically with the advance of the semiconductor process technology. Hardware Description Languages (HDL) such as Verilog and Very High Speed Integrated Circuit Hardware Description Language (VHDL) have been developed and used to manage this ever increasing design complexity. HDLs have become industry standard in digital design because of their simplicity and powerful modeling capabilities. Additionally, the languages have been standardized by the IEEE standardization process. This has lead to a plethora of vendors offering products that allow digital systems design using HDLs.

While reaching maturity in industry as a digital design tool, there are many benefits in using VHDL in academia as a digital system design teaching tool, especially for undergraduate courses [4,5,6,7,8]. The first is the availability of VHDL tools which students can use outside class to relieve the problem of school laboratory resource contention. There are many suitable VHDL programs that are offered as free student editions by many vendors, these free tools allow students to work asynchronously outside of the classroom on their personal computers. Eliminated are the issues associated with limited school laboratory resources. A second benefit is the focus of HDL learning topics. Computer science students require a more detailed understanding of computer architecture than a black box approach can provide. The goal of using HDLs in academia for beginning undergraduate computer science students is to provide a deeper understanding of computer architecture. HDLs allow a look at the details in the black box. Through HDLs, students are exposed to the simulators, available in most development packages, which allow them to understand and experiment with several low level computer architecture concepts [1,2].

The technological advances in digital design and embedded systems offer a unique opportunity for computer science educators, the chance to reinvent computer science curriculum and education by including modern digital design techniques. Digital design and embedded systems design is important to computer science curriculum, which must be rethought to encompass models and methods that allow computer science students to learn and understand the details in the black box. This project therefore seeks to bridge the gap between hardware and software design to produce a unified view that enables the co-development of systems using both hardware and software [3,5,11].

Current trends in industry [7] demand that Computer Science graduates have a fluency in modern digital systems design techniques and become agile product development and rapid prototyping team members. There is also a convergence of hardware design methodologies (Software Based) currently used by Computer Scientists and Electrical Engineers. Modern digital design techniques (Software Based) are essential to both Computer Science and Electrical Engineering curricula [9,10,11]. This provides a unique opportunity for computer science educators to produce graduates that are comfortable working in a realm traditionally reserved for Electrical Engineers and demands that Electrical Engineering curricula become more software centric. This novel approach to digital systems design will allow the students to apply the knowledge gained as a computer science major to the design and understanding of computer architecture.

Our vision is to provide a learning experience featuring a methodology that is "natural" to computer science majors allowing them to develop a deeper understanding of the design and implementation of digital systems. Computer Science majors learn to develop software using Object Oriented Design techniques (OOD), Object Oriented Programming (OOP), project driven learning, and visualization techniques. Object Oriented Hardware Design provides a novel approach to digital systems design as a natural progression allowing student's to apply the knowledge and skills gained as a computer science major to the design and understanding of complex digital circuits and computer architecture.

This paper describes a revitalization of the curricula associated with undergraduate computer architecture in the Computer Science Department, we propose a project to integrate modern digital hardware design technology into the curriculum while

integrating teaching techniques to improve student understanding of computer hardware and to increase higher level learning [2]. The goal of this project is to infuse modern digital hardware design technologies into our existing computer science curriculum. The impetus for this initiative is to improve, expand, and facilitate student learning about digital design, computer circuits and hardware, while capitalizing on the student's existing strengths in Object Oriented Design techniques. Our approach will include using software from Xilinx and Aldec, VHSIC Hardware Description Language (VHDL), and rapid prototyping boards equipped with Field Programmable Gate Arrays (FPGA's). Additionally, to facilitate a collaborative student driven learning to reinforce these concepts and practices, a dedicated Embedded Systems Programming and Design Lab (ESPDL) will be established.

THE CURRENT CURRICULUM

In their sophomore and junior years, Computer Science majors take a series of courses that cover computer hardware.

- Computer Organization - This course provides an introduction to computer organization. Students are exposed to the register level architecture of a modern computer and its assembly language.
- Principles of Digital Computers and Digital Design Laboratory - Digital Design Laboratory is taken concurrently with Principles of Digital Computers. These courses provide an introduction to the fundamentals of computer hardware systems.
- Operating Systems - This course focuses on the major issues that are involved in the design of modern operating systems.
- Advanced Computer Architecture - This is an advanced course in computer architecture designed to expand the knowledge gained by students in the Principles of Digital Computers course.
- Embedded Systems Programming - This is an advanced course that deals with software issues that arise in embedded systems.

COURSE DESCRIPTIONS

Computer Organization provides an introduction to computer organization. Students are exposed to the register level architecture of a modern computer and its assembly language. The topics include machine level data representation, Von Neumann architecture and instruction execution cycle, memory hierarchy, I/O and interrupts, instruction sets and types, addressing modes, instruction formats and translation.

The Principles of Digital Computers course provides an introduction to the fundamentals of computer hardware systems. The topics include digital logic, combinational circuits, sequential circuits, memory system structure, bus and interconnection structure, computer arithmetic and the ALU unit, I/O system structure, hardwired control unit, microprogrammed control unit, and alternative computer architectures.

Digital Computer Laboratory provides the student with hands-on experience in the design and implementation of digital components. Software from Xilinx Inc. is used to design, test, and implement digital circuits: Combinational circuits, sequential circuits, registers, counters, datapath, arithmetic/logic units, control units, and CPU design. The individual projects are student driven with some guidance by the instructor. Projects are prototyped on Field Programmable Gate Arrays (FPGA) to actually observe the laboratory exercises in real operation.

Operating Systems focuses on the design and functions of the operating systems of multi-user computers. Its topics include time sharing methods of memory allocation and protection, files, CPU scheduling, input-output management, interrupt handling, deadlocking and recovery and design principles. The course discusses one or more operating systems for small computers, such as UNIX.

Advanced Computer Architecture is an advanced course in computer architecture designed to expand the knowledge gained by students in the Principles of Digital Computers course. The topics include various performance enhancement techniques such as DMA, I/O processor, cache memory, multiport memories, RISC, pipelining, and various advanced architectures such as high-level language architecture, data-flow architecture, and multiprocessor and multi-computer architectures. This course also allows detailed examination of one or two contemporary computers.

Embedded Systems Programming deals with important concepts in developing embedded systems. Device drivers, multitasking with real-world constraints, task synchronization, device testing and debugging, and embedded software development tools such as emulators and debuggers are covered. These concepts will be applied to design and implement embedded software for on or more modest-sized embedded systems.

THE NEW CURRICULUM

First Semester - Computer Organization

Computer organization offers computer science students an introduction to the internal workings of a computer system. The major components of a computer system are described and the language that allows the components to interact as a system is explored, including topics covering the performance and optimization of a computer system. The traditional focus of this course is on the programming aspects of the hardware. The components are used as “black box” entities that direct the flow of information in a computer system. With the advent of relatively inexpensive hardware and software to design hardware, there should be a component in the traditional computer organization curriculum to provide students a hands-on view of the internal workings of the “black box”.

This project will integrate traditional computer science methodologies such as Object Oriented Design (OOD), Object Oriented Programming (OOP), project driven learning, and visualization techniques to allow the “natural” exploration of the details of implementation for the “black box” components introduced in computer organization. This will be accomplished by projects and lectures that allow the students to apply their computer science knowledge in understanding computer hardware components and the

software that allows communication between the components. An introduction to VHDL and FPGAs will allow the students to examine the details of implementation and experiment with the "black box" components taught in Computer Organization. A specific mini MIPS processor will be used in the four semester sequence being proposed. Computer Organization will introduce the architectural features of this processor.

Second Semester - Principles of Digital Computers and Digital Design Laboratory

Principles of Digital Computers is a lecture course that provides an introduction to the theory of computer hardware systems and the Digital Design Laboratory provides the student with hands-on experience in the design and implementation of digital components. We will alter the teaching of the laboratory course to emphasize using VHDL and continue the use traditional computer science methodologies such as Object Oriented Design (OOD), Object Oriented Programming (OOP), project driven learning, and visualization techniques in hardware design. A series of projects using VHDL and FPGAs will expose the students to digital systems design from a computer science perspective. Complex components such as multiplexors, counters, registers, arithmetic logic units, and memory will be used to construct datapaths. These datapaths will be combined with finite state machines to design a mini MIPS processor. Exploring the mini MIPS processor from the ground up will provide a detailed understanding of the components and inner workings of microprocessors and prepare the students to explore the topics introduced in the advanced courses.

Third Semester – Operating Systems

This course will build upon the concepts learned in Computer Organization. In a series of projects in this course, students will build various important operating system components such as the memory manager, scheduler, device driver etc. The projects will be built on an embedded platform (FPGA) that has the same mini MIPS processor that was explored in the prior courses. An appropriate toolchain that enables students to program operating systems code in a high level language such as C or C++ will be used.

Fourth Semester – Embedded Systems Programming or Advanced Computer Architecture

The Embedded Systems Programming course deals with software issues that arise in embedded systems programming and Advanced Computer Architecture is an advanced course in computer architecture designed to expand the knowledge gained by students in the Principles of Digital Computers course. A series of projects using the mini MIPS processor from the second semester courses and the appreciation of operating systems issues from the third semester will allow students to explore advanced topics. The systems will be prototyped on Field Programmable Gate Arrays (FPGA) to actually observe the laboratory exercises in real operation. Programs will be built using the same toolchain that was used in Operating Systems.

EMBEDDED SYSTEMS PROGRAMMING AND DESIGN LAB (ESPDL)

In order to facilitate the unified hardware/software approach to digital systems design, a design lab will be implemented. The Embedded Systems Programming and Design Lab (ESPDL) will allow to become engaged in digital system design and embedded programming by "doing" unified hardware/software design. This approach

will put less emphasis on just reading books and taking test and will make the students active learners. The Embedded Systems Programming and Design Lab (ESPDL) will also allow the students to learn asynchronously and work in groups on digital design and embedded system projects.

SUMMARY AND CONCLUSIONS

Computer science majors are taught to design software using techniques featuring Object Oriented Design (OOD), Object Oriented Programming (OOP), code reuse, rapid prototyping, project centered learning, and visualization. In addition, they are taught the theory, components, and design of computer hardware. Traditionally, hardware design was done by electrical engineers, but with the advent of HDLs, computer scientist can also design hardware. There is a convergence of hardware design methodologies used by electrical engineers and computer scientists. This provides a unique opportunity for computer science educators to produce graduates that are comfortable working in a realm traditionally reserved for electrical engineers. A learning experience featuring a methodology that is "natural" to computer science majors will allow them to develop a deeper understanding of digital systems. This novel approach to digital systems design will allow the students to apply the knowledge gained as a computer science major to the design and understanding of computer architecture. The curriculum described in this paper will enable students to bridge the gap between hardware and software design to produce a unified view that enables the co-development of systems using both hardware and software.

REFERENCES

[1] Areibi, S. , A first course in digital design using VHDL and programmable logic, *Proceedings of the Frontiers in Education Conference, 1*, 19-23, 2001.

[2] Black, M., Building a computer from scratch: a hardware lab sequence for computer science students. *J. Comput. Small Coll., 24*, (3), 32-38, 2009.

[3] Bouchhima, A., Chen, X., Pétrot, F., Cesário, W. O., and Jerraya, A. A., A unified HW/SW interface model to remove discontinuities between HW and SW design, *Proceedings of the 5th ACM international Conference on Embedded Software*, 18 - 22, 2005.

[4] Edwards, S. A., Experiences teaching an FPGA-based embedded systems class. *SIGBE, 4,* (2), 56-62, 2005.

[5] Hamrita,T., Potter, W., Bishop, B., Robotics, Microcontroller, and Embedded Systems Education Initiatives at the University of Georgia An Interdisciplinary Approach, *International Journal of Engineering Education, 21,* (4), 730-738, 2005.

[6] Henzinger, T. A. and Sifakis, J., The Discipline of Embedded Systems Design, *Computer, 40*, (10), 32-40, 2007.

[7] Jerraya, A. A., Long Term Trends for Embedded System Design, *Proceedings of the Digital System Design, EUROMICRO Systems,* 2004.

[8] Martin, F. G., Integrating hardware experiences into a computer architecture core course, *J. Comput. Small Coll., 21*, (6), 39-52, 2006.

[9] Patterson, A., Hennesy, J. L., Computer Organization and Design, Fourth Edition: The Hardware/Software Interface, Burlington, MA, Morgan Kaufmann, 2008.

[10] The Joint Taskforce on Computing Curricula, Computing Curricula 2001, *Journal on Educational Resources in Computing*, 2001.

[11] Vahid, F., Givargis, T., Embedded System Design: A Unified Hardware/Software Introduction, Hoboken, NJ, John Wiley & Sons, 2002.

THE ESSENCE OF OBJECT ORIENTATION FOR CS0: CONCEPTS WITHOUT CODE*

Raja Sooriamurthi
Information Systems Program
Carnegie Mellon University
Pittsburgh, PA 15213
raja@cmu.edu

ABSTRACT

Why is object-orientation so popular? Is it a fad or is there real value to developing software systems the object-oriented way? Given the emerging prevalence of computational thinking across the disciplines these are questions that a wide range of students are curious about. This paper describes our approach to providing a conceptual overview in a CS0 context of the essential ideas of and the value provided by object-orientation without resorting to code.

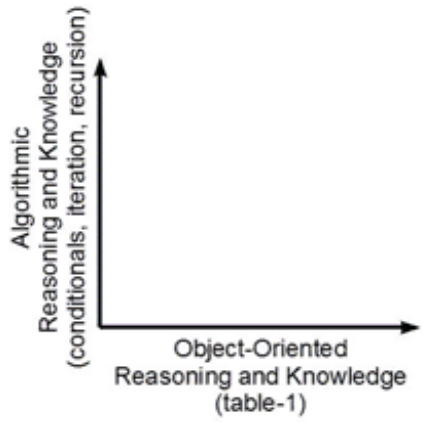
1. INTRODUCTION

For more than ten years the concepts of object-orientation have formed an integral component of the under-graduate curriculum. CS1 and CS2 courses introduce object-oriented programming with a variety of approaches[2, 5, 7, 10, 13]. While the pedagogical issue of teaching objects-early vs. objects-late is unresolved[11], object-orientation is well established as a predominant paradigm of software development. Why is object-orientation so popular? Is it a fad or is there real value to developing software systems the object-oriented way? If there is real value, what is it and how does object-orientation provide that value? Given the emerging prevalence of computational thinking across the disciplines[15] these are questions that a wide range of students—not just computer science majors but also engineering, information systems, science, business majors etc.—are curious about. This paper describes our approach to providing an overview of object-orientation in a CS0 context. Our approach is couched in an intuitive and qualitative discussion of the essential concepts of and value provided by object-orientation. These ideas are conceptually discussed sans code. Hence the discussion is

---

accessible to students with a wide range of backgrounds and limited prior development experience. We have successfully used this approach to several audiences: college freshmen, high school participants in outreach programs, professional technology users groups, graduate students transitioning from other disciplines, birds-of-a-feather session at CS education conferences, participants of the Java Engagement for Teacher Training (JETT) workshops for high school computer science teachers [14] etc. Whereas tools such as Alice, BlueJ, Greenfoot, jGrasp, Dr.Java etc. help enormously in providing hands-on exposure to object-orientation, the focus of the approach we summarize in this paper is to provide a bird's-eye conceptual overview of the essential ideas of object- orientation to those new to the field.

Algorithmic and Object-Oriented Reasoning: While the focus of our discussion will be on object-orientation, algorithmic reasoning is the bedrock foundation upon which one builds software development skills. These two critical skills of algorithmic thinking and object-oriented thinking are interestingly in many ways related. Yet they are also orthogonal and complementary. Algorithmic reasoning focuses primarily on data and functional abstraction and how primitive computational elements may be combined and composed by sequencing, alternation, iteration and recursion. Object orientation focuses more on identifying the nature of the problem, the main entities involved in the solution, their respective responsibilities and how they interact in a cooperative manner to obtain the computational solution [12, 16].



2. TWO FACETS OF SOFTWARE DEVELOPMENT

At the highest level, software development addresses two questions: (1) how to build the right system and (2) how to build the system right? Any paradigm of software development needs to support these two activities well. But what do these two activities entail?

Building the Right System: To build a right system means to build a system that meets the requirements of the client, i.e., the system does what a client would like the system to do and does it in the way the client would like it to be done. Perhaps the biggest challenge to meeting the requirements of a client is that rarely are the requirements predetermined and unchanging. At the onset of a project clients usually have a vague and ambiguous notion of what they want. As a result, during the construction of a software

system requirements change—new requirements are added and existing requirements are modified. A good software development process needs to be able to effectively accommodate such changes.

Building the System Right: Apart from functional requirements (what a system should do) there are numerous non-functional requirements (also known as quality attributes) such as usability, reliability, performance, supportability etc[1]. Of these, over the life time of a system, the most expensive quality is maintainability. Estimates are that as much as 80% of the cost of a system of is in maintenance. But the word maintenance, as applied to software, is bit of a misnomer in that it is not like house or automobile maintenance—there is no wear and tear. Rather maintenance is about fixing bugs and more often it is about enhancing the functionality of a system to do things above and beyond what the system was expected to do when originally conceived. Any software development process needs to offer good support for the evolution of a deployed system.

In both of the above the common core is managing change. This can be done in two broad ways (i) reduce the need for change and (ii) when change is inevitable make it easier to effect. Reducing the need for change is partly a function of understanding the domain and modeling the system requirements better. An effective mechanism for mitigating the amount of effort that needs to be changed is to build systems in an incremental, iterative manner with continuous/ regular feedback from the stakeholders[3]. But when change is inevitable it is desirable to reduce the impact of the change on the overall system. Change is initiated by the need to provide new functionality. There are two alternatives when we want enhanced functionality: (i) we need to modify pieces of the existing system or (ii) we don't modify existing pieces but just add new pieces to the system. A significant attraction of building systems the object-oriented way is that object-orientation offers good support for managing change in terms of building the right system and building the system right. The next several sections expand upon this in detail.

3. PROGRAMMING AS A PROCESS OF MODELING A WORLD

Object-orientation (OO) can help to reduce the need for change by helping us to better model the domain. Programming is a process of modeling a world. All paradigms of programming assist us with different aspects of this modeling process which is also the essential idea behind the process of abstraction[6, 8]. As a case in point, we discuss the very first killer application for PCs, the electronic spreadsheet. Starting with Visicalc, spreadsheets have modeled the information processed by accountants and the accounting processes they use. More generally, we humans are good problem solvers. A significant attraction of object- orientation is that it tends to model the way we humans solve problems. Consider the way we get things done in this world: Either we have the ability to do a task or we ask someone who has the ability to do it for us. For example, let us consider the following task: a professor wants to reserve a room to conduct an exam. Professors typically do not have the authority to make room reservations. Hence the professor would typically make a request to the department secretary. The secretary in turn forwards the request to the building manager who makes the room reservation. The reservation details are then sent back to the secretary and from there back to professor. We draw a parallel between that which we term as making a request in the real world and sending a message in object oriented parlance and emphasize that the way we do anything

in a pure OO way is by sending a message to an object whose responsibility it is to perform that action. The response to a request (message) is a method. The set of messages a person responds to is termed the interface (e.g., sit, stand are typically part of our interface but fly isn't).

An anthropomorphic view of reasoning wherein inanimate objects are also deemed to have the ability to respond to requests helps to develop the OO thought process. The animation in Disney movie classics such as Bedknobs and Broomsticks and Beauty and the Beast helps to illustrate this notion of anthropomorphisation. In this manner we give our students an intuitive introduction to various concepts of object orientation some of which are listed in Table 1.

Instance	class	message	method
inheritance	delegation	polymorphism	interface
abstraction	encapsulation	overloading	overriding

Table 1: Some of the foundational concepts of object-orientation. Each of these concepts are introduced to students in an intuitive manner using examples from outside the realm of programming.

Symbolic Math	$\sum_{i=m}^{n} i$	Textual Math	n sigma(i) i=m
Functional	sigma(m,n)	Message sending	m.sigma(n)

Table 2: Four alternative notations for the same concept of summing an interval of integers.

4. NOTATIONAL VARIATIONS AND THEIR SEMANTIC EQUIVALENCE

Barring a very few exceptions (CLOS being a prominent exception), most OO languages use the "dot notation" for denoting message sending. We have found this to be a useful context to discuss the power of notations in general and notational variations. Consider Table 2 which illustrates four ways of denoting the sum of all integers from m to n inclusive. Conventionally this summation is denoted in mathematics with the Greek letter S. But suppose we did not have the facility to graphically denote the Greek letter sigma (e.g., we were typing notes in plain text etc), then the textual math notation would meet our needs. Now suppose we had a further restriction on our notation system: we could only write linearly on a line. Then the functional notation would suffice. It is important to recognize that all three notations (symbolic math, textual math, and functional) are semantically equivalent---they denote the same value. Finally if we wanted to express the summation from m to n using the message sending notation of object oriented programming we get the last notation. In pure object oriented languages such as Ruby or Smalltalk one could actually express summation with this notation as in these languages integers are full fledged objects. We have found that appreciating the fact that the same concept could be denoted in multiple ways is a key step towards getting comfortable with the novel message sending notation of object-orientation.

As an interesting aside, it is useful to look at the linguistic typology and notations used in human languages. English is termed an S-V-O language as we write sentences of the form "Jack ate an apple" with the subject followed by the verb followed by the object. Many East Indian languages (e.g., Tamil), Korean, Japanese etc., are S-O-V languages. It is quite interesting to note that there are human languages that fall into one of the six possible permutations of linguistic typology! The rarest order seems to be O-V-S with only a handful of human languages. Students find it amusing that the artificial language Klingon (from Star Trek) was specifically designed to be O-V-S so as to be very different from English.

5. OBJECT-ORIENTATION: METAPHORS AND ANALOGIES

In this section we give a sample of a few more qualitative overviews of some essential concepts of object-orientation.

The Power of Inheritance: A question we ask in class is "do you know what an *okapi* is?" Most students do not and wonder whether it is a place, a fruit, an animal etc? Once we mention that an okapi is like a zebra and a giraffe students immediately get a mental model. If we were to ask how many legs[1] an okapi1 has, a reasonable answer would be 4 (correct). If we were to ask what color is the tongue of an okapi another reasonable answer would be pink, which is wrong as it turns out an okapi has a blue tongue! A simple example of over-riding a default.

As an another example of the utility of organizing knowledge in an inheritance hierarchy and using default reasoning we discuss the psychological experiments of Collins and Quillian[4]. Participants of the experiment were asked a series of questions and their reaction times were noted. A sample set of questions were: can a canary sing? Can a canary fly? Does a canary have skin? The reaction time of the participants were progressively longer for these series of questions leading credence to the fact that information about canaries, birds, and animals seem to be organized in an inheritance hierarchy.

[1] An Okapi is a peculiar animal discovered in the Congo rain forest only in the early 1900s. Amongst other oddities it is the only animal with a tongue long enough to lick its own ears http://en.wikipedia.org/wiki/Okapi.

The Flexibility of Delegation: While inheritance is powerful and with very little effort one can gain a lot of information, its disadvantage is its static nature. The inheritance links are traditionally predetermined and not flexible. The inflexibility of inheritance is akin to being required to ask one's parents an answer to a question when one's uncle may be more of an expert on that particular topic! An alternative to inheritance is delegation. While in most programming languages it requires more effort to setup it provides more runtime flexibility. A good example of the flexibility of delegation is exhibited in the TV game Who wants to be a millionaire?. The contestant has the option of dynamically choosing during the game (i.e., at run time) to whom to direct a question.

Interface vs. Implementation One of the core tenets of design is separation of concerns. In object-orientation it is useful to separate what we can ask an object to do (its interface) and how the object actually does what is asked of it (its implementation). What is the advantage of this separation in reasoning? In short: it fosters resilience to change. An interesting analogy of this resilience is contrasting the TV shows Law & Order and Seinfeld. Currently in its 19th season, Law & Order has had an interesting evolution. Over its long and successful run the show has changed: various actors have come and gone, but the popularity of the show has been undiminished. The audience tunes in for the roles played by the different actors (district attorney, police captain, detective, prosecutor etc.) and not necessarily the people who play the roles. Law & Order is a show programmed to its interface and is only loosely coupled to its implementation. As a result of which the show has been flexible and resilient to change. But a show such as Seinfeld is programmed to its implementation. The audience tunes in for the actors and hence the show would probably handle change less gracefully.

Overloading: Words in human language are overloaded i.e., the same word takes on different meanings in different contexts. The same request (message) can result in different responses (method) depending on the context. This notational convenience increases the usability of natural language. Consider the request open. Based on the context, the actual action executed is different. The same message has different methods. For example "opening" a door, a can of soda, or a pen are different actions. If our use of human language was not overloaded we would have to concoct unique words for each type of opening e.g., dopen, copen, popen! Object-orientation provides a similar convenience in the context of determining method names.

Responsibility Alignment: One of the most important skills of an OO designer is to properly align responsibilities with the objects that will perform them [9, 12, 16]. Misaligned responsibilities can lead to contorted systems. Two interesting examples of responsibilities that were re-aligned are (i) The introduction of penny postage by Sir Rowland Hill.[2] When postage was first introduced the recipient had to pay which led to many problems. The postal reform of 1840 switched the responsibility of paying for postage from the recipient to the sender leading to our current system (ii) in the early days of telephony the responsibility of actually placing a phone call was with a telephone operator. Subsequently this shifted to the call initiator via the introduction of the rotary dial and subsequent automatic exchanges which in turn led to the mass scaling of the

[2] Actually based on an analysis of the British postal system by computer pioneer Charles Babbage.

phone system. In a similar manner we use various analogies and metaphors for conveying the essential idea behind other OO concepts (Table-1) such as polymorphism, encapsulation, the Java construct of an interface (which is related to but slightly different from the more general notion of an interface) etc. (Due to space limitations in the paper we are not discussing them here.)

6. IT IS A MATTER OF TIME --- BUT WHOSE?

The time it takes to solve a problem is a valuable resource. Over the past couple of decades there has been an interesting shift as to which time is more valuable. In the early days of the field computer processing time was expensive. Hence the time measured was the interval between when the program was given to the computer and when it finished running. Now the more valuable resource is the programmer's time. Hence what is being measured is the duration of when the problem is given to the programmer and when the programmer develops the correct solution. Yet another reason for the popularity of object-orientation is that it contributes towards the productivity gains in programmer development time in terms of reuse, customization, frameworks etc. Languages like Ruby and Python add an extra dimension of productivity on top of their object-oriented features by being dynamically typed and thereby enhancing the productivity of a developer even more, albeit at a cost of execution speed. Then again, the critical question is not whether the program is fast, but whether the program is fast enough.

7. SUMMARY

While there are a number of reasons why object-oriented programming is a current dominant paradigm of software development, two prominent reasons are that object-orientation helps us to (1) better model our domain and (2) better manage change during the evolution of the system and after its deployment. We've discussed how we provide a conceptual 10,000 foot view of the essential ideas of object-orientation that is independent of any particular OO language. How effective is this approach? At present we have anecdotal feedback to support our approach. In both class-room as well as professional settings where we have delivered this content we have had very good interaction and feedback. To our amusement (and delight) we also see that some of our students have been using these metaphors and analogies during technical interviews to the mirth of their interviewers.

REFERENCES

[1] BASS, L., CLEMENTS, P., AND KAZMAN, R. Software Architecture in Practice, 2nd ed. Addison- Wesley, 2003.

[2] BRUCE, K., DANYLUK, A., AND MURTAGH, T. Java: An Eventful Approach. Prentice Hall, 2005.

[3] COCKBURN, A. Agile Software Development: The Cooperative Game, 2nd ed. Addison-Wesley, 2006.

[4] COLLINS, A. M., AND QUILLIAN, M. R. Retrieval time from semantic memory. Journal of verbal learning and verbal behavior 8 (1969), 240--248.

[5] DANN, W. P., COOPER, S. P., AND ERICSON, B. Exploring Wonderland: Java with Alice and Media Computation. Prentice Hall, 2009.

[6] HAZZAN, O. Reflections on teaching abstraction and other soft ideas. SIGCSE Bulletin 40,2 (2008), 40--43.

[7] KELLEHER, C., AND PAUSCH, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys 37,2 (June 2005), 83--137.

[8] KRAMER, J. Is abstraction the key to computing? Commun. ACM 50, 4 (2007), 36--42.

[9] LARMAN, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed. Prentice Hall PTR, 2004.

[10] LEWIS, J., AND DEPASQUALE, P. Programming with Alice and Java. Addison-Wesley, 2008.

[11] LISTER, RAYMOND, E. A. Research perspectives on the objects-early debate. In ITiCSE proceedings (2006), pp. 146--165.

[12] MEYER, B. Object-Oriented Software Construction, 2nd ed. Prentice Hall PTR, 2000.

[13] REGES, S., AND STEPP, M. Building Java Programs: A Back to Basics Approach. Addison-Wesley, 2007.

[14] SOORIAMURTHI, R., SENGUPTA, A., MENZEL, S., MOOR, K., STAMM, S., AND BORNER, K. Java engagement for teacher training: An experience report. In Proceedings of the Frontiers in Education Conference (2004).

[15] WING, J. M. Computational thinking. CACM 49, 3 (March 2006), 33--35.

[16] WIRFS-BROCK, R., AND MCKEAN, A. Object Design: Roles, Responsibilities, and Collaborations. Addison-Wesley, 2002.

PERSPECTIVES CONCERNING

THE UTILIZATION OF SERVICE LEARNING PROJECTS

FOR A COMPUTER SCIENCE COURSE*

Richard A. Scorce
Computer Science, Math & Science Division
College of Professional Studies
St. John's University
Queens, NY 11439
718 674-7245
scorcer@stjohns.edu

ABSTRACT

The utilization of a real life project for a Computer Science Database Management Course can provide students with additional learning experiences beyond the concepts contained in text books or instructors' lectures. In addition to the practical application of database design and development principles students can become especially involved and motivated since they will witness the outcome their work within a non-profit organization. At the same time that they will be absorbing the course material, they will be rendering their services to a non-profit organization which will also benefit from the development and installation of a "production" level database. Many advantages, disadvantages, cautions and insights, which were originally presented in a CCSC paper regarding service learning projects in December of 2008, are supplemented and expanded by the author's own experience in the utilization of a similar service learning project within a Database Management Systems course. Finally, instructors are encouraged, keeping in mind several important cautions, to investigate the utilization of a service learning project as part of their Computer Science course.

INTRODUCTION, BACKGROUND AND OBJECTIVE

One of the papers published in the *Journal of Computing Sciences in Colleges*, Volume 24, Number 2, December 2008, as part of the proceedings of the Southeastern Conference is titled "*A Service Learning Project for a Software Engineering Course*". Anne L. Olsen of Winthrop University, as author, discussed the employment of a service learning project within her Software Engineering course in place of utilizing a traditional instructor designed project. [1]

Service learning can be broadly defined as a student simultaneously devoting time to a service related activity, usually at a non-profit organization, while enrolled in a related college level course. The service learning project would be administered and managed by the student's course instructor and would pertain to the course's goals and content. [9] For example, Ms. Olsen's students provided software development services to a non-profit while they were enrolled in her Software Engineering course. Another example would be the provision of student services to La Fuerza Unidad, a Long Island, NY non-profit, regarding the development and installation of an "industry grade" database for the non-profit's usage and benefit. Service learning projects typically run for the duration of an entire semester or even two full semesters and must be carefully selected since they will involve significant interaction with the selected organization and can have an impact upon their operations.

In her paper, Ms. Olsen reiterated that most Computer Science courses utilize software tools in a lab environment and often employ instructor led and designed projects over the duration of the semester. [1] However, these projects do not fully reflect the real life experiences that students will face when assigned to their first project in an industry environment. Employment of service learning projects within the Computer Science curricula is a relatively new concept but rich in potential benefits. Furthermore, non-profit organizations can be an excellent source of service learning projects that may be incorporated within Computer Science courses.

Ms. Olsen's paper also covered the criteria for selection of such projects, their advantages and disadvantages, how they can fit within the semester's curriculum as well as the responsibilities and tasks of both the instructor and students. [1] At St. John's University, in the Fall 2008 semester, two similar introductory level Database Management classes from different schools worked in tandem to employ a common service learning project for a non-profit organization. The findings and observations published in Ms. Olsen's paper served as an excellent, comprehensive and insightful document which closely reflected the database classes' service learning project experiences.

This paper's objective is to provide additional thoughts, cautions, and suggestions regarding the selection of such a project, further discussion of the advantages and disadvantages regarding its utilization, and, most of all, words of encouragement regarding its application. IT professionals with many years experience in industry who have managed software development projects, including the design and implementation of enterprise wide databases, had found that there were several lessons learned "on the job" that were not previously made known in texts or classroom lectures. Should students be fortunate enough to be involved with a service learning project as part of their Computer Science courses, they will be better prepared for industry projects upon

graduation. This is especially true for honing their user interface skills, for analysis of problems and for developing potential solutions. [3] Their experiences will also benefit them regarding project planning, design tasks and the actual implementation of the service or product. An additional aim of this paper is to suggest, that with careful analysis and instructor management, many Computer Science courses can successfully utilize a non-profit related service learning project. It should also be kept in mind that the non-profit organization, often constrained by budget and resource limitations, will also gain from such a project.

THE DATABASE MANAGEMENT SYSTEMS COURSE

As described in our course syllabus, the Database Systems Management course at St. John's University's College of Professional Studies consists of an introduction to database fundamentals, design and implementation. It is a required course for Computer Science majors and coverage includes data modeling, database normalization, database design, Access, Structured Query Language (SQL), Oracle, multi-user databases and object oriented database design. Students are assigned several practical projects in creating ERD's (a diagram indicating the contents of a database and how the data relates to other data), normalizing tables (basically ensuring that all of the attributes or fields are placed in the appropriate tables or files), as well as creating and accessing databases in Access, SQL and Oracle. A semester long service learning project is also included which allows a student to provide database related services to an organization in need.

THE SERVICE LEARNING PROJECT

La Fuerza Unidad Inc. is a Glen Cove Long Island (NY) based non-profit organization which provides a variety of services primarily to Latino people in the Len Cove area. Services include:

- food
- housing
- relocation assistance
- foreclosure assistance
- credit assistance
- job placement
- job training
- English as a Second Language training
- Skills Training
- Educational Placement
- Child Care
- Coordinating assistance from governmental agencies

Operations are currently conducted by utilizing numerous unrelated Excel spreadsheets compiled and managed by individuals responsible for the provision of a particular service. The data contained in the spreadsheets includes name and address lists of donors, potential donors, clients, and contacts. Additional spreadsheets contain detailed transaction data pertaining to services rendered as part of the many programs made

available to clients. The input for such spreadsheets is manually gathered on input forms and a variety of reports are produced as the sole type of output. There is no simple way to determine if a client is receiving multiple services or if a donor is supporting multiple programs which provide multiple services.

In the Fall 2008 semester, the database classes began to address several major tasks of the Analysis and Logical Design Phases of the database design life cycle. Upon the commencement of the semester, the instructor explained the service learning project in detail and asked for volunteers. Five students, which is the maximum for this project, volunteered and served as the liaison between the non-profit and the remaining students. Those student volunteers, the instructor and representatives of the non-profit met on several occasions at the location of non-profit. The initial objectives of this first semester (the project will run over two semesters) were to gather:

- the objectives of the non-profit regarding the capture, retention and usage of data
- all data utilized in current operations
- the source, usage, and distribution of such data
- report requirements
- documentation of external interfaces (outside agency reporting requirements, etc.)

A database design life cycle template was then employed as a project planning tool and contained the major tasks that needed to be undertaken. These were assigned to the student volunteers and other class members. User site related tasks, such as gathering current files documentation and interviewing, were conducted by the five volunteers. Tasks which did not require direct user interface, such as analysis and initial logical design, were carried out by all class members working in teams and guided by the volunteers who had direct contact with the user representatives.

During the Analysis and Logical Design phases of the project, the following tasks were addressed, some of which were completed in the Fall semester, others to be completed in the Spring 2009 semester:

- the development of the business requirements including screen content and flow, reports, functions, calculations and data storage requirements
- analysis of the sources of data including interviews, observation, existing forms & reports, manual files, existing electronic files and databases, training manuals and operations manuals
- determination of the required tables, attributes, keys, relationships, cardinalities and normalization
- design issues such as transaction concurrency, security, backup and recovery.
- population of the database and validation of the data model.
- development of application logic including processes, queries, reports, forms, processing logic, controls and stored procedures.

An initial proof of concept model will be developed utilizing Access and implemented on a local LAN. After evaluation of the model, training and adjustments, it is planned to develop the production model in MySQL and run in a Client-Server environment.

DISCUSSION

Employment of a real life project as part of a Computer Science course such as Database Management Systems has several distinct advantages, including: [2]

- Increased student involvement, engagement and participation. The students realize that the project is a "real" one and that it is reflective of what they will be expected to do when they graduate and become involved in application development projects.
- Improved understanding of course principles and concepts since many of them are addressed in a real life environment as opposed to a paper and pencil exercise.
- The opportunity to lend assistance and expertise to a non-profit organization which, most often, has few resources or funds to otherwise undertake such a project.

On the other hand, use of a service learning project can have some disadvantages, namely: [2]

- The instructor is committing the college or university to the development and possibly the maintenance, of a production level application for a non-profit. This application development effort can be a challenging and risky undertaking especially considering that the project team consists mainly of untrained and inexperienced students working over just one semester at a time. A significant degree of dependency exists on the part of the non-profit upon the college or university.

  Perhaps the most prudent course of action to take in order to manage the risk and dependency issues is to carefully control the scope of the project including the definition of deliverables (products) and the expected time frame. [10]
- There may not be a close correspondence between the course content, goals and concepts with all of the service learning project needs. For example, given the scope of the project, while the course content may contain material about let us say data conversions, importing and data utility programs, the project might not involve such tasks. On the other side of the coin, the service learning project may require that certain tasks be completed, utilizing specific technologies, which are beyond the scope of the course content.

  Again, controlling the project scope may be solution to these potential difficulties. Other solutions involve having the support of the university's Information Technology organization which can provide experienced technical human resources to the project if needed. Be sure that the IT division does, in fact, have such resources available and has made the commitment to provide them if needed.

CONCLUSION

Incorporation of a non-profit service learning project is potentially a most worthwhile substitute for an instructor designed project within a Computer Science

course and can easily be a superior alternative. It can render significant benefits for the students as well as the non-profit but the instructor must balance the needs of both. Controlling the scope of the project and preparing to use the college's IT resources are two methods that may be employed to ensure that the non-profit's goals are met.

The instructor will be taking on an increased set of responsibilities concerning this type of endeavor. It is important for her to first meet with the non-profit's representatives in order to set expectations, goals and deliverables as well as to discuss how such a project might differ from a conventional one. Additionally, very early in the course, the instructor must make clear to the students what their roles are in such an effort, what their tasks are, and provide an explanation of the commitments to the non-profit. The student volunteers, who compromise a small subset of the full class, should meet early on with the non-profit representatives, along with the instructor, at the organization's site in order to learn more about their business, processes, data, people and policies.

Early in the course the instructor will most likely witness a significant level of student participation, enthusiasm and engagement. The practical experience gained, especially by the student volunteers who engage personally with the non-profit users, is invaluable. Even the non-volunteer students will gain significantly as they witness the reinforcement of course concepts. Students have a need, often unfulfilled in standard course projects, for interacting with users including interviewing, gathering of business requirements, testing and implementing solutions. As members of teams (user interface, analysis, design, implementation, etc.) the students will also be following "real world" practices. It has been stressed to the class that attendance at user meetings and project plan review sessions is mandatory. To date, experiences with student engagement and response have been excellent.

There appear to be few reasons as to why a non-profit related service learning project cannot be employed in most entry level Computer Science courses. [5] Non-profits make for a good source of projects but it must be kept in mind that an instructor will become involved in managing an application development project while simultaneously teaching her own course. A commitment is being made to the non-profit which becomes dependent upon the institution for the success of their project.

Several suggestions come to mind for those instructors who may wish to initiate this type of effort in their future course offerings:

- Consider running the non-profit project over two semesters.
- If the fit of the project's deliverables initially does not appear to be a good match with the course goals and contents, consider altering the project's scope to better make the match.
- If possible, provide for additional resources such as your institution's IT staff in order to fulfill commitments.
- Remember that the instructor is taking on the additional role of Project Manager and that the non-profit is depending upon her. Does the instructor have project management experience or are resources available to assist with tough decisions and to review project progress?
- Finally, there is no substitute for the creation and adherence to a well thought out project plan containing the tasks (in detail), the person(s) assigned, starting and ending dates, a description of the deliverable

(product) and dependencies. This plan should be reviewed with the non-profit in order to gain their full involvement as well as with the students. The instructor should cover with the students how each task contained in the plan relates to appropriate course content.

It becomes evident that additional planning, effort and management come into play should a service learning project be adopted for use in a Computer Science course but the rewards for the students, the non-profit organization and even the instructor, are most worth while.

REFERENCES

[1] Abelson, Hal and Greenspan, Phillip, Teaching Software Engineering, *Tenth International World Wide Web Conference* (Hong Kong), May 1-5, 2001.

[2] Almstrum, V. Condly, S, Johnson, A. Klappholz, D., Modesitt, K., and Owen, C., A Framework for Success in Real Projects for Real Clients Courses, in Ellis, H., Demurjian, S., & Naveda, F., (eds) (2008). Software Engineering: Effective Teaching and Learning Approaches and Practices. Hershey, PA: *IGI Global*.

[3] EPICS Program, Purdue University, http://epics.ecn.purdue.edu/, 2008.

[4] Gibson, J. Paul and O'Kelly, Jackie, Software Engineering as a model of Understanding for Learning and Problem Solving, *ICER*, Seattle, Washington, October 1-2, 2005.

[5] Henry, Joel, *Software Project Management: A Real-World Guide to Success,* Addison-Wesley, 2004.

[6] *Joint Task Force for Computing Curricula, Computing Curricula 2005.* ACM, 2005.

[7] Murray, M. and Guimaraes, M., Expanding the Database Curriculum, *Journal of Computing in Small Colleges* 23, (3), 69-75, 2008.

[8] Olsen, Anne, L., A Service Learning Project for a Software Engineering Course, *The Journal of Computing Sciences in Colleges, 24*, (2), 130-136, 2008.

[9] Traynor, Carol and McKenna, Maria, Service Learning Models Connecting Computer Science to the Community, *Inroads - The SIGSE Bulletin*, 35(4), 43-46, 2003.

[10] Umpress, David, A., Hendrix, T. Dean, and Cross, James H., Software in the Classroom: The Capstone Project Experience, *IEEE Software,* September/October, 78-85, 2002.

MOTIVATING PROGRAMMERS VIA AN ONLINE COMMUNITY*

Poul Henriksen and Michael Kölling
Computing Laboratory
University of Kent
p.henriksen@kent.ac.uk

Davin McCall
School of Engineering and IT
Deakin University
davmac@deakin.edu.au

ABSTRACT

Motivation is one of the decisive factors in the process of learning to program. This paper describes the Greenfoot Gallery, a project created to increase motivation of beginning programmers. The Greenfoot Gallery is a community driven web site that allows the publication of programming projects created with the Greenfoot environment. In this paper, we especially describe and discuss community communication patterns via commenting on the site, and possible effects of these on student motivation.

1. INTRODUCTION

One of the major problems for college and university computing departments over the last eight years or so are the rapidly declining enrolment numbers in our courses. This has been a consistent trend internationally.

Various studies have attempted to analyse the cause of this development [2, 7], and the most consistent factors seem to point to a serious image problem: Computer science is perceived as boring, geeky and not intellectually challenging. People working in the computing industry are viewed as lonely, isolated individuals sitting alone in dark rooms

---

in front of keyboards. Social interaction is not one of the associations that comes up in context of computing as a profession.

This means that the problem of declining interest in our discipline cannot be solved at university level. Students decide against engaging in computer science before they even come to us; an intervention is needed at a much younger age.

Thus, our problem of teaching programming has shifted and grown: Not only are technologies evolving under our feet, but we must now teach computing to younger students, who are originally less motivated, and make it interesting at the same.

2. PROGRAMMING AT SCHOOL AGE

While computer science is much more than just programming, we firmly believe that programming is one very good entry path into our field. Programming has several immensely useful characteristics in terms of generating interest in beginners: It can be constructive, immediate, "hands-on", fun, engaging and "real", while being intellectually stimulating and instructive at the same time. Few other topics of computer science have these advantages.

This argument, however, must be approached with a certain amount of care: Programming does not always or automatically have these characteristics. It can also be introduced in ways that are dull, abstract, theoretical, and where students do not see any practical relevance or personal enjoyment.

In this paper, we shall discuss some challenges and solutions to the problem of teaching programming to a young (high-school age) audience, and especially concentrate on aspects of online communities around programming.

Learning programming can be supported in various ways, which we can group into three broad classifications:

- We can provide support for the learning of programming, via educational means;
- We can simplify the mechanics of programming, via use of dedicated programming languages and other tools; and
- We can increase motivation.

All pedagogic interventions fall into one of these groups. Here, we shall discuss mostly the third category: increasing motivation.

One of our reasons to focus on this topic is that the shift in introductory programming teaching from university to high school has caused the greatest disruption in this category. Teachers at university level have a much easier starting point than school teachers – because they have a selective audience. University students have all (in some sense) made a conscious decision to be there. Some interest in the topic of study can be assumed (even though it is not always at a level or of a nature that we would consider ideal). By and large, the students are initially interested, and that makes it much easier for the teacher.

At high school, the situation is different. At the start of computer science or programming courses, many students have already decided that they are not interested in the subject. Prior experiences and hearsay often lead to negative attitudes in many

pupils, even before the course starts. Thus, generating motivation is the first thing that is needed in such a course, and this is much more important than at university level (although increased motivation is not a bad thing there, either).

Various software tools exist that aim at introducing programming at school age in a highly motivating context. Greenfoot [4], which is at the centre of this paper, is on of them; Alice [1] and Scratch [6] are the other best known tools at this level.

All these systems (and many more) have been documented in the literature. In this paper, we aim to discuss one specific aspect of the Greenfoot project: the online programmer community, supported though the Greenfoot Gallery web site [3].

3. ONLINE PROGRAMMING COMMUNITIES

With the dominance of the Web as the preferred internet interface, most communities are now web based (often with RSS and email integration), but still rely on essentially text-based discussion forums. Countless of these forums now exist, for every general or specific topic of programming, and for any experience level.

Specifically, we are interested in the motivational aspect of community. While the main aim of many general programming communities is the provision of technical help, this is only one of two main aims of our Greenfoot Gallery community, the other being motivation. In the case of the Gallery, motivation is achieved by facilitating recognition and encouragement through community interaction.

Community interaction has been proven to be a strong motivator for many teenagers today. Web communities such as Facebook, MySpace and Orkut are among the most popular places on the Web, and many young people spend considerable time and effort on interacting with communities through those sites. The communities here are often friends of the users, known to them from the physical world, so it is not clear that the enthusiasm easily transfers to other communities formed around different focal points. However, other sites form communities of strangers, who meet to discuss specific topics. Examples are Slashdot, Reddit, Digg, countless blogs and fan sites and Google groups, etc. Prior acquaintance does not seem to be a requirement for successful community interaction.

The two most directly relevant systems for comparison to Greenfoot – Alice and Scratch – both provide online support for their user communities as well.

Scratch has a community site similar to the one described in this paper, where users can publish their projects, and others can leave comments [8]. The Alice community forum does not have the ability to upload Alice projects directly. It is, instead, a traditional text-based discussion forum.

For Greenfoot, both exist: There is a traditional online discussion forum in addition to the Greenfoot Gallery, which allows publication of projects and discussion.

4. GREENFOOT

Greenfoot is a learning environment to support the learning of programming (especially object-oriented programming). The language exposed to users is Java, and the system is highly specialised for the development of animated, two-dimensional graphical applications, such as games and simulations. Greenfoot is typically used with students from age 15 upwards.

Using built-in functionality, any project created with Greenfoot can easily and quickly be exported to the Gallery.

5. GREENFOOT GALLERY DESIGN

The Gallery was designed specifically to increase motivation and engagement in programming activities through interactions with a subject-specific community. The goals were to increase motivation for Greenfoot users, and potentially to initiate interest in programming in young people who are not yet programmers.

The fact that motivation and engagement are primary goals in our project leads to some important design decisions for the com¬munity site. The most important is the sharing of artefacts.

The Greenfoot Gallery is a public web site that hosts Greenfoot projects. Any Greenfoot user can create an account on the Greenfoot Gallery and then publish their Greenfoot projects.

While within the Greenfoot environment, projects are editable and playable. Once exported to the Gallery, projects are no longer editable, but they are executable for anyone looking at the site.

Thus, the Greenfoot Gallery functions as a kind of "YouTube for Greenfoot projects": It provides a showcase for publishing work and offers options for others to view, rate and comment.

5.1 Live Program Publication

The central aspect of the Gallery design is the facility to publish working versions of Greenfoot projects directly to the web site. Projects - such as games - can be played directly in the web browser with full functionality. No additional software installations (other than the prevalent Java browser plug-in) are necessary for users of the web site.

This functionality serves two distinct purposes:

- Making their work visible to others on the web is a highly motivating factor for students developing those programs. It can be seen by friends and family, but also by many other people on the web, and many students get a thrill out of receiving feedback from strangers on their work. Examples of student reactions are given below.
- Some young people may interact with the Gallery site initially for the purpose of playing the available games (as pure consumers). Through this, they may start

thinking about starting to program these themselves, maybe because they wish to improve certain aspects of a game. The comments discussing implementation and a low hurdle to viewing and manipulating the source code encourage this. The Gallery site is too new to draw conclusions about whether or not this strategy is working, and no systematic investigation has been undertaken. This is an interesting question for future work.

Publication of a project may or may not include full source code of the project, at the choice of the publisher. Publishing the source code supports the second goal described above, and supports learning by example. The choice to withhold source code is given to support use of the Gallery for marked class projects where teachers do not want students to copy each other's solutions.

Students have to create an account on the Gallery to be able to publish projects. Creating an account is a quick and easy process and requires only a valid email address. The email address is not publicly visible on the web site, so users can choose to remain anonymous to the public (but not to the site administrators).

Projects can be played by anyone without creating an account.

5.2 Commenting

Once a project is published on the Gallery site, a separate page is created, where the live project, its author and time stamp and a description is displayed. On this page, other users can leave comments relating to this project.

This commenting facility is, jointly with the publication of the projects, the most important community functionality. The comments serve various purposes: feedback is given on projects, technical questions are asked and answered, suggestions are made, and more. A more detailed analysis of comments is given below.

Thus, the comments support several distinct goals: Firstly, seeing comments on their projects (especially positive comments) greatly increases excitement and motivation for students. Secondly, the comment section can carry the traditional function of online forums to provide help with technical problems.

To increase the feedback for project authors, the number of page views for a project page is also displayed. Through this, authors can also see how many people have looked at their projects even if they did not leave a comment.

5.3 Ratings

Users can also rate projects on the Gallery. The purpose is again to maximise the amount of feedback and motivation that authors get out of the site.

Different rating systems were considered. We decided against a numeric rating and opted for an "I like it" button that users can select for each given project. This way, projects can be singled out positively by community members, and negative judgement simply manifests itself in not selecting this button. Thus, the harshest experience possible for authors is to be ignored, which we consider an acceptable form of feedback.

Each project page presents a count of the number of people who "like it" as an approximate popularity rating.

5.4 Front Page Presentation

The front page of the Greenfoot Gallery consists of several important sections. Three of these sections present selected projects: The top of the page displays "showcase" projects (manually selected projects that site administrators have chosen to emphasize), below this are dynamically created lists of the most recent and most popular submissions.

All of these are intended to give each project author the chance to receive special attention by being listed on the site's front page. Especially the "most recent" list guarantees that every project is visible on the front page for at least a while.

The front page also includes a summary of the most recent comments on the site, a news section and a tag cloud.

5.5 Further Functionality

Many other functionality elements were discussed and several additions are planned to increase the potential for interaction and engagement on the Gallery site, and to increase the potential of generating motivation. Some of these are discussed in Section 8.

6. OBSERVATIONS OF USE

The Greenfoot Gallery site was published in July 2008. Public visibility was initially low. Starting from September 2008, site visibility was increased by linking to it from other Greenfoot web sites and announcing it on Greenfoot related discussion groups.

Between then and mid January 2009, 544 users have signed up to the site, submitting 302 projects and leaving 1311 comments. The web site received approx. 34,000 visits (180,000 page views) by 19,400 absolute unique visitors (data collected with Google Analytics). The average page views per visit are 5.25 and the average time on site is 5 min, 23 sec. This demonstrates a significant interaction with the Gallery also by users who have not registered as users on the site.

The bounce rate (people who leave the site directly from the page where they entered it) is 33.85%. This supports the observation that most visitors enter into some interaction with the site.

6.1 User Engagement

To investigate the frequency of different use patterns, we distin¬guish four user categories: Visitors, Consumers, Interactors and Creators. Descriptions and numbers of each are given in Table 1.

User category	Actions	Count
Creator	Submit projects	171
Interactor	Leave comments and ratings	18
Consumer	Create user account; no other creation; read and play	335
Visitor	Read and play; no account creation	18900

Table 1: Different user types (higher categories include actions of lower ones)

As expected, the number of visitors is far higher than those of the other categories. Among the users who have created accounts, it is encouraging to see a fairly high percentage of Creators (32.6%).

Creators often submit more than one scenario. The average number of scenarios per Creator is 1.8.

An interesting research question for the future is to investigate migration of individuals from one category into a higher one, and whether such migration can be actively supported.

6.2 Comments

In this paper, we are especially interested in the comments left for published projects, as this provides deeper insight into the different forms of interaction.

A project has a high chance of receiving comments. 81% of all pro¬jects have at least one comment by someone that is not the owner of the project. The average number of comments per project is 5.2.

We have analysed all comments left for any scenario that was submitted or updated in a period of 6 weeks between 10 December 2008 and 21 January 2009 (70 scenarios in total).

The first surprising result was that we have found no negative comments. As administrators, we expected to have to occasion¬ally remove inappropriate comments from the site, but this has not been the case. Not only were there no offensive postings in the six months of the site's existence, but the analysis also showed no nega¬tive (purely critical or discouraging) comments left for any project.

When projects were uninteresting or poorly written, the worst that happened was to receive no comments. But even incomplete or poorly written projects often received feedback encouraging improvement of the scenario. The following comment for a project that simply allowed movement of a character is typical:

> "I like the way the animations work. Is this going to be some kind of platform game? It'd look much better if you had to get to the star by jumping on bricks and avoiding enemies rather than just holding the arrow keys down until you get there :)"

We are not certain about the cause for this positive tone on the site, but offer some speculation below.

We evaluated the main purpose and content of each comment and created categories reflecting their focus. The categories chosen are: Encouragement, Technical discussion, Exchanging ideas, Sharing resources, and Miscellaneous.

6.3 Encouragement

44% of all comments were classified as encouragement. Some comments contained elements of more than one category. In this case, their were counted in more than one category. The simplest form of encouragement consists purely of positive remarks about the project, e.g.:

> "Nice! It's really fun, and the graphics and movements are very nice too."

Often, the feedback was specific about certain concrete aspects of the project:

> "Great job. I really like the whole game in general. No major problems, perhaps you could have added music. Great images and a nice GAME OVER at the end (which I happen to see alot)!!!"

This feedback was not only useful for the immediate author, but we observed cases where these remarks were picked up by other project authors to get ideas for improving their own projects.

Another form of encouragement includes suggestions for further improvement. This often went together with positive feedback:

> "Great idea for a game. The graphics are very nice now. I especially like the bubbles.
>
> "Seems a bit hard to navigate from side to side. Maybe you could gradually slow the ship down when no key is pressed?"

Suggestions like these seem to have been successful in creating motivation for the authors. We observed numerous cases where these suggestions were subsequently implemented in the project.

Another interesting pattern we observed repeatedly was the development of a dialogue, where the project author replied to the user comments:

> "Thanks all for the suggestions. All the bugs have been ironed out and it runs quite smoothly. If anyone can think of any new creative ways of killing off the people or any other features let me know and I will try to implement them."

This demonstrates directly the interaction between project development and community response.

6.4 Technical Discussion

Another obvious category were comments asking for or offering technical help (37% of comments). This includes technical questions, bug reports, and technical advice.

Most interesting for us was that help was not only given as a result of direct questions (although that also happened), but frequently unprompted. It seems to have

been a common pattern that users studied the source code of a published project, discovered bugs or possible ways to improve it, and offered their advice to the project author. The following comment illustrates this:

> "hey, i discovered this neat trick, java has a way of drawing a scaled image onto another one, so you can actually replace your enemy drawing code with this single line:
>
> im.getAwtImage().getGraphics().drawImage(en.getAwtImage(), (int)(x1-l), (int)(150-l*0.9), (int)(2*l), (int)(2*l),null);"

6.5 Exchanging Ideas

In the context of dealing with diverse skill levels of students in a class, it has been suggested that open assignments are a way to cope with this problem [5]: Good students can then develop additional project extension ideas to challenge them at their own level of ability.

We observe that many comments on the Greenfoot Gallery help to support this process by providing ideas for additional project extensions. 14% of the comments fall in this category. The following is a typical example:

> "Pistols shouldn't drop when you die: they clutter up the map a lot.
>
> Add a weapon "Sniper Rifle" with low fire rate but a laser line. (This can't be too hard, can it?)
>
> Add the option to have movement controls be relative to the direction faced instead of relative to the screen."

Again, this process works both ways. Comments were also used by project authors to check ideas with users before implementing them:

> "Btw, how tall are your screens? Because right now the game has a resolution of 1024x800. I was thinking about lowering it to 600? Or would that not be enough?
>
> Thanks again for your help."

6.6 Sharing Resources

Some comments revolved around the sharing of resources for project development (3% of all comments). Sometimes the resource was a class or other unit of code from the project itself (typically initiated by the potential user of the resource). For example:

> "Hey, do you mind if I use your code for the midi player for my game?"

And the reply:

> "You can use it ;). Anyone can use my code as long as credit is given where it's needed."

In other cases, the resource in question was external to the source code. In that case, it was sometimes offered by the author:

"i wrote some tutorial in greenfoot in my blog (sorry all, Indonesian language ^_^")

this, maybe useful, http://mandelag.blogspot.com/"

We also observed many cases of source code sharing where it was not explicitly discussed through the Gallery comments. Some classes were regularly reused across many different projects.

6.7 Miscellaneous Comments

The remaining comments were either categorised as Social talk (unrelated to the project development, 2% of comments) or did not fit any of our categories (3% of comments).

7. PEDAGOGICAL CONSIDERATIONS

At this stage it is still impossible to draw firm conclusions from the data available. Especially, it is difficult to make definite statements about the effects of the gallery interactions on motivation and learning. The user group was self-selected, and it is not clear to what extent communication and interaction patterns can be transferred to, for example, a classroom setting.

However, some of the observations are interesting enough to formulate some hypotheses for future investigation.

- It seems that the ability to publish student work may increase motivation for students. This is indicated by a level of engagement with projects that was deeper than what is typically observed in student projects.
- The encouragement received on the web site may help to sustain engagement with the project for a longer time.
- The technical advice provided through the site may help teachers by offering additional sources of help.
- The exchange of ideas may help to challenge especially the good students.
- Sharing existing resources may produce programs that are better in visual appearance or interaction, thus allowing students to learn programming in a context that is closer to their existing expectations of modern software.

Overall, the interaction and communication patterns that have evolved between users point to the Greenfoot Gallery as a promising instrument for improving motivation and engagement in classroom computing courses, as well as for informal learning.

We plan to execute a more formal test of these hypotheses at a later time.

8. DISCUSSION AND FUTURE WORK

One striking observation of communication on the Gallery is the exclusively positive and polite tone of the comments. We speculate that this may have been caused

by very slow initial growth. In the first two or three months, the number of subscribers was low, and the Greenfoot development team members presented a fairly high percentage of users on the site, thus able to set a tone on the site and implant culture and expectations. It will be interesting to see whether this carries on as user numbers grow.

Amongst plans for technical development of the Gallery, we consider the addition of personal scenario collections a high priority. This would allow a member of the gallery to create their own named collection of scenarios, into which they can invite scenarios developed by other members. The creator of the invited scenario can then choose to accept or decline the invitation. In a similar way, a scenario creator can ask to have their scenario added to an existing collection, and the collection owner can then accept or decline the addition.

Other than providing an alternative to the "I like it" mechanism for showing which collections a member personally appreciates, we envisage that themed collections could be created (such as a collection for simulations, for instance, or one for "platform" games). Collections might also be used to group scenarios for other reasons, such as to showcase scenarios developed by a particular school or group.

The addition of personal collections adds a new creative facet to the Gallery. It potentially allows non-programmers to become more involved by being able to create something that they "owned", rather than just commenting on other members' scenarios. A popularity system for collections, similar to that already provided for scenarios, would provide a motivation to carefully choose suitable scenarios and to maintain the collection over time.

Collections also have the potential to increase the interactivity between Gallery members. Creating a collection and populating it with scenarios naturally requires establishing a communication between the collection owner and the scenario creators.

Collections may also present an additional motivating factor for scenario developers – that is, to get their scenarios into various collections. Collection owners are free to make whatever demands they wish on the quality and nature of scenarios that are to be part of their collection; this may provide a challenge to scenario developers to meet such requirements, and give them a sense of satisfaction and accomplishment once their scenario has been accepted.

9. REFERENCES

[1] Cooper, S., Dann, W., & Pausch, R., Alice: A 3-D tool for introductory programming concepts. In Proceedings of the 5th Annual CCSC Northeastern Conference 2000, Ramapo, NJ, April, 28-29, 2000.

[2] Council of Professors and Heads of Computing (CPHC), Investigation into the decline in BSc Computing/IT Applications to British Universities; http://www.cphc.ac.uk/docs/cphc-admissions-trends-report.pdf , July 2006, accessed Jan 2009.

[3] Greenfoot Gallery. Web site at http://www.greenfootgallery.org, accessed Jan 2009.

[4] Henriksen, P. & Kölling, M., Greenfoot: Combining object visualisation with interaction. In Companion to OOPSLA'04, pages 73-82, 2004.

[5] Kölling, M. and Barnes, D., Apprentice-based learning. In Bennedsen, Caspersen, Kölling (Eds.), Reflections on the Teaching of Programming, Lecture Notes in Computer Science, Vol. 4821, pages 29-43. Springer, April 2008.

[6] Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., and Resnick, M., Scratch: A Sneak Preview. Second International Conference on Creating, Connecting, and Collaborating through Computing. Kyoto, Japan, pp. 104-109, 2004.

[7] Microsoft Corporation, Developing the Future– A report on the challenges and opportunities facing the UK software development industry, http://download.microsoft.com/documents/UK/developingthefuture/Developing%20_The_Future_07.pdf, accessed Jan 2009.

[8] Monroy-Hernández, A. and Resnick, M. Empowering kids to create and share programmable media. Interactions, March-April 2008.

CYBER-POLITICS: DEVELOPING AN INTERDISCIPLINARY LEARNING COMMUNITY IN AN ELECTION YEAR*

Michael Ruth, Adrian Ionescu
Math & Computer Science Department
Wagner College
Staten Island, NY 10301
{michael.ruth, ionescu}@wagner.edu

ABSTRACT

This paper is focused on exploring the experience gained through developing a learning community integrating the political science and computer science disciplines. The two courses from the disciplines, which are general education courses, are an Introductory to Government and Politics course and an Introduction to Networking and the Internet course. The final component is a freshman writing composition course focused on the intersection of the two disciplines. The major contribution of this work is in describing the weaving of political science concepts into the Introduction to Networking and the Internet course and the development of the writing composition course. In addition, the paper will discuss the experiential learning component of the learning community which was designed to ensure that students are made aware of the connections between what they are learning and the needs of the community at large, especially during the 2008 presidential election.

INTRODUCTION

Interdisciplinary learning through the use of learning communities is becoming more common in higher education today and for good reason. In [1], the author reports that students benefit from learning communities in that they tend to form self-supporting groups which extend beyond the classroom, students become more actively involved in classroom learning even after class, and student participation in the learning community seemed to enhance the quality of student learning. Additionally, learning communities also play a role in enhancing student retention initiatives [2]. These student benefits

provide the motivation for colleges to move towards interdisciplinary learning through learning communities.

At Wagner College, the learning community model, including interdisciplinary learning, has been adapted into the Wagner Plan [3]. This plan requires all entering freshman to take part in a program which involves interdisciplinary and experiential learning through the use of learning communities. The learning community consists of three separate courses, two from different fields, and a freshman writing composition course focused on integrating material from the two courses through discussion, writing, and reflection focused on the topics being presented in the two courses. In addition, the developer of each field course is encouraged to further integrate topics from the other field course into their courses.

This paper is focused on exploring the experience gained through developing the learning community which integrated the politics and computer science disciplines. The two courses from the disciplines, which are generally general education courses, are an Introduction to Government and Politics course from political science and an Introduction to Networking and the Internet course from computer science. The major contribution of this work is in describing the weaving of political science concepts into the Introduction to Networking and the Internet course and the development of the writing composition course. In addition, the paper will discuss the experiential learning component of the course which was designed to ensure that students are made aware of the connections between what they are learning and the needs of the community at large, especially during the 2008 presidential election.

The rest of the paper is organized as follows. Section two describes the integration of politics into the Networking and the Internet course. In section three, the development of the writing composition course, which is also called the freshman reflective tutorial, is discussed in detail. Section four discusses the experiential learning component of the course and section five concludes.

NETWORKING AND THE INTERNET

As described earlier, the Networking and the Internet Course is an introductory course in networking concepts, topics relevant to the Internet, and the development of their own Web content. In the beginning of the course, students are introduced to networking concepts and basic Internet application and protocol concepts. However, the rest and the majority of the course is focused on students becoming proficient in Web development using HTML and CSS. In addition to discussions, reading assignments, and lectures students were required to complete a number of in-class assignments and a course project to demonstrate proficiency in Web development. These in-class assignments and course projects were ideal places to integrate political science concepts into the course.

After several individual topic discussions, which involved required reading assignments and a lecture, the students were asked to perform an in-class assignment related to the discussion. Each of these involved some political concept in some way. For instance, after discussing text decoration, the students were asked to develop a candidate profile of themselves as a candidate for President of the United States. This particular assignment allowed the students to determine for themselves what

qualifications were necessary for them to eventually become President of the United States in addition to using the text decoration skills developed in the course. Another example of an in-class assignment geared toward integrating political concepts involves the use of HTML lists. Each student was asked to develop a list of speaking engagements in their home towns and produce the list in HTML list form. These speaking engagements, or events, should connect a group, a location, and a topic that group is interested in. The assignment was designed for the students to consider the importance of groups in the campaign process and the importance and dangers of pandering to these groups while practicing the skills developed in the course. All of the in-class assignments were designed similarly. Each assignment had some political component and was designed to demonstrate the skills being developed in the course while promoting critical thinking about political concepts. The course project was designed very similarly to the in-class assignments.

The course project was designed for the students to demonstrate their proficiency in Web development concepts while focusing on political science concepts. Each student was asked to develop a presidential Website using several elements discussed in the course, including HTML tables, forms, text decoration elements, lists, and custom graphics. The students worked in pairs (president/vice president) in order to design and layout the page, but then individually on the implementation. In addition to having the elements presented throughout, including the list of speaking engagements and the resumes of the involved individuals, the students were asked to create a platform consisting of five major issues that were the most important to them. Some student selected issues from the actual presidential candidates running in the 2008 presidential election, while others developed their own issues.

Since 2008 was an election year, there were a number of campaigns operating Web sites in operation, presenting a diverse set of issues and platforms using a wide variety of Web development design and layout options. The design/layout of these Web sites not only provided the course with a very useful discussion about layout/design but also provided a wide variety of examples and ideas on design, layout, and platforms for the students. The students were given wide latitude to discuss and consider the issues that they felt were most important to them and only use the best parts of those discussions on their Web sites. Finally, some of the students also personalized their Web sites using some of the pictures and slogans from the actual campaigns.

The projects and assignments in the Networking and the Internet course were all designed and developed to enhance critical thinking and encourage engagement in contemporary political issues while demonstrating mastery of Web development concepts. The majority of students demonstrated both critical thinking and engagement in the issues on their Web sites as well as mastery of the concepts presented in the course. In addition, several students reported that their favorite portion of the course was the course project on their student evaluations.

FRESHMAN REFLECTIVE TUTORIAL

The freshman reflective tutorial is a freshman writing composition course focused on the intersection of two courses from separate fields, in this case an introductory Government and Politics course and a course focused on the Internet. The course is designed to improve oral and written communication skills and develop critical thinking and problem solving skills through systematic reflection and discussion. The students are presented with a number of topics and asked to discuss, reflect, write, and present their views on these topics. The writing composition component is focused on demonstrating proficiency in several different writing styles including exploratory writing, argumentative writing, and the development of research papers with the content of those writing skills focused on the five topics. Fortunately, Wagner College provides considerable assistance developing the writing composition component of the course, including providing examples and discussions involving grading, instruction, and leading writing workshops and holding successful student debates. The development of the topics is then the most critical element and the topics developed for the course were focused on the impact the Internet has had on the American political landscape, including issues such as Internet and digital privacy, freedom of speech and the Internet, Democracy and the Internet, the Internet as a record, and the digital divide. These topics were chosen in part because they lend themselves to a variety of viewpoints which provide students with the opportunity to practice their critical thinking skills. This section will discuss each of these issues in detail.

The first topic involves privacy and the internet and the focus of this portion of the course was on personal, or individual, privacy. The topic was logically divided into three separate elements: defining what is meant by individual privacy, privacy and the workplace, and the impact that the combination of personal technology and the Internet has had on personal privacy. The discussions and readings were primarily focused on the second two elements. The readings and discussions involving privacy in the workplace described the balance between the rights of the employers to protect themselves with the privacy rights of employees and involved several cases in which individuals were impacted by their online presence. Some of the individuals discussed had online profiles, on sites like myspace.com and facebook.com, and were denied work or advancement due to information found on those profiles. Another group of individuals we discussed at a firm under federal indictment found all email they had sent using their employee account online. The discussions and readings used to discuss the impact that the collision of personal technology, such as cellular phones and cameras, and the Internet has had on personal privacy were primarily focused on a variety of cases generally involving an individual whose privacy was violated online through the use of personal technology. This topic was chosen for a variety of reasons but most specifically as technology and their use of it increases, privacy, or the lack of it, will have a direct and immediate effect in their personal and professional lives.

Freedom of speech as it relates to the Internet was the second topic discussed and had three separate components including libel as it pertains to the Internet, pornography on the Internet, and a comparison of speech laws around the world. The readings and discussion concerning libel considered several cases relating to posting negative information on the Internet about individuals and organizations which can be considered harmful. The discussions involving pornography on the Internet were focused on

protecting children from accessing adult content online. Finally, the discussions and readings regarding the comparison of speech laws around the world were focused on several cases involving the relative levels of speech allowable online by three main areas: the US, China, and the EU. In comparing the US and China (or other less free zones), the cases involved politically motivated censorship and in comparing the US and the EU, the cases involve primarily the concept of "hate speech". This topic was chosen to help students understand the importance of free speech to representative democracies including its connection to other rights.

After discussing the importance of freedom of speech to representative democracies, the third issue is the impact the Internet has had on democracy in the United States. The discussions and readings centered around whether the Internet has had a net positive and or net negative impact on democracy. The majority of the arguments in the readings and discussions were focused on the factors for participatory democracy and the impact the Internet has had on each of the factors. The two most discussed factors were an informed, engaged citizenry and an informed, unbiased media. In discussing the positive impact of the Internet on informed, engaged citizenry, profile sites such as facebook.com and myspace.com were discussed in terms of spreading information and improving political organization. The negative impact is in terms of the uninformed being engaged and performing the same tasks. The positive aspects of the media revolve around the blogger and the ability of the average person to become part of a "grassroots" media, but the negative aspects involved cases where a uniformed or biased media became part of the same media. The other factors discussed are fair and balanced elections, accountability, and transparency. The topic was chosen because it allowed the students the opportunity to discuss and debate the various factors of a representative democracy and discuss and debate the relative importance of each.

The fourth issue concerned considering the Internet as a record. As technology advances, the permanent, or semi-permanent, storage of information is becoming more common. The discussions and readings were focused on the impact that information "left" on the Internet, including speech transcripts and online videos of speeches, has had on the American political landscape. Another avenue of discussion was the increased exposure of politicians to this information and the lasting impact that exposure can have. For instance, even when presidential candidates spoke in very small forums and town hall meetings, those speeches were broadcasted over the Internet where they remain viewable on sites such as YouTube long after the speech is over. In fact, during the 2008 presidential campaign, pundits and campaigns were comparing speeches and producing collages of speeches to show position changes (flip-flopping) in real time. This topic was chosen because it is a relatively new phenomenon which allows students to see the impact of the Internet on politics on a grander scale especially in the realm of holding politicians accountable. Additionally, it helps the students understand the lasting impact of information online and that every piece of information on the Internet may be there for a very long time.

The fifth and final topic concerned the controversy surrounding the digital divide. The readings and discussions involved several aspects of the digital divide including whether or not it exists, how the US government has changed its views of the digital divide, and the challenges involved in overcoming the digital divide. Several relevant factors involved in where the division is drawn were discussed including socioeconomic

status, income, education, and race. In addition, the US Department of Commerce produced two separate reports under different administrations with very different conclusions [4]. The students read and discussed the factors, methods, and results of both studies. In addition, students read and discussed articles related to approaches to overcome the digital divide and the merits and challenges each presented. This topic was chosen for a variety of reasons including the questioning of methods, factors, and results and the importance of understanding how socioeconomic status, income, education, and race can impact the everyday lives of Americans.

EXPERIENTIAL LEARNING

As mentioned earlier, the course involves experiential learning and occurred during the 2008 Presidential election season. Every student is required, as part of the Wagner plan, to complete at least 30 community service hours during the course of the semester. In the learning community developed, there were 23 students needing community service hours and a place to volunteer. Since the election season was underway, several voter registration drives were in operation and starving for volunteers. The instructors obtained a large number of voter registration forms from one of these drives and registered voters alongside the students to encourage the students to participate more freely. In addition, several students, on their own, volunteered and worked on the Obama and McCain presidential campaigns. At the end of the semester, no students were in need of community service hours and the students registered over 200 voters in the NYC area!

CONCLUSION

Overall, several factors indicate that the course was successful including the student evaluations. The ongoing 2008 presidential campaign enhanced the course offering by making some issues easier to present and by easing the placement of student volunteers. One factor under special consideration was whether or not the topic selections were successful. Near the end of the semester the students were provided with an anonymous survey to help determine which topics went well and which needed improvement. Overall, the students seemed very pleased with the privacy component of the course, the students overwhelmingly ranked it the highest in terms of the most interesting and students did seem engaged in the topic in the writing components which covered privacy. The freedom of speech as it relates to the Internet was the second most interesting topic although it appears that some students seemed more interested in the pornography component while others seemed more interested in the international comparison component of the free speech topic. The students were given a choice during the course which they preferred to write about and their writings on the topic clearly demonstrated engagement in the topic. As discussed, the students seemed to clearly understand and appreciate these topics and the writing required during those topic discussions demonstrates interest and engagement in the topic.

However, the other three topics did not fare nearly as well and clearly need improvement. Of the three topics left, there was no clear winner or loser as some students preferred one over the other depending on their interest level. Students reported that the democracy and the Internet topic needed to become more focused with less coverage. Some of the students reported being confused about the fringe topics outside

of discussing the impact the Internet has had on informed, unbiased media and an informed, engaged electorate. In the future, this topic discussion would need to be reduced to include only the pieces where the Internet has had the largest impact, namely the electorate and media. The majority of students felt that the discussion and readings concerning the Internet as a record was either confusing or irrelevant to their daily lives. In order to improve the course, this topic needs to be taught with several of the other components rather than as separate topic as it seems only to enhance the discussions of other topics, like the Internet is making a lasting impact on the media and electorate, rather than being strong enough to be discussed on its own. Finally, several students reported that the discussion surrounding the digital divide was biased, but this is perhaps caused by the majority of students arguing for one side and in order to bring about balance, the instructor led the other side of the discussion. In order to improve this topic, more balanced readings must be found to provide both sides of the argument. For the readings, an instructor may choose to allow the students to select their own readings and argue their relevance. The instructor, in this case, chose to select the readings to achieve balance with respect to not only bias, but the issues as well. Even though the students reported that these were their least favorite topics, some students showed engagement in their writing, just not at the same high level as the other two topics where virtually everyone demonstrated engagement. Overall, after reviewing the comments of the students and the engagement demonstrated in their writing, the chosen topics were successful, even though there are some necessary adjustments that must be made, which is not altogether surprising, given that this is the first time the instructor taught the course.

REFERENCES

[1] Tinto, V., Learning Better Together: The Impact of Learning Communities on Student Success, *Journal of Institutional Research*, Vol. 9, 48-53, 2000.

[2] Ivanitskaya, L., Clark, D., Montgomery, G., Primeau, R., Interdisciplinary Learning: Process and Outcomes, *Innovative Higher Education*, Vol. 27, No. 2, 95-111, 2002.

[3] Wagner College, Wagner Plan, 2008, www.wagner.edu/ wagner _plan, Nov. 2008.

[4] US Department of Commerce National Telecommunications and Information Administration, http://www.ntia.doc.gov/reports/anol/index.html, Dec. 2008.

IMPLEMENTING A BACCALAUREATE PROGRAM IN COMPUTER FORENSICS*

Jigang Liu
Department of Information and Computer Sciences
Metropolitan State University
700 East Seventh Street
St. Paul, MN 55106
651-793-1472
Jigang.liu@metrostate.edu

ABSTRACT

Since the beginning of this century, motivated by the demand for professionals in computer forensics from the industry, law enforcement, and government in dealing with internal affairs and fighting against digital crimes and terrorism in cyberspace, various curriculum programs have been developed, including certificate, associate, baccalaureate, and graduate programs. In this paper, we will share our experience in implementing a baccalaureate program in computer forensics with respects to its challenges and implementation tactics and strategies. We hope our experience will help others not only in developing a program in a computer forensics related area but also in exploring a new program in one of the other emerging technology fields.

1. INTRODUCTION

As the Internet and information technology has reached every corner of our daily lives, criminal activities have come with it, too. Based on the data provided by the Office of Management and Budget reported by USA Today (6A) on March 14, 2008, incidents of security breaches reported to federal agencies and to the U.S. Computer Emergency Readiness Team has increased by 44 percent from FY2005 to FY2006, and 152 percent from FY2006 to FY2007. New challenges in the field are the new opportunities for academia to develop new programs that take these challenges and meet both the job markets' as well as the students' needs.

Based on the information provided by the E-Evidence website [11], 68 institutions around the world offer computer forensic related courses and curricula. Out of those 68 institutions, 52 are from the U.S. Although the information provided at the website is not complete, we can at least see the scope of the interest in the academia.

Bachelor programs have been progressing stably since a survey made in 2004[13]. As Table 1.1 shows, the number of regular computer forensics programs has reached five in the nation, excluding the programs with computer forensics as a minor, track, or a concentration.

In addition to those five "computer forensics" programs, other related four-year programs are also available, such as "Computer Investigations and Criminal Justice"[7] and "Information Assurance and Forensics"[16]. Some have included the computer forensics component to their existing program with a minor or a concentration [6]. To prevent the new offering from disturbing the existing programs and to keep the investment at its minimum, many institutions have chosen to create a separate certificate program in computer forensics [4,5,9,12]. Similar to the development in other academic disciplines, online programs in computer forensics are also available, such as the one offered at Champlain College [2].

Table 1.1 Baccalaureate programs in Computer Forensics in the U.S. (April 2009)

Institution	Program	Degree	Type	Location	Website
Bloomsburg University	Computer Forensics	BS	Public	Bloomsburg, PA	www.bloomu.edu/admin/acad/mat/forensics.plp
Champlain College	Computer & Digital Forensics	BS	Private	Burlington, VT	digitalforensics.champlain.edu/
Defiance College	Computer Forensics	BS	Private	Defiance, OH	www.defiance.edu/pages/BASS_majors_CF.html
Metropolitan State University	Computer Forensics	BAS	Public	Twin Cities, MN	archive.metrostate.edu/cas/csci/forensic.html
University of Advancing Technology	Computer Forensics	BS	Private	Tempe, AZ	www.uat.edu/majors/computer_forensics.aspx

Another recent development trend is to offer a Master's level program in the field [20,21]. The Master program in Forensic Computing developed at John Jay College of Criminal Justice can be considered the one with the longest history [22]. To find a way to meet their students' need, University of Rhode Island has crafted a digital forensics track for both their MS and Ph.D. programs in computer science [10]. The latest development of graduate programs in computer forensics is a Ph. D. program in Digital Forensics launched at California Sciences Institute in January, 2009[23].

The rest of the paper is organized as follows. In next section, the status of our program is presented. Implementation tactics and strategies are discussed in Section 3. In section 4, conclusion and future work is provided.

2. CURRENT STATUS OF THE PROGRAM

The new program was approved by the university system of Minnesota Spring 2005 and made available to students in Fall 2005. Since then, the program has grown steadily and operations have been running smoothly.

As discussed in [18], we intentionally did not propose to have a substantial number of new courses for the new program initially, but rather, to include more existing courses. This decision helped us in having a very smooth start and avoided many distractions and crises in finding qualified instructors and preparing teaching materials.

For each new course, it has to go through a 5-step approval procedure, including approvals from the department, the college's curriculum committee, the Dean of the college, the academic steering committee of the university, and the VP for Academic Affairs. By the end of Spring 2007, all six new courses listed in the proposal had been approved.

Since computer forensics is an emerging field, we have experienced extreme difficulty in recruiting resident as well as adjunct faculty members. Fortunately, we managed to offer computer forensics courses through the hiring of adjunct professors for the first two years and had a full-time tenure track professor join us in Fall 2007. We could not have made this happen without a strong local professionals' support.

To avoid having a one-man show for the entire program, we have tried to recruit people from varying backgrounds to teach in their areas of expertise. Among the faculty members (14 adjunct as well as resident faculty members) we have recruited so far, 3 have a Ph.D., 2 have a J.D., and 4 have a CISSP (Certified Information System Security Professional, a de fact certification for either computer security or computer forensics)[3].

The student enrollment to the program has grown steadily over the first three years. The total number of the students had reached 152 by the end of summer 2008, excluding the 9 students who had already graduated by then. Among 152 students, 30% are women, which is higher than the percentage of female students in either the computer information systems or the computer science programs in our department. Out of 9 students who have graduated, three were women, which constitute 33% of the graduates.

From a financial point of view, we did not receive any new money to run the program at the beginning. However, we chose not to wait for this money, but rather, to reallocate the departmental budget in favoring the new program. With the university's support in the lab facility and IT infrastructure, we managed to run the program in the first year without any additional funding. For the next three years, we were fortunate to be granted roughly $200,000 from the state for purchasing equipment, software licenses, and training.

3. IMPLEMENTATION STRATEGIES AND TACTICS

Our experience in implementing this new program has exactly echoed the statement made in [14], which says that "historically, the formation of a new discipline has been a slow process that happens in a very ad hoc manner. Only after years of growing pains does the discipline become defined to the level that real progress in research can be made." In this section, we share the challenges we have encountered and the ways in

which we dealt with them in hopes of helping others ease the process when they jump into this new field or any other emerging areas.

3.1 New Course Development

To minimize glitches, all the new courses are developed through the collaboration between full time and part-time faculty members. With knowledge of the new curriculum and the university's requirements, the full time faculty plays an important role in balancing the course materials, determining prerequisites, and coordinating the new course approval process. On the other hand, adjunct faculty members provide their first-hand experience and knowledge in arranging the course content and lab activities.

Selecting a book for a single computer forensics course should not be too difficult due to the vast availability of books on this subject. However, to select textbooks for an entire curriculum is quite challenge due to the considerations such as overlapping content, sequence of knowledge, and balance of teaching materials in lectures, research projects, and lab activities. To help give us an overview of the availability of books on the subject, we did a survey on computer forensics textbooks [19]. The report of the survey helped us match the right books for the right courses.

Developing a new course is an ever evolving process. With this fundamental concept in mind, we always looked carefully at students' teaching evaluations each semester to see where we could make any improvements and adjustments for the next semester. Moreover, to balance the view on a particular course, we also intentionally had different faculty members teach it so that we could have multiple opinions on the course and strategize the best way to teach the course.

3.2 Faculty Recruitment

To have a successful program, the faculty plays one of the most critical roles. Due to the limited pool of qualified candidates, we didn't have a full time faculty in computer forensics until Fall 2007. In other words, for the first two years, we mainly relied on our adjunct faculty members to teach the three core courses. In order to find local talents, besides utilizing the traditional channels like the university's website and local news papers, we had taken three tactics in our local recruitment, namely "social networking," "showing your face," and "targeted search."

We started to look for potential adjunct faculty long before the program had launched because we had to provide sufficient time for them to make a proper preparation for the course they were going to teach. Asking fellow colleagues and acquaintances for their recommendations was the first thing we did in our recruitment. As soon as we heard a name, we followed up with that person with either a phone call or an email. If they were interested, an on-campus interview was scheduled.

"Showing your face" means to present ourselves in front of the professionals, public media, as well as social events to inform others about our program. At the same time, we have the opportunity to get to know them. This strategy has been proven to be the most effective way in recruiting our adjunct faculty members as 50 percent of our hires were initiated in this way.

In addition to the fundamental knowledge and the practical skills, we are also looking for the professionals who are able to teach upper level courses that include a research component. Since there are only a handful of people who fall under this category, we usually have to do a "targeted search." We either introduced ourselves to the presenters who were from Minnesota at national and international conferences or locate local authors who had just published research papers. For those we could not meet at conferences or find online, we would check their websites to see where we could meet them in person at a local professional or social event. Unless we had met them in person, the recruitment never started via a phone call or an email.

3.3 Resources for Funding

There is nothing more difficult than asking someone to invest in your program. In this section, we share our experiences and strategies in fundraising for the new program.

Due to the overwhelming preparation and groundwork as well as our shortage of manpower, we did not attempt to request any federal funding when we started offering the program. Instead, we focused on local grant opportunities, which usually have a shorter turnaround time and a higher approval rate. For instance, we used a state grant in organizing a symposium on computer forensics in Oct. 2004. It took us about three months to have our grant proposal approved. The symposium played a critical role in having the program proposal approved in time as well as for establishing a positive tone on our campus before we introduced the program a year later. We used the same strategy later in assisting the university in obtaining a state grant of 4 million dollars for four years in Dec. 2006, which would mainly be used to promote and coordinate the technology education and training in the region.

In addition to the cash-based grants, we had looked for various types of support for our program as well, such as faculty teaching release time, keynote speaker at conferences (without honorarium), internship opportunities, free registrations to conferences, free proprietary computer forensics software for teaching, etc. For instance, Kroll OnTrack, one of the world's leading companies in computer forensics and eDiscovery, did not ask for an honorarium for sending a nationally renowned keynote speaker to our 2004 symposium, which was a significant saving for us. A couple of computer forensics companies have expressed their interest in allowing us to use their proprietary software in teaching some of our computer forensics courses.

We realize that we are a small state university with limited resources and potential to obtain a significant grant. To work around our disadvantages, we have had to be more flexible and open minded in looking for support. We always prepare a summary of our program in terms of enrollments, students' demography, and accomplishments we have had so far. Thus, whenever possible, we will share this information with the people who have influence in our funding. Generally, people need to understand the program and to know what you have done before they are willing to offer their financial support.

3.4 Retention Management

"Can I get a job after I get a degree in computer forensics?" This is the most commonly asked question at our program information meetings. How to respond to a question like this and how to advise students during their study is very critical to the students who are interested in the field as well as important to the retention of the program. We have developed four principles in advising our students, namely "state the facts," "make no promises," "individualize advising," and "ask for feedback."

We told our students upfront that computer forensics is an emerging field, which means, it is a market that is small now but is growing and there will likely be a much bigger market in the future. To prepare them for a situation like this, we recommend our students in having either a two-year degree or a certificate in one of the system-related fields, such as system administration, database administration, or network administration so that they have a relatively broader skill set when they enter into the job market.

It is not recommended to make any promises in predicting the job market in a few years or internship opportunities in order to get more students in the program. We try to avoid making any of these sorts of promises, but rather, we say we are preparing and intending to do certain things. Promises may help increase the enrollment initially but eventually it may come back to haunt you. Lay out the facts and their background, interest, and strength, and then ask them to make the decision for themselves. If a long-term decision is unrealistic to make right away, we usually provide some options for the students to consider one semester at a time. Encouraging students to participate in professional activities has proven to be a very effective way to help them reach their own conclusions.

Computer forensics is a team sport. This is another main concept we have built into our advising. Computer forensics is not just about investigation, but it also includes argument preparation, expert witness, technology visualization, and tech-based legal writing and documentation. So, when we advise our students, we help them figure out their strengths and encourage them to work towards their strengths. Those strengths will help them stand out in a pool of candidates when they are in the job market.

To keep your students in the loop on the curriculum's evolution is one of the best ways to increase the retention rate of the program. Whenever we have an advising meeting with our students, we ask them for any feedback regarding the courses, lab facility, and any other related issues. Their feedback often helps us make quick adjustments to the program. Moreover, we share our department activities on computer forensics with them, such as new equipment purchases, new hiring in the department, or a new book on a particular computer forensics subject. To make students feel like they are part of the program is one of the best ways to keep them in the program.

4. CONCLUSION AND FUTURE WORK

Graduate programs should emphasize more on the future development of computer forensics program if computer forensics professionals and educators want to keep pace with the development in forensic science. What should be the direction in developing computer forensics programs remains a hot topic of discussion in the academic arena as

the current trend tends to have more two-year programs as well as certificate-related programs, which draws a significant contrast with the academic program development in forensic science. As presented in [15] and [1], the ratio of the numbers of MS programs between forensic science and computer forensics is 51 to 3 in the U.S.

The field of computer science has been challenged for the past five years due to significant declining in enrollment. Many have questioned whether computer science as a discipline has reached its end? We are more inclined to believe what Alan Kay said at his lecture in memory of Yahiko Kambayashi-sensei at Kyoto University last January, which is, computing as a discipline is still in its evolution age [17]. We have yet to fully grasp what should or should not be in computer science. Let us continue to explore it.

REFERENCES

[1] Almirall, J. R., *"Education and Research Perspectives: Needs and Opportunities in Trace Evidence Examinations,"* presentation at Trace Evidence Symposium in Clearwater Beach, Florida, Aug 13-16, 2007

[2] Bachelor of Science in Computer Forensics and Digital Investigations, online program, Division of Continuing Professional Studies, Champlain College, Burlington, Vermont, http://digitalforensics.champlain.edu/details_CPS_2008.html, retrieved April 14, 2009.

[3] Certified Information Systems Security Professional, the International Information Systems Security Certification Consortium, Inc. $(ISC)^2$, http://www.isc2.org/cissp/default.aspx, retrieved April 14, 2009.

[4] Computer Forensics Certificate, Forensic Science Center, Marshall University, Huntington, West Virginia, http://forensics.marshall.edu/MISDE/MISDE-CertProg.html, retrieved April 14, 2009.

[5] Computer Forensics Certificate, Lane Department of Computer Science and Electrical Engineering, College of Engineering and Mineral Resources, West Virginia University, Morgantown, West Virginia, http://www.lcsee.cemr.wvu.edu/forensics/, retrieved April 14, 2009

[6] Computer Forensics Option, Bachelor of Science in CIS and Bachelor of Science in Criminal Justice, Missouri Southern State University, Joplin, Missouri, http://www.mssu.edu/schtech/criminaljustice/BSForensics.htm, retrieved April 14, 2009

[7] Computer Investigation and Criminal Justice Major, Computer and Information Science, S'Ambrose University, Davenport, Iowa, http://web.sau.edu/cis/CICJ/CICJMajor.htm, retrieved April 14, 2009

[8] Cyber Forensics, a Masters area of specialization, the College of Technology, Purdue University, West Lafayette, Indiana, http://cyberforensics.purdue.edu/Masters.aspx, retrieved April 14, 2009

[9] Digital Forensics Certificate, Professional and Continuing Education, University of Washington Extension, Seattle, Washington, http://www.extension.washington.edu/ext/certificates/cpf/cpf_gen.asp, retrieved April 14, 2009

[10] Digital Forensics Track in MS and Ph.D. programs, Department of Computer Science and Statistics, College of Arts and Sciences, University of Rhode Island, Kingston, Rhode Island, http://forensics.cs.uri.edu/index.php, retrieved April 14, 2009.

[11] E-Evidence, http://www.e-evidence.info/, retrieved April 14, 2009.

[12] Forensic Computer Investigation Certificate, Henry C. Lee College of Criminal Justice and Forensic Sciences, University of New Haven, West Haven, Connecticut, http://www.newhaven.edu/5805/, retrieved April 14, 2009.

[13] Gottschalk, L., Liu, J., Dathan, B., Fitzgerald, S., and Stein, M., "Computer Forensics Programs in Higher Education: A Preliminary Study," the proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, Missouri, USA, Feb. 23-27, 2005

[14] Hall, Gregory A. and Davis, Wilbon P., "Toward Defining the Intersection of Forensics and Information Technology," International Journal of Digital Evidence, spring 2005, Volume 4, Issue 1

[15] Hankins, R., Uehara, T. and Liu, J., "A Comparative Study of Forensic Science and Computer Forensics," to appear in the proceedings of the Third IEEE International Conference on Secure Software Integration and Reliability Improvement," Shanghai, China, July 8-10, 2009

[16] Information Assurance and Forensics, Bachelor of Technology Program, Institute of Technology, Oklahoma State University, Okmulgee, Oklahoma, http://www.osuit.edu/academics/information_technologies/ba_about.php, retrieved April 14, 2009

[17] Kay, A. C., "Systems Thinking for Children and Adults," a lecture in memory of Yahiko Kambayashi-sensei, Kyoto University, Kyoto, Japan, Jan. 20, 2009

[18] Liu, J. "Developing an Innovative Baccalaureate Program in Computer Forensics," the proceedings of the 36th ASEE/IEEE Frontiers in Education Conference, San Diego, CA, Oct. 28 – 31, 2006

[19] Liu, J., Gottschalk, L., Jian, K., "Textbooks for Computer Forensic Courses: A Preliminary Study," the proceedings of ADFSL 2007 Annual Conference on Digital Forensics, Security, and Law, Arlington, Virginia, USA, April 18-20, 2007

[20] Master of Science in Digital Forensics, College of Engineering and Computer Science, University of Central Florida, Orlando, Florida, http://www.graduatecatalog.ucf.edu/programs/Program.aspx?ID=1160, retrieved April 14, 2009.

[21] Master of Science in Digital Investigation Management, Graduate Studies, Champlain College, Burlington, Vermont, http://extra.champlain.edu/master/msdim/, retrieved April 14, 2009.

[22] Master of Science Program in Forensic Computing, John Jay College of Criminal Justice, The City University of New York, New York City, New York http://web.jjay.cuny.edu/~forensiccomputing/, retrieved April 14, 2009.

[23] Ph.D. Program in Digital Forensics, California Sciences Institute, http://calsci.org/, retrieved April 14, 2009.

MAKING SERVICE ORIENTED ARCHITECTURE RELEVANT USING A MULTIDISCIPLINARY APPROACH*

Thomas Way and Vijay Gehlot
Center of Excellence in Enterprise Technology
Department of Computing Sciences
Villanova University
Villanova, PA 19085
(610) 519-7307
thomas.way@villanova.edu
vijay.gehlot@villanova.edu

ABSTRACT

The use of a Service Oriented Architecture (SOA) approach to the design of enterprise-scale applications is increasing in industry, yet there are few sufficiently multidisciplinary college courses available for students that cover the topic the way students will see it in practice. In this paper, we present the design of a graduate level course that integrates elements of business and engineering into the computer science foundation. The original course design is motivated, and experiential data from the first offering is given. An analysis is provided, leading to guidelines for a relevant SOA course. Modifications for future offerings of the course, including an undergraduate version, are given based on our first offering experiences.

1. INTRODUCTION

Service Oriented Architectures (SOAs), the techniques for developing and deploying loosely coupled, interoperable, implementation independent network services that exchange data with each other [2], are continuing a trend toward increased adoption by industry [1]. The technology is sufficiently mature, and the available software tools make it possible to develop SOA services rapidly. Although the underlying design, development and networking issues are often complex, well-trained software engineers can master SOA and most of the technical challenges are quite solvable. Thus, the key

---

issue preventing broader adoption of SOA is the practice of governance [5], which includes establishing agreed upon standards for interoperability and the significant challenge of balancing the competition of the marketplace with the cooperation and collaboration required for SOA to be successful.

It is into this marketplace of significant but solvable software engineering challenges and identifiable but potentially intractable business practice conundrums that our graduates enter. The best prepared graduates who seek careers in SOA should be astute on the technical issues of enterprise-scale, networked, software design and deployment and also on the motivating business aspects that are manifest in the natural tension between competition and collaboration. Opportunities for education are few, but growing as the demand increases. In addition to a small number of vendor-offered, industry-oriented training seminars, when the proposal to develop this course was first made only Brandeis University offered a course specifically on SOA. In a recent online search, just six universities were found that offered a graduate or undergraduate course specifically on SOA, with a focus almost exclusively on technical aspects.

Software engineering curricula includes technical topics of software tools, methods and design, and typically acknowledges the business aspects needed to manage a software project [7]. The incorporation of sufficiently complex, real-world projects [4] and exposure to industry practices and experts [8] into a software engineering course can provide students with valuable experience applying technical concepts. To fully understand SOA, one also must be facile with its business aspects. The current dearth of formal SOA coursework, coupled with the expertise developed by faculty researchers at our institution gained through a DoD-funded, enterprise-architecture research project, motivated the development of a multidisciplinary course for graduate, and future upper-level undergraduate, computer science students.

2. COURSE DESIGN

This semester-long course incorporates three learning objectives supported by outcomes and a broad selection of content, met one evening per week for 2¾ hours, and had an expectation of student's performing significant outside-of-class work.

2.1 Learning Objectives

Three learning objectives, and corresponding outcomes, covering computer science, engineering and business aspects of SOA, were developed to provide students with the technical and theoretical foundations of SOA.

Objective 1 is to survey the topic of Service Oriented Architecture (SOA), including related aspects of computer science, engineering and business, and how these all relate within the larger topic of software engineering. Outcomes are that students will demonstrate an understanding of a broad range of SOA topics through a wide variety of course content and activities.

Objective 2 is to provide experience developing SOA applications individually or as part of a team. This objective and its outcomes were accomplished with a large-scale, team-based project, and numerous related lectures, laboratory exercises and reading

assignments, giving students first-hand experience with the business and people aspects involved with managing a team working toward a common goal and of encouraging collaboration among competitors, in this case among other teams in the class.

Objective 3 is to learn about modeling and simulation of Service Oriented Architecture, and gain experience using current, professional modeling and simulation software. This third objective and its outcomes were accomplished using CPN Tools and OPNET modeling software to provide hands-on application of SOA.

2.2 Content Development

Lacking a single, comprehensive SOA textbook and little existing classroom tested course material presenting the subject from computer science, engineering and business perspectives, a significant quantity and variety of course content was developed.

Two **textbooks** on SOA were selected to provide a foundation, broad context and practical implementation techniques on the subject, although the texts were augmented with online readings. The texts used were *Service-Oriented Architecture (SOA): Concepts, Technology, and Design* by Thomas Erl [2] and *SOA: Using Java Web Services* by Mark D. Hansen [3].

Selected readings from online sources supported lecture topics. Online sources were used to provide material that is as current and relevant as possible.

Lectures on technical and business aspects of SOA were presented for a portion of each class meeting to introduce and reinforce material from assigned readings, and were made available to students ahead of time to assist with note-taking.

Guest speakers, who were paid a small stipend, presented first-hand accounts of their professional, insider experiences working with technical, engineering and business aspects of SOA in industry. Scheduling suitable guest speakers was very time-consuming, and a number of ideal speakers were unavailable at class time.

Laboratory activities enabled learning of the practical aspects of SOA software development using industry standard tools, with a focus on NetBeans. Students also modeled SOA using professional modeling software CPN Tools and OPNET.

Topic presentations were assigned, with each student selecting a relevant, advanced SOA topic and preparing and delivering a short talk to the class on that topic, giving students deeper exposure to a variety of advanced topics.

Team-based projects had students practicing real-world collaborative skills in the development of models and SOA services, equipping students with the technical skills and experience needed to work together on the final project, where student teams created interoperable services as part of a "smart house" simulation. Teams implemented services to produce and consume "live" house environment management data such as banking balances, grocery ordering, and power use billing. This project gave students first-hand experience with governance and collaboration issues that are critical to the use of SOA services in the marketplace. Details of this project, as well as plans for a variety of additional projects, are omitted here for space considerations but are available from the authors.

3. ASSESSMENT

To assess student learning and measure the effectiveness of the course design, the 13 graduate students in the course were given an initial survey to gather basic demographic information. Of the 13 students, 11 had earned a BS degree in computer science, with one BS each in business administration and finance. Student experience with software engineering was distributed relatively evenly, from no experience to more than 5 years experience. The majority indicated that they had some professional business management experience with 3 having served as managers and 8 (including the 3 managers) having performed project leadership duties. The remaining 5 students reported no previous business experience. All had experience programming in Java, most also had experience with C/C++, and about half with web applications and a variety of other programming languages.

To measure learning, students were surveyed at the start and finish of the semester regarding their understanding of SOA with regards to software design, business issues, engineering, networking and hardware issues, and general terminology or SOA literacy. For each topic, students began the semester distributed roughly evenly across the levels of understanding, from poor to excellent. At the conclusion of the semester, all students reported significant improvement to their understanding, with all winding up in the good to excellent range (Fig. 1).

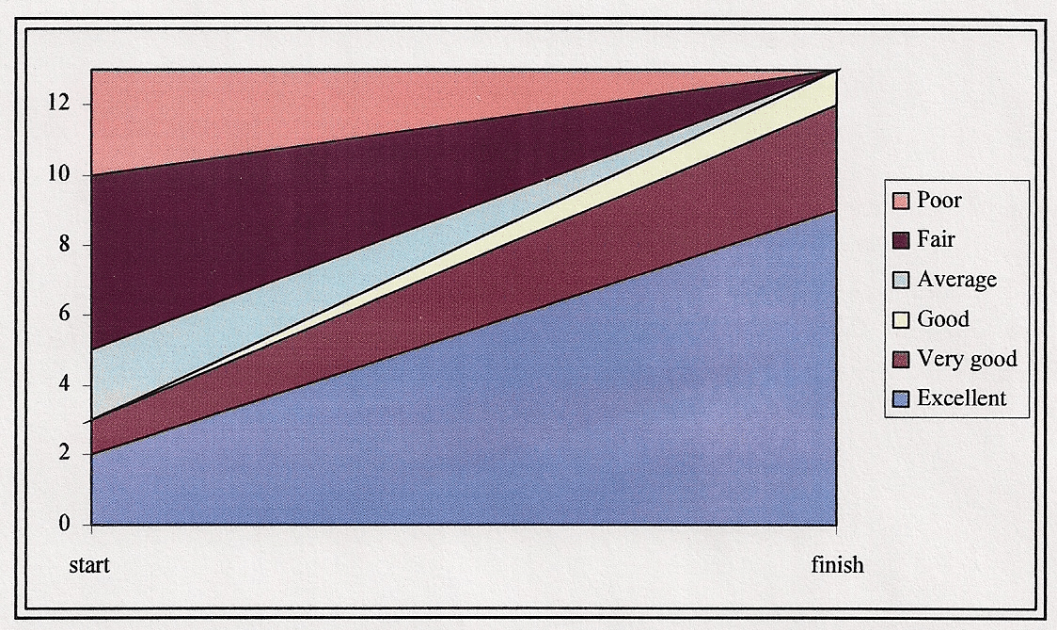


Fig. 1. Student self-assessment of familiarity with SOA terminology and general SOA literacy at start and finish of semester, a typical distribution for all topics surveyed.

As part of the semester-concluding survey, students were asked for feedback on various aspects of course content, specifically: topic coverage, reading material, guest lectures, exposure to current tools and technologies and SOA programming exercises. All 13 students reported that topic coverage was adequate, with numerous comments about liking the wide variety of topics. One student suggested an increased emphasis on SOA business process management to benefit students entering the workforce.

Reading material was judged adequate by 11 students, and inadequate by 2, although for an unexpected reason. Some felt that the textbook was not needed because of the wide availability of material online. The 2 inadequate ratings resulted from material being too advanced for students with less technical backgrounds. Two guest lecturers spoke during the semester, and 10 students found them to be adequate, serving as a "good break" from the other course activities, with inadequacies including topic repetition overly advanced material.

Current software tools and SOA technologies were covered, with 12 students finding it adequate. Students felt they gained a broad appreciation of the technology, security and business issues, and found the projects to be the most valuable activity to improve understanding. Shortcomings included the steep SOA learning curve, need for more lab time, and a desire for more coverage of SOA programming in lectures. Students suggested a desire for more and earlier programming projects, better team organization of the projects, fewer or no student presentations, more in-class activities and labs, and a better defined structure for the overall course.

4. ANALYSIS AND ADAPTATIONS

Student satisfaction with the course and its content was very high, with a few noted exceptions. Reading assignments from the textbook and online sources were judged valuable, but too advanced at times, and the textbooks were not necessary due to the preponderance of free and up-to-date reading material online. As a result of this feedback, a new text for the second offering of the course. *Applied SOA: Service-Oriented Architecture and Design Strategies* by Michael Rosen, et al. [6], was chosen for its accessible presentation of material, currency, good overview and design-oriented approach.

Lectures, homework and laboratory assignments were appreciated, with students suggesting more hands-on work early in the semester. In order to better prepare all students for the semester project, laboratory activities will be assigned earlier in the next offering to get students working with the technology earlier. Guest speakers were important to students, yet time-consuming to identify and schedule, so in the future we will begin earlier to line them up. Programming projects, in particular the large-scale final project, were viewed by students as invaluable to their learning, with the plan to introduce them earlier in a future offering.

Colleagues in academia and industry expressed satisfaction with the preparation students were receiving in the course. Viewed as most valuable to industry was the multidisciplinary "big picture" that includes business and engineering, and modeling and simulation (i.e., Formal Methods) as risk-management design approaches. As a result, we are producing a variety of lab assignments in modeling and simulation suitable for all levels of computer science students.

5. CONCLUSIONS

This paper reported on the design and first offering of a course for graduate students on the conjunction of computer science, engineering and business aspects of the emerging science of Service Oriented Architectures. Student learning was significant and all objectives were met. There was a variety of shortcomings and ideas for improvement expressed by the students, and by the instructor, and this feedback was actively used to design a second offering of the course which is currently in progress. We plan a comprehensive follow-up study to compare the results of the second offering with the version reported in this paper, and to continue to incorporate feedback of all sorts into further development of the course.

Students particularly appreciated any attempts to provide real-world experiences, through topical lectures, large scale projects and guest speakers, and viewed such content as invaluable foundation for their current or future careers. Because the course was offered at the graduate level only due to course scheduling constraints, a number of options for adapting the course to upper level undergraduates were noted and will be incorporated. For example, undergraduates will be given more, shorter programming projects, less advanced readings, and additional introductory material.

As a graduate course, a multidisciplinary course on SOA fits well into our Software Engineering and general computer science MS sequences. For undergraduates, we suggest that it be offered as an elective for a Software Engineering track. We are currently developing lab modules that cover the technology, business and engineering elements of SOA, with a particular emphasis on modeling and simulation that can be integrated appropriately throughout a computer science curriculum. Because SOA is increasingly widespread in industry, we believe that the best prepared graduates should have ample exposure to this important subject within the context of the multidisciplinary nature of SOA.

6. ACKNOWLEDGEMENTS

Support for the development of this course was provided by a Villanova Institute for Teaching and Learning (VITAL) Minigrant, with thanks to Dr. Carol Weiss for her suggestions and comments that helped with the initial course offering. Thanks also are due to members of the Villanova Center of Excellence in Enterprise Technology.

7. REFERENCES

[1] Michael J. Carey. SOA What? Computer, 41:3, March 2008, pp. 92-94.

[2] Thomas Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Upper Saddle River, NJ, 2005.

[3] Mark D. Hansen. SOA: Using Java Web Services. Prentice-Hall, Upper Saddle River, NJ, 2007.

[4] Chang Liu. Enriching software engineering courses with service-learning projects and the open-source approach. In ICSE ’05: Proceedings of the 27th international conference on Software engineering, 2005, pp 613–614.

[5] Eric A. Marks. Service-Oriented Architecture (SOA) Governance for the Services Driven Enterprise. Wiley Publishing, 2008.

[6] Michael Rosen, Boris Lublinsky, Kevin T. Smith and Marc J. Balcer. Applied SOA: Service-Oriented Architecture and Design Strategies. Wiley Publishing, Inc., June 2008.

[7] Mary Shaw. Software engineering education: A roadmap. In: A. Finkelstein (ed.): The Future of Software Engineering: the 22nd International Conference on Software Engineering, New York, NY, ACM Press, 2000, pp. 371-380.

[8] Rayford B. Jr. Vaughn and Jeffrey Carver. "Position Paper: The Importance of Experience with Industry in Software Engineering Education. Conference on Software Engineering Education and Training Workshops, 2006, pp. 19.

GREENFOOT – INTRODUCTION TO JAVA WITH GAMES AND SIMULATIONS*

TUTORIAL PRESENTATION

Michael Kölling
University of Kent

ABSTRACT

- This tutorial will demonstrate Greenfoot, a programming environment developed by the creators of BlueJ, that allows teaching of object-oriented programming concepts – using Java – in a highly engaging and motivating context.
- One of the major problems in teaching computing today is the lack of interest in many young people. Computing suffers from a disastrous reputation (to a great extent wrongly). It is perceived as boring, geeky, unrewarding, and antisocial. Popular preconceptions of overweight male teenagers with thick glasses and skin problems sitting alone in windowless cellar rooms in front of a computer screen with pizza boxes strewn around them do little to attract a more diverse group to computer science.
- Greenfoot is designed to enable teachers to bring fun and engagement back into computing, while teaching real programming concepts, and without trivializing the subject matter. Students quickly start to program graphical, interactive applications, such as games and simulations. A very diverse set of possible projects and sample programs serves to attract groups of students who would not normally take an interest in programming.
- The Greenfoot environment provides tools for students to achieve real successes quickly, while providing tools for teachers to illustrate and discuss fundamental concepts of object-oriented programming. Once completed, student work can easily be published and shared on the Internet.
- Greenfoot should be of interest to anyone teaching Java, especially in early programming courses, at schools and colleges.
- Greenfoot is available from www.greenfoot.org. The tutorial is practically oriented and allows participants to use Greenfoot in their classroom immediately. Audience members with laptops will be encouraged to play along during the tutorial, giving a chance to get some first-impression, hands-on experience with the software during the session.

INTRODUCING MULTI-CORE PROGRAMMING INTO THE LOWER-LEVEL CURRICULUM: AN INCREMENTAL APPROACH*

TUTORIAL PRESENTATION

Timothy J. McGuire
Sam Houston State University

ABSTRACT

Historically, improved hardware processing power has been due to increases in clock speed. Recently, however, chip manufacturers have begun to increase overall processing power by adding additional processing cores to the microprocessor package, while clock speeds have remained virtually unchanged. New processors will eventually come in heterogeneous configurations such as combinations of high- and low-power cores, graphical processors, etc. These future configurations are being termed "many-core" architectures. For software developers, this change raises many challenges. No longer will programmers be able to "ride the wave" of increasing performance without explicitly taking advantage of parallelism. Currently, only a very small proportion of developers have expertise in parallel programming. Software developers need new programming models, tools, and abstraction by the operating system to handle concurrency and complexity of numerous processors. For computer science educators, this change will also require radical shifts in the way computer science is taught. Parallelism will need to be introduced early in the curriculum, preferably in the CS1/CS2 sequence.

Despite the fact that a small group of programmers have been programming parallel applications for many years, most programmers have only a cursory understanding of the issues involved in developing multi-core applications. As machines with 32 or more cores become commonplace, students must gain a working knowledge of how to develop parallel programs. It seems evident that students must be exposed to concurrency throughout the curriculum, beginning with the introductory CS sequence.

While new parallel languages will be developed (as well as extensions to existing languages) it is not yet evident which direction that development will head. Message-passing (MPI) and threads (POSIX threads and Java threads) have been the methods of choice for teaching parallelism at the undergraduate level. Since multi-core systems use shared-memory, and the message-passing model is more suited for clusters than it is for shared-memory systems, it is not a likely candidate for multi-core and many- core

---

systems. Java threads have the advantage of being built-in, so the language can be used for parallel programming on multi-core machines. Although threads may be seen as a small syntactic extension to sequential processing, as a computational model, they are non-deterministic, and the programmer's task when using them is to "prune" that non-determinism. POSIX threads are implemented by a standard library providing a set of C calls for writing multithreaded code. They have the same difficulties in programming that Java threads do.

OpenMP (Open Multi-Processing) is an API which supports shared memory multiprocessing. It consists of a set of compiler directives and a library of support functions. Open MP works in conjunction with Fortran, C, and C++. Compared to Pthreads, OpenMP allows for a higher level of abstraction, allowing a programmer to partition a program into serial regions and parallel regions (rather than a set of concurrently executing threads.) It also provides intuitive synchronization constructs.

This tutorial will survey the parallel programming landscape, summarize the OpenMP approach to multi-threading, and illustrate how it can be used to introduce parallelism into the lower-level curriculum to novice or intermediate C programmers.

A STATE DIAGRAM CREATION AND CODE GENERATION TOOL FOR ROBOT PROGRAMMING*

Sambit Bhattacharya and Bogdan Denny Czejdo
Department of Mathematics and Computer Science,
Fayetteville State University
Fayetteville, NC 28301
910 672-1156 & 2466
{sbhattac, bczejdo}@uncfsu.edu

ABSTRACT

This paper discusses a new tool for the design and code generation of reactive programs for robots based on the state diagram design methodology. The tool's simple graphical interface enables interactive diagram creation followed by code generation and testing; furthermore these steps can be cycled to incrementally modify the graphical design and generated code until the desired robot behavior is achieved. We have observed that using robots provides a strong motivation for learning programming skills; however, the rapidly growing complexity of code for reactive behavior programming can result in the student losing track of the original problem and the correct design that could lead to its solution. Our tool addresses this challenge by constraining the design within a an intuitive graphical framework; the design, once it has been created, provides a frame of reference which can lead to more productive ways of design and code modification.

INTRODUCTION

Research in robot behavior design has produced diverse approaches which can be broadly classified as reactive, planning, deliberative and hybrid [12]. Of these, reactive behavior in which the robot responds directly to the environment is well suited for undergraduate programming coursework [3, 13]. Furthermore reactive behavior can be conveniently modeled by state diagrams [6, 8, 9]. With this in mind we have created a new tool with a simple graphical interface for interactive design of state diagrams and

---

code generation. Such an interface is very useful in an educational setting since it allows students to incrementally modify the graphical design and generated code until the desired robot behavior is achieved.

There exist several robot programming platforms which have graphical tools to define robot behavior [10, 11]. However these platforms are more suited for specialized research and are difficult for undergraduate classroom adoption. Also these platforms are tied to specific robot architectures and do not have support for other existing architectures such as the IPRE robot kit [1, 2, 13] which we use in our classroom.

Using robots provides a strong motivation for learning programming skills [4, 5, 14]; however, the rapidly growing complexity of code for reactive behavior programming can result in the student losing track of the problem and the correct design that could lead to its solution. Our tool addresses this challenge by constraining the design within a an intuitive graphical framework; the design, once it has been created, provides a frame of reference which can lead to more productive ways of design and code modification. The students can learn how to modify the robot's behavior by adding new states and/or transitions to the existing state diagrams. It is easier for them to enrich the robot behavior without changing the existing structures. The tool is downloadable for free from [15].

STATE DIAGRAM CREATION

We will first discuss the state diagram creation feature. To specify state diagrams in our tool we use the notation based on Universal Modeling Language (UML) [9] where a state is indicated by a rounded box and a transition is indicated by an arrow with a label. The first part of the transition label (before the slash) specifies the trigger and the part

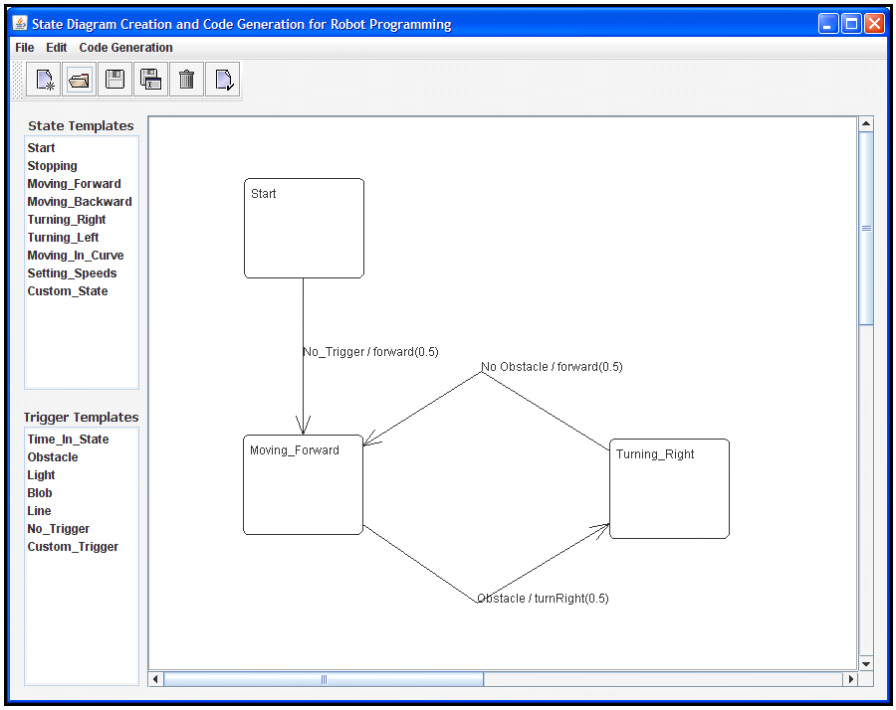


Figure 1: main window of tool showing obstacle avoidance state diagram.

after the slash specifies the action to be performed during the transition. The trigger is a Boolean condition and the action is initiated during the transition and maintained by the next state. E.g. in Figure 1, Moving_Forward transitions to Turning_Right when the "Obstacle" condition is true; the action initiated is the "turnRight" command which is maintained by the Turning_Right state until a new transition occurs.

Our state diagram creation and code generation tool has a simple graphical user interface shown in Figure 1. Besides the menus and toolbar, the interface consists of a scrollable drawing canvas and two scrollable lists. The top list on the left lets users select from existing state templates or create a custom state template. Table 1A provides a summary of the features of these state templates. Similarly the bottom list on the left provides existing trigger templates and an option to create a custom built one. The information for trigger templates is summarized in Table 1B. The user drops a new state into the canvas by selecting from one of the state templates and clicking anywhere in the drawing area; for a single diagram there can be only one Start state and multiple states from the same template are conveniently numbered in increasing order. A trigger is drawn by selecting a trigger template followed a sequence of two mouse clicks: first one within the area of the "from" state shape and the second within the "to" state shape. Our tool fully supports interactive diagram modification through point, click and drag; state shapes can be moved as needed and the geometric characteristics of trigger shapes connected to them will be automatically updated, or the triggers can be split into line segments for better presentation of information.

Let us consider a state diagram for the movement of a simple robot that avoids obstacles. The state diagram is shown in Figure 1. In addition to the initial Start state, the diagram has two states: Moving_Forward and Turning_Right. These states describe the two step sequence for the robot movement that is repeated indefinitely. More specifically, the robot moves forward until an obstacle is encountered, then continues turning right until obstacle cannot be detected any more. After that it continues moving forward until another obstacle is encountered, and so on.

This sequence can be described in more detail as below:

1. Initially the robot is in the Start state and the transition with no trigger to Moving_Forward state takes place immediately. During the time of transition the *forward (0.5)* command is executed. This built-in command engages the motors to move the robot half-speed forward. It is an important part of the semantics of the forward command that it sets the motor speed and the motors will continue to be engaged until a new command deactivates the older settings.

2. The robot stays in the Moving_Forward state (and continues moving forward half-speed) until it detects an obstacle. Detecting the obstacle triggers the invocation of the *turnRight(0.5)* command and the transition to the Turning_Right state. This built-in command engages the motors to start turning the robot around clockwise at half-speed. Similar to the *forward* command, the *turnRight* command causes the motors to be engaged until any new movement command deactivates the older settings.

3. The robot stays in Turning_Right state (and continues turning clockwise half-speed) until it detects no obstacle. Detecting no obstacle triggers the invocation of the *forward(0.5)* command and transition to the Moving_Forward state. This built-in command engages the motors to start moving the robot half-speed forward as before.

Again, the motors will continue to be engaged until any new movement command deactivates the older settings.

Trigger Template	**Condition**
Time_In_State	not timeRemaining(<duration arg.>)
Obstacle	obstacle( )
Light	light( )
Blob	blob( )
Line	line( )
No_Trigger	True
Custom_Trigger	To be specified by user

State Template	**Action when transitioning to state**
Start	No action
Stopping	stop( )
Moving_Forward	forward(<speed arg.>)
Moving_Backward	backward(<speed arg.>)
Turning_Right	turnRight(<speed arg.>)
Turning_Left	turnLeft(<speed arg.>)
Moving_In_Curve	motors(<left speed arg.>, <right speed arg.>)
Setting_Speeds	set_wheel_speeds( )
Custom_State	To be specified by user

Table 1: (A) summary of state templates (B) summary of trigger templates settings.

In addition to the convenience of state diagram creation our tool facilitates interactive code modification while the user is in the process of building the diagram. These features are frequently used when the trigger or state being inserted is based upon a certain template but different enough to require code modification. A good example is the "No Obstacle" trigger in Figure 1, which is similar to the "Obstacle" template with only the Boolean condition inverted. After drawing the "Obstacle" trigger from Turning_Right to Moving_Forward, the user simply goes through the following steps: (i) selects the trigger shape through mouse click, (ii) chooses the "Properties" submenu item or toolbar button after which the properties window shown in Figure 2A is displayed; (iii) the user then modifies the trigger properties to get the "No Obstacle" trigger where the modified properties window is shown in Figure 2B.

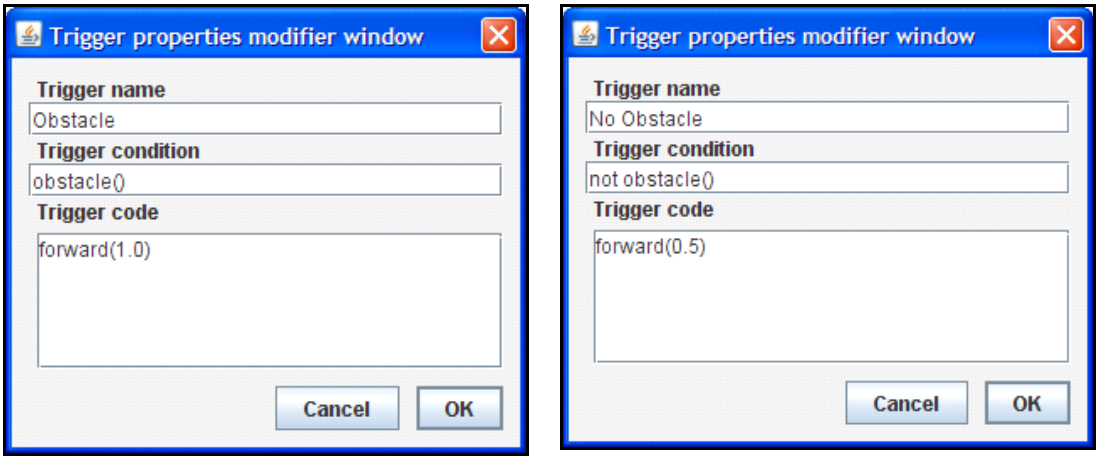


Figure 2: (A) Initial trigger properties and (B) Modified trigger properties

CODE GENERATION

Our tool converts state diagrams into a programming code, more specifically into Python code. Many implementations were proposed for converting state diagrams into code: some based on object-oriented paradigm [7, 14] or an imperative programming paradigm [13]. Our generation tool can convert state diagrams into the code based on any of these paradigms. However, in this paper we concentrate our discussion of conversion of state diagrams into Python code using the imperative programming paradigm.

In order to describe the algorithm for the code generation let us assume that in the state diagram the state names are identified as StateName1, StateName2, … StateNameN, and state transitions out of each state K are identified by TransitionK1, TransitionK2, … etc. Let us also assume that each transition TransitionKL has two components TriggerKL and ActionWhenTriggerKL and that each TransitionKL leads to NewStateKL.

The algorithm contains several parts. The first part is to generate functions for each state as is described below:

For each state StateNameK

a) generate an initial template of the function execute_StateNameK with an infinite loop (initially with empty body).

b) Identify all transitions out of the state StateNameK and for each transition generate an *if* statement with the trigger as a condition and action for each trigger as a body of the if statement as shown in Figure 3. The *if* statement body also contains the return statement with the function name for the next state. If the transition is based on the "No_Trigger" template then only the action part is appended. The order of transitions as entered by the user determines order of execution of *if* statements and priority of transitions (in case of several triggers fired the same time).

c) Place the sequence of *if* statements constructed in b) within the infinite loop constructed in a) as shown in Figure 3.

```
def executeStateNameK():
  while True:
    if TriggerK1:
      ActionWhenTriggerK1
      return execute_NewStateK1
    if TriggerK2:
      ActionWhenTriggerK2
      return execute_NewStateK2
    if TriggerK3():
      .....
```

Figure 3. Complete Python code for each state.

The second part of the algorithm is to create a code for a State Manager i.e. the main driver of the generated program. It is a universal driver responsible to invoke onetransition after another in the loop where one transition corresponds to one cycle of the while loop. This driver has a fixed structure and is implemented by the execute_StateManager function as shown in Figure 4.

It is assumed that the StartState is an initial state and therefore the corresponding function name generated in the algorithm part 1a) is execute_StartState. The name of this function is assigned to the variable execute_current_state. This variable contains a function name so when appended with "()" it becomes a function call as shown inside the loop in Figure 4. Therefore, in the first cycle of the loop the execute_StartState() function is called. The returned value is the name of the function corresponding to the next state. This value will be used in the next cycle to cause the next transition.

```
def execute_StateManager():

  execute_current_state = execute_StartState
  while True:
    execute_current_state = execute_current_state()
```

Figure 4. Complete Python code for State manager.

Let us look at the example of the state diagram in Figure 1. For each state Start, Moving_Forward, and Turning_Right the function is generated as shown in Figure 5. The Start state has only one transition out of the state and therefore the function execute_StartState() has an infinite loop with no *if* statements – only one action *forward(0.5)*. This function returns the function name *execute_Moving_ForwardState*. The *Moving_Forward* state has only one transition out of the state triggered by an obstacle. Therefore the function *execute_Moving_ForwardState* () has a infinite loop with one if statement containing turnRight(0.5) and return of the function name execute_Turning_RightState. Similarly the definition of function execute_Turning_RightState() is generated with one if statement containing one action

forward(0.5) and returning the function name *execute_Moving_ForwardState*. The State Manager function is appended as described above.

```
def execute_StartState():

  while True:
    forward(0.5)
    return execute_Moving_ForwardState

def execute_Moving_ForwardState():

  while True:
    if obstacle():
      turnRight(0.5)
      return execute_Turning_RightState

def execute_Turning_RightState():

  while True:
    if not obstacle():
      forward(0.5)
      return execute_Moving_ForwardState

def execute_StateManager():

  execute_current_state = execute_StartState
  while True:
    execute_current_state = execute_current_state()
```

Figure 5. Python code for the diagram in Figure 1.

.

CONCLUSIONS

We have described a new state diagram creation and code generation tool for robot programming which can be adopted in early programming classes. The students can learn how to modify the robot's behavior by adding new states and/or transitions to the existing state diagrams. It is easier for them to enrich the robot behavior without changing the existing structures. We will continue to work on improving this tool to support multi-level diagrams and higher level constraints on diagrams.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the Institute for Personal Robotics in Education (IPRE) and the Belk Foundation for financially supporting our course re-design efforts which supported this research.

REFERENCES

[1] Kumar, D., Blank, D., Balch, T., O'Hara, K., Guzdial, M., Tansley, S., Engaging computing students with AI and robotics, *AAAI Spring Symposium Series, presented at the Symposium on Using AI to Motivate Greater Participation in Computer Science, tech. report SS-08-08,* AAAI Press, 2008.

[2] Blank, D., Kumar, D., Marshall, J., Meeden, L., Advanced robotics projects for undergraduate students, *AAAI Spring Symposium Series, presented at the Symposium on Robots and Robot Venues: Resources for AI Education, tech. report SS-07-09,* 10-15, AAAI Press, 2007.

[3] Blank, D., Robots make computer science personal, *Communications of the ACM,* 49, (12), 2006.

[4] Turner, C., Ford, K., Dobbs, S., Suri, N., Hayes, P., Robots in the classroom, *Proceedings of the Ninth Florida Artificial Intelligence Research Symposium (FLAIRS '96),* Florida AI Research Society, 497-500, 1996.

[5] Maxwell, B.A., and Meeden, L.A., Integrating robotics research with undergraduate education, *IEEE Intelligent Systems,* 2-7, November/December 2000.

[6] Baszun, M., Miescicki, J., and Czejdo, B., Multilevel modeling of iterative engineering design of remote robot system, *Proceedings of The Second World Conference on Integrated Design and Process Technology,* Austin, 1996.

[7] Czejdo, B. and Messa., K., Generating Smalltalk code from graphical operations and rules, *Proceedings of the IEEE Symposium on Visual Languages,* Bergen, Norway, 1993.

[8] Embley, D., Kurtz, B., Woodfield, S., *Object-Oriented Systems Analysis – a model driven approach,* Prentice Hall, New Jersey, 1992.

[9] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., *Object-Oriented Modeling and Design,* Prentice Hall, New Jersey, 1990.

[10] http://www.tekkotsu.org/

[11] http://msdn.microsoft.com/en-us/robotics/default.aspx

[12] Murphy, R., *Introduction to AI Robotics,* MIT Press, 2000

[13] Kumar, D., (Editor), *Learning Computing with Robots (Python),* Institute for Personal Robots in Education; 1st edition (2007)

[14] Czejdo, B. D. and Bhattacharya, S., Programming robots with state diagrams, *Journal of Computing Sciences in Colleges,* Vol. 24 (5), 19-26, 2009.

[15] http://faculty.uncfsu.edu/sbhattac/downloads.htm

ROBOTS IN THE CLASSROOM … AND THE DORM ROOM*

Jennifer S. Kay
Computer Science Department
Rowan University
201 Mullica Hill Road
Glassboro, NJ 08028
(856) 256-4593
kay@rowan.edu

ABSTRACT

The purpose of this paper is twofold. First, to argue that despite some disappointing results in using robots in the computer science classroom in the past, we should not yet conclude that robots do not belong there. Second, to present the results of a small pilot study comparing two approaches to teaching an introductory computer science class – one traditional and one which used robots. While the study is not sufficiently controlled to be considered proof of success, initial results are compelling and support the need for further investigation.

1. INTRODUCTION

The notion of introducing robotics into the traditional computer science curriculum is a compelling one. Anyone who has brought a robot into a classroom knows the level of excitement that this tool brings to a classroom full of even the most unenthusiastic students. Robots have the potential to make our current students more engaged in their projects.

Robots also have the potential to attract students to a field they might not otherwise have considered. Many students graduate from high school with no understanding of what the field of computer science is actually about. Classes with robots may help bring some of them into our general education courses and subsequently into our major.

However, several studies have produced both qualitative and quantitative evidence that classes with robot assignments can be frustrating, and even worse, detrimental to student performance.

The purpose of this paper is twofold. First, to argue that despite these disappointing results we should not yet conclude that robots do not belong in the computer science classroom. The greatest source of frustration for students seems to be the lack of access to robots, and with the newest generation of robots that are approaching the price of an (admittedly expensive) text book, there is greater opportunity for students to have their robot alongside their laptop in the dorm room. Second, to present the results of a small pilot study performed at Rowan University in which two introductory programming courses for non-majors were compared, one traditional course and one that used robots to introduce the same concepts. While the study is not sufficiently controlled to be considered proof of success, initial results are compelling and support the need for further study.

2. ROBOTS IN THE CLASSROOM

Robots have been used in a wide variety of computer science classes at the undergraduate level, from introductory courses for non-majors, to advanced electives for majors. [1] [2] [3] [5][6] [7][8] [9]. But they have not always been successful. In an impressive study, Fagin & Merkle evaluated the performance of 938 students taking a core computing course required of all students at the U.S. Air Force Academy. [2] Roughly one-quarter of these students were in special robotics sections of the class in which students used Lego Mindstorms Robots in 4 out of the 6 laboratory exercises. Much to the dismay of the robotics community, test scores were lower in the robotics sections of the course than in the traditional sections, and the use of robots did not affect students' choice of discipline.

Students in the robot sections of this class were limited to assigned lab times to work on their robot projects. Fagin and Merkle concluded that this was the key factor in the poor performance of these students, and suggest the need for a simulator, or the ability to "check out" robots so that students could work on the projects on their own schedules.

In another study, 35 students in a senior-level course were required to share a single Khepera robot for a graphical user interface assignment. [1] While the experience was considered to be successful, students reported that the majority of the problems that they encountered had to do with access to the robot. Many of the students would have preferred to work late at night when the server and robot were not available.

3. ROBOTS IN THE DORM ROOM

One of the newest and cheapest robot platforms designed for education is the IPRE robot which consists of a Parallax Scribbler Robot paired with a plug-in Lancet Fluke board. [4] Programmed in Python, it is designed for introductory computer science students. Initial reports on its use at Georgia Tech and Bryn Mawr indicate that in contrast to Fagin and Merkle's results, that the use of these robots in their classes does no harm. [9]

In the Fall semester of 2008, the author taught three sections of Introductory Programming courses for non-majors, two of the sections were traditional courses taught in Visual Basic, and the third was taught in Python using the IPRE robot base. At a cost of around $150, the IPRE robot is still too expensive to be attractive to non-majors at

Rowan University, but thanks to an award from IPRE, the author was able to purchase enough robots to loan to students for the semester.

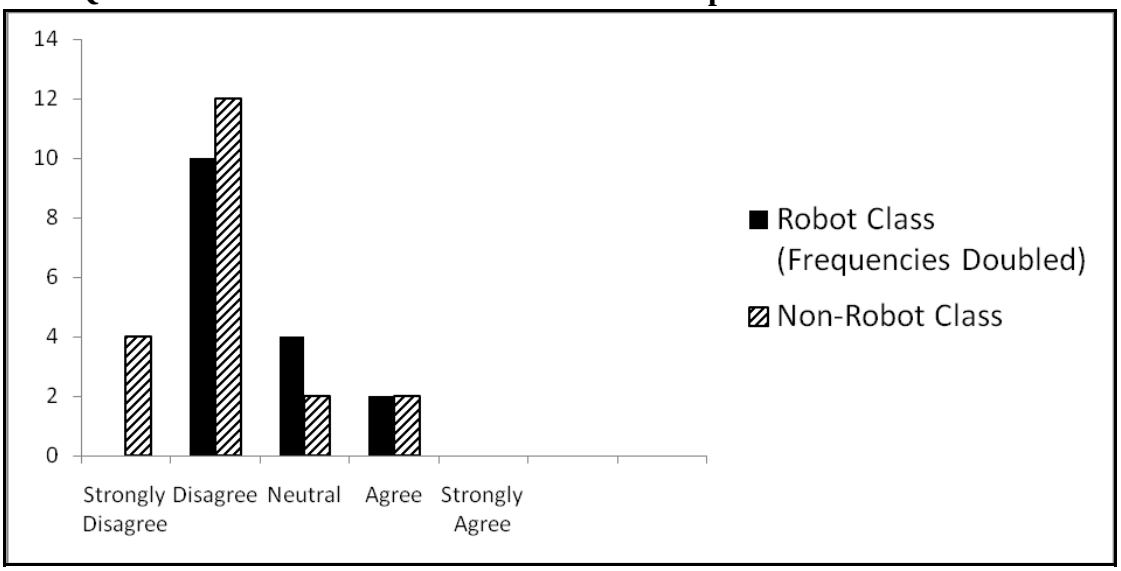
The goal for both versions of these courses is simple: by the end of the semester, students should be comfortable with conditional statements, loops, and functions. The non-robot classes were made up of a variety of majors, primarily math, science, and business, and most of the students were required to take this course as a part of their major. The robot class consisted entirely of students who had yet to declare their major, but who had math SAT scores of at least 560 (560 was chosen because it is the minimum math SAT score required to be admitted to the Computer Science Major).

4. SOME COMPELLING RESULTS

Students in all 3 classes were surveyed at the beginning and the end of the semester. In the non-robot class, 20 out of the 44 students responded to the final survey, in the robot class 9 out of 10 students responded. The graphs that follow are from questions administered at the end of the semester. In the graphs that follow, the robot data frequencies have been doubled for visualization purposes. In other words, 3 students in the robotics class are illustrated as 6 on the graph so that the two classes are easier to compare.

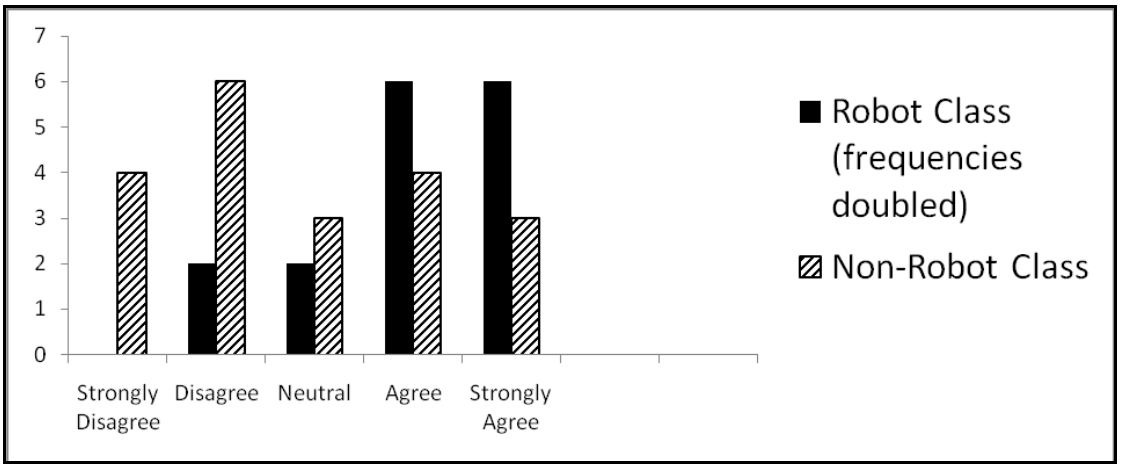
The first graph (Question 1, below) looks at the original students' motivations for taking the classes. Students in both the robotics and non-robotics sections seem, alas, to be equally unmotivated by the desire to learn more about computer science is all about.

Question 1: I took this course to see what computer science is all about



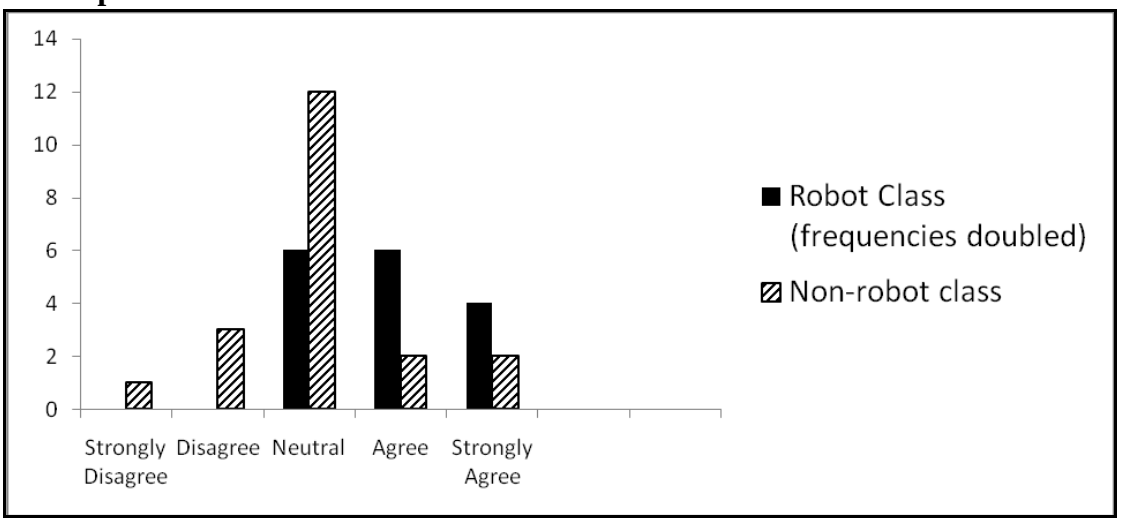
The second graph (Question 2, below) looks at whether students wrote a program during the course of the semester that was not an assignment for the class. Now the results get more interesting. The non-robot class is spread across the graph, with a bit of an emphasis on the strongly disagree/disagree responses. In contrast, the students in the robot classes tended toward agreement, with only 1 out of the 8 students disagreeing, and 6 out of the 8 agreeing/strongly agreeing. Perhaps including robots in the class increased the students' motivation outside of class.

Question 2: During the class, I wrote a program that was not an assignment for this class



The final graph (Question 3, below) addresses the question of whether robots might aid as a recruiting tool for our courses and our major. Students were asked whether their experiences in their class caused them to decide to take another computer science class. Students in the non-robot classes were predominantly neutral. In contrast, students in the robot class were more inclined to agree.

Question 3: My experiences in this class caused me to decide to take another computer science class



5. CONCLUSIONS

While it is important to emphasize that these data are not sufficient to declare that robots should be in the computer science classroom, they are compelling. Students in the robot and non-robot classes seem to have been equally uninterested in computer science when they signed up for the course. But students in the robot course seem more motivated to write programs outside of class that are not required, and to be more interested in taking another computer science class.

Robots are fun and engaging, for teachers as well as students. At a minimum, Summet et. al. have shown that robots can be used and do no harm. Further study is clearly needed, and hopefully the data from my own classes will be mirrored in future statistically significant studies that demonstrate that robots can, in fact, aid in computer science education.

6. REFERENCES

[1] Challinger, J., Efficient use of robots in the undergraduate curriculum. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, St. Louis, Missouri, : ACM, 2005.

[2] Fagin, B. S. and Merkle, L., Quantitative analysis of the effects of robots on introductory Computer Science education, *J. Educ. Resour. Comput. 2*, (4), 2002.

[3] Goldweber, M., Congdon, C., Fagin, B., Hwang, D., and Klassner, F. The use of robots in the undergraduate curriculum: experience reports. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, Charlotte, North Carolina: ACM, 2001.

[4] Myro Hardware – IPRE Wiki, http://wiki.roboteducation.org/Myro_Hardware#IPRE_Scribbler, accessed April 2009.

[5] Kay, J. S. Teaching robotics from a computer science perspective. *J. Comput. Small Coll.*, 19, (2), 329-336, 2003.

[6] Klassner, F., Enhancing lisp instruction with RCXLisp and robotics. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, Virginia: ACM, 2004.

[7] Lauwers, T., Nourbakhsh, I., and Hamner, E., CSbots: design and deployment of a robot designed for the CS1 classroom. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, Chattanooga, TN: ACM, 2009.

[8] McNally, M. F., Walking the grid: robotics in CS 2. In *Proceedings of the 8th Austalian Conference on Computing Education - Volume 52* , Hobart, Australia:ACM, 2006.

[9] Summet, J., Kumar, D., O'Hara, K., Walker, D., Ni, L., Blank, D., and Balch, T. 2009. Personalizing CS1 with robots. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, Chattanooga, TN: ACM, 2009.

7. ACKNOWLEDGEMENTS

This work was made possible by an award from the Institute for Personal Robotics in Education.

INDUSTRIAL ROBOTIC GAME PLAYING: AN AI COURSE*

Sebastian van Delden
University of South Carolina Upstate
800 University Way
Spartanburg, SC 29303
864 503-5292
svandelden@uscupstate.edu

ABSTRACT

A unique approach to teaching a general purpose Artificial Intelligence (AI) course is presented in this paper. The course provides an active learning environment which strengthens traditional AI concepts through a semester long industrial robotic game playing project. It requires an industrial robotics laboratory that includes robotic arms which are available for students to use. Several foundational topics related to AI are implemented by the students on an actual robotic arm, including basic robotic coordinate systems and transformations, game playing algorithms, image segmentation, and feed forward neural networks.

INTRODUCTION

Most universities feature an Artificial Intelligence (AI) course that is available to junior/senior level computer science students who already have knowledge of basic data structures and algorithm design. The content of this course can vary substantially, depending on the background and interests of the instructor who teaches the course. For example, if an instructor's graduate work was based on Natural Language Processing, then the course will feature a large Language Processing component. If the instructor does not have an interest in robotics, then the course might not include any robotics-related material. It could also be argued that some topics like robotics and computer vision do not even belong in a general-purpose AI course and that they belong in a proper robotics or computer vision course. Certainly this argument is valid, however, sometimes hard to realize because of departmental limitations in offering a wide variety of AI related courses.

There have been many articles written on AI course design. An early paper by [2] describe an expert-system based AI course. [3] outlines an AI course that features the RoboCup soccer simulation system. [7] describes a course that combines algorithm analysis, AI, and technical writing. [1] uses an interactive visualization tool (CIspace) to help students learn AI concepts. [8] presents a machine learning approach to teaching AI. Similar to our work, [6] introduced a robotics laboratory based AI course. There have also been numerous proposals to incorporate basic AI topics into introductory computer science courses ([11]) or courses for non-majors ([5], [12]). Indeed, the more recent published works in this area focus on how modern technologies (hardware and software) can be used to help students grasp traditional AI concepts. The consensus is that there is still no standard syllabus for teaching an AI course.

The course described in this paper outlines a general purpose AI course which touches numerous AI related topics without diving too deep into each individual area, including:

- **Robotics**: Transformations and 3D Coordinate Systems
- **Traditional AI**: Game Playing Algorithms and Searching
- **Computer Vision**: Basic Image Segmentation and Thresholding
- **Neural Networks**: Simple Feed Forward Neural Networks

What makes our approach unique is that the course is centered around a tremendous resource at our institution: an industrial robotics laboratory. Our lab houses five Stäubli RX60 6-dof robotic arms, a Stäubli RS20 4-dof arm, and an Adept 550 4-dof arm. Each machine has been networked to a PC in the lab so that students can develop Java code (our core language) to program the machines. This lab easily allows a class of around twelve students to divide into groups of two and develop a semester-long project. Very few institutions have such a resource at their disposal, and so very few institutions will be able to immediately benefit from this paper. However, we hope that contributions like this will help facilitate more donations of industrial robotic equipment to educational institutions.

Figure 1. The robotic game-playing project setup.

THE COURSE PROJECT

The key component of our AI course is a semester long project that must be completed by the student groups. The goal is for the student groups to implement a 2-player board game where the human physically plays against a robotic arm, and all pieces are known at all times. Each of the student groups develops a different board game. The Fox and Geese, Breakthrough, Checkers, Chess, and Lines of Action are some of the games we implemented. Figure 1 shows the project setup. Notice that the end-effector consists of a USB camera and gripper. A 6-dof robotic arm is shown here, however, 4-dof machines can be used in these projects as well.

To complete such a project, each stage of development requires some knowledge of a specific AI related topic. We briefly outline each step here and in the below sections we provide more details on the minimum amount of information that needs to be covered on each topic. First, some basic industrial robotics material on how to operate and program the specific robotic arm must be covered. Each robotic platform has its own proprietary programming language. Stäublis, for example, are programmed with the high level languages (3rd Generation) V+ and VAL3. The robotics material should also include foundational information on 3D coordinate systems and transformations – mathematical models that are true for any robotics platform. The students should have a firm handle on the internal representation and manipulation of 3D locations and movements. Second, a game-playing algorithm like Mini-max with ?-? pruning must be covered. In this step, the students will need to implement the rules of the game as well as a traditional game playing algorithm. Third, a camera is integrated into the system so that the robot can physically see when its opponent has made a move and react to it. Up to this point, the game could be played via a simple graphical drag-drop interface, or keyboard input that indicates which move has been mode. Adding a computer vision component is an important step to making the game more interesting and realistic to play. Fourth and time permitting, the game playing algorithm of the system is scrubbed and a new Feed Forward Neural Network is implemented which learns how to play the corresponding game from scratch. The students get to see that no game rules are hard coded in the system and the robot learns how to play the desired game simply be interacting with the human opponent – who indicates whether the robot makes a right or wrong move.

ROBOTICS COMPONENT

In this and the following sections, the basic material that would be covered in each stage of the project is presented. The first thing that the students need to know is how to operate the robot arms, and the basic syntax of the robotic arm's programming language. Junior or senior level computer science should be able to quickly learn the basics of this new language.

Along with robotic programming comes some concepts of 3D coordinate systems, matrix algebra, and transformations that will need to be covered so that the student understand the mathematical models that drive the motions of the machine (see [4] for a more detailed presentation of the following material). Figure 2 depicts a robot work area in which several transformations have been defined which represent the typical 3D coordinate systems and 3D locations in the robot's work cell.

Movement of the robot's end-effector to pick up the bolt in Figure 2 (for example) is the result of moving its coordinate system {T} with respect to {B}and {S} so that it becomes aligned with {G}. A 4 x 4 matrix, called a transformation, is used to represent rotational and translational differences between coordinate systems. Locations in one system can be mapped to another system by multiplying the transformation with the location: $${}^{A}P = {}^{A}_{B}T\,{}^{B}P: \begin{bmatrix} {}^{A}P \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{A}_{B}R & {}^{A}P_{BORG} \\ 000 & 1 \end{bmatrix} \begin{bmatrix} {}^{B}P \\ 1 \end{bmatrix} \quad (1)$$

where ${}^{A}P$ is a point in coordinate system A, ${}^{B}P$ is a point in coordinate system B and ${}^{A}_{B}T$ is the transformation matrix that maps points in system B to system A. The 3x1 column vector ${}^{A}P_{BORG}$ is the translational difference from A's origin to B's origin, and ${}^{A}_{B}R$ is a 3 x 3 rotation matrix comprised of three orthonormal column vectors that map the orientation of each axis in A with respect to B.

$$({}_{\downarrow}B^{\uparrow}A)R = [(\cos\alpha\cos\beta\cos y - \sin\alpha\sin y\&- \cos\alpha\cos\beta\sin y - \sin\alpha\cos y\&\cos a\sin\beta@$$

$$\sin\alpha\cos\beta\cos(y + \cos\alpha\sin y)\&- \sin\alpha\cos\beta\sin y + \cos\alpha\cos y\&\sin\alpha\sin\beta@$$

$$- \sin\beta\cos y\&\sin\beta\sin y\&\ldots \quad (2)$$

Where α is the rotation around the moving Z in degrees, β is the rotation around the moving Y in degrees, and γ is a second rotation around the moving Z in degrees – i.e. Z-Y-Z Euler angle representation which is the internal representation used on Stäubli machines.

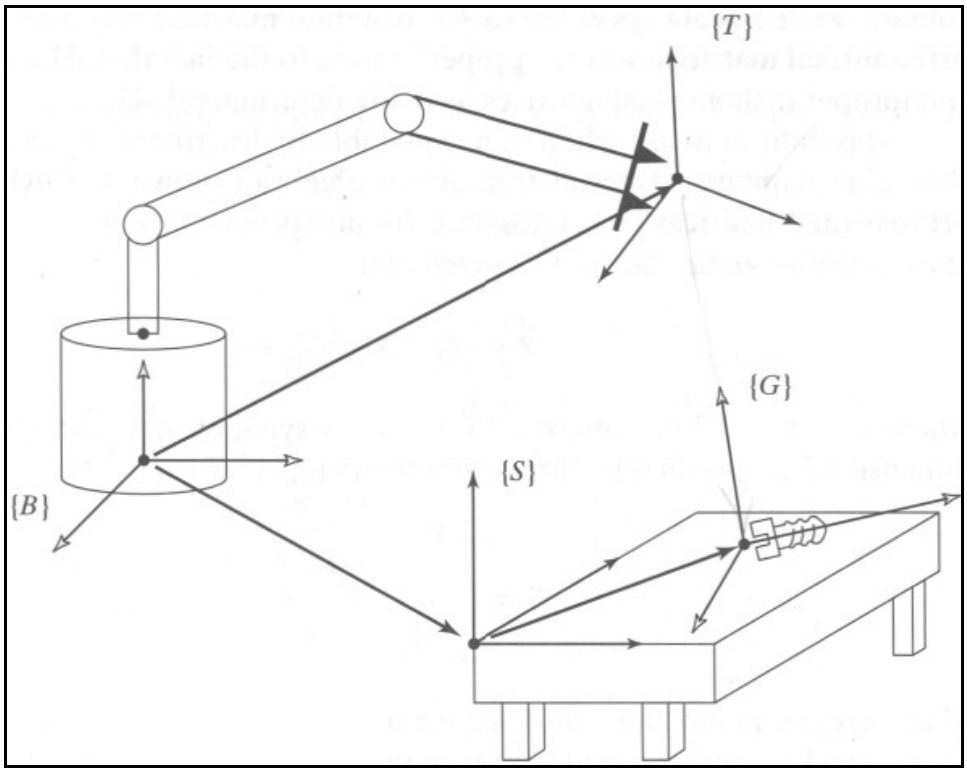


Figure 2. Typical transformations (3D coordinate systems and locations) defined in a robot work cell. This figure is very similar to a figure in [4].

With this material, the student should be able to implement the pick and place component of the semester long project – the basic picking up and moving of pieces to and from appropriate locations on the checker board.

AI COMPONENT

The next step is to implement the rules of the particular game. The robot needs to be able to make a valid move and also verify that its human opponent has made a valid move. In order to interact with the robot a simple drag and drop interface could be designed by the instructor ahead of time. This is used by the human to let the robot know which move they have made. [9] provides a good text that covers the following material in more detail.

Initially, the AI algorithm could be designed to just make a random valid move. However, this does not make for an interesting robot opponent. Since the projects are based on two player board games where all pieces are known, the Mini-max search algorithm would be an excellent choice to integrate into the system at this point.

The Mini-max search can be described as follows. From the root node of a Mini-max search tree, the "best" child (move) should be made that maximizes the robot's chance of winning. Once the robot has made a move, the algorithm assumes that the (human) opponent is going to try to make the "best" move to minimize the robot's chance of winning. And so on, and so on. The Mini-max search tree is shown in Figure 3 where square nodes are maximizing and circular nodes are minimizing.

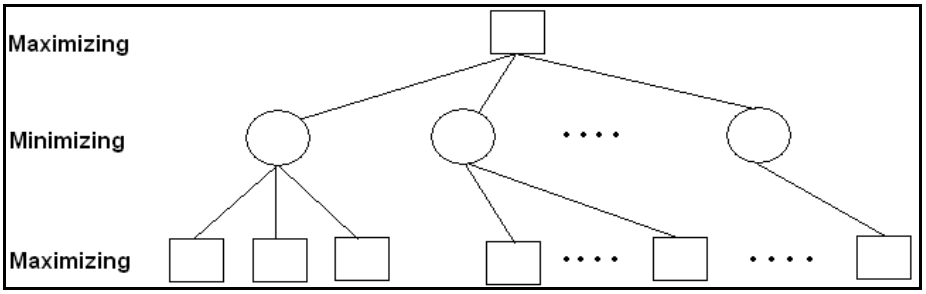


Figure 3. Mini-max search tree concept. Nodes represent board states and arcs represent a single player move.

A heuristic function must be implemented which assignments a score to each node which indicates how "good" that board state is for the robot's chance of winning. A heuristic function is an estimate of how far the robot is from winning the game. This function should be admissible – it does not over estimate the distance to the goal. The heuristic function will be different for each board game and must be carefully thought out by the student. To reduce the exponential $O(B^d)$ search space associated with the Mini-max algorithm (where B is branching factor and d is tree depth), the α-β pruning mechanism can also be implemented. The α-β pruning mechanism removes certain sub-trees from the search by tracking the current return value in the parent and comparing it to the current return value of the child. Sub-trees are cut as soon as the child returns a value that can never be returned by the parent, as shown in Figure 4.

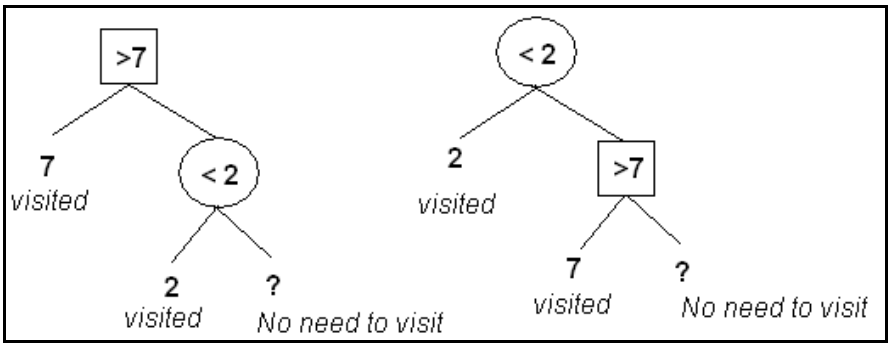


Figure 4. The two versions of α-β pruning. The values 2 and 7 are the scores that indicate how "good" the state is for the robot. In both of these cases, the node value of "?" does not need to be known or checked because it will never be returned by the top level node.

COMPUTER VISION COMPONENT

Now it is time to make it possible for the human to directly interact with the machine. See [10] for a good general purpose computer vision text that explains the following topics in more detail. Up to this point, the human had to somehow let the robot know what move had been made. As seen back in Figure 1, a camera is also mounted on the robot's end-effector along with the gripper and is pointed downward to observe the board. This starting location must be remembered and the entire board must be visible. The project is also simplified if the board is manually aligned to the row/column dimensions of the input image, as shown in Figure 5. The starting location and board cannot move. Also, game pieces must consist of two sharply contrasting colors – in our projects we used red and blue circular game pieces on a black and white checker board.

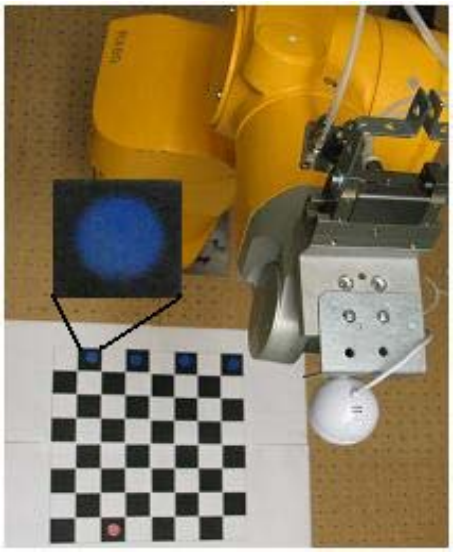


Figure 5. Top-down view of the board from the camera's perspective. The bounding box of a board square that has a piece on it is magnified.

The bounding box of the board (image coordinates of the upper-left and lower-right corners of the board) must be manually identified by the student. The bounding boxes for each board square can then be calculated, or manually hardcoded. This bounding box for the board is important because it takes all the surrounding noise out of the image. Notice that the only pixels in the board's bounding box are white and black, and the two colors of the game pieces.

An initial grayscale image, I0, of the board with no pieces is stored in memory. The human is then asked to place the pieces on the board in their starting positions and a new grayscale image, I, is captured. The locations of the pieces on the board can be determined using the sum of square differences of the pixel values. For each game square bounding box {(k,p), (n,m)}:

$$\text{if } \left[\sum_{i=k}^{n-1} \sum_{j=p}^{m-1} \sqrt{(I[i][j] - I_0[i][j])^2} \right] > \tau \text{ then a piece is present} \quad (3)$$

where τ is a user-defined threshold. The difference between the color of the two types of pieces needs to also be distinguished. Since we used red and blue pieces, this was easy to implement. Once it has been determined that a piece is present by (3), the red and blue quantities of the RGB color components are simply compared:

$$\text{if } \left(\sum_{i=k}^{n-1} \sum_{j=p}^{m-1} I_{color}[red][i][j] \right) > \left(\sum_{i=k}^{n-1} \sum_{j=p}^{m-1} I_{color}[blue][i][j] \right) \text{ then piece is red} \quad (4)$$

Finally, the robot needs to know when a move has been made so that it can react. The camera should take pictures periodically (every half second or so) and see if the board state has changed. A global comparison of all the board pixels to the original board in I0 should be performed each time. If there is dramatic global difference in the overall board, then the user's hand has been imaged while the human opponent was making his/her move. This global comparison is again accomplished with a user-defined threshold and sum of square differences.

At this point the game could be deemed complete and would be enjoyable to play. However, an additional step could be added – a neural network which learns how to play the game.

NEURAL NETWORK COMPONENT

In this stage, a simple feed forward neural network ([9]) is implemented which learns the rules of the game. The rules which were hard-coded earlier should be discarded for this section. The human "teaches" the robot how to play by observing random moves by the robot and indicating whether the robot has made a correct or incorrect move. This is an interesting stage because each student implements the same neural network and during the interactive training process, the neural network will learn the specific rules for the particular game. Figure 6 depicts the classic single layer feed forward network. The typical sigmoid activation function for the units could be used $g(x) = 1/(1 + e^{-x})$ as well as the standard weight update rule is:

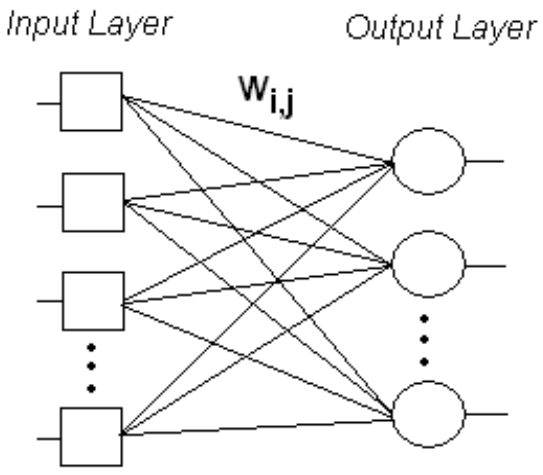


Figure 6. A basic single-layer feed forward neural network

$$W_{i,j} \leftarrow W_i j + \alpha * I_i * Err_j \quad (5)$$

where $Err_j = Truth_j - Observed_j$ and α is the learning rate which should be set relatively high so that the network quickly learns the rules. In this project, the input layer could consist of 256 Boolean inputs – 64 that indicate where the robot's pieces are initially (assuming the board is 8 x 8); 64 that indicate where the robot's pieces end up after the move; 64 for the initial positions of the human's pieces; and finally 64 for the final positions of the human's pieces.

In the simplest form, the output layer could be a single Boolean unit which indicates whether the move was legal or not. More output units could be added to indicate whether the move was a 'good' legal move. Of course, here we are assuming the linearly separable function captured by the weights will be enough to learn the rules. A hidden layer and the error-back propagation algorithm might have to be implemented for some games.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Amershi, S., Arksey, N., Carenini, G., Conati, C., Mackworth, A., Maclaren, H., and Poole, D. 2005. Designing CIspace: pedagogy and usability in a learning environment for AI. In *Proceedings of the 10th Annual SIGCSE Conference*. Caparica, Portugal, June 2005. 178-182.

[2] Burghart, J. 1988. *Design of a course in artificial intelligence and expert systems*. In Proceedings on the Frontiers in Education Conference. Santa Barbara, CA. October 1988. 279-284.

[3] Coradeschi, S., and Malec, J.. 1999. How to Make a Challenging AI Course Enjoyable Using the RoboCup Soccer Simulation System. *Lecture Notes in Computer Science*. Volume 1604/2008. Springer. 120-124.

[4] Craig, J. 2004. Introduction to Robotics: Mechanics and Control. Third Edition. *Prentice Hall*.

[5] Danyluk, D. 2004. Using Robotics to Motivate Learning in an AI Course Aimed at Non-Majors. In *Proceedings of the AAAI 2004 Spring Symposium on Accessible Hands-on Artificial Itelligence and Robotics Education*.

[6] Kumar, D. and Meeden, L. 1998. A robot laboratory for teaching artificial intelligence. In *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*. Atlanta, Georgia, United States, February 1998. 341-344.

[7] Martin, J. 1994. Algorithms: An Integrated Algorithm Analysis, Writing and Artificial Intelligence Course. In *Proceedings of the AAAI 2004 Fall Symposium*.

[8] Russell, I., Markov, Z., and Neller, T. 2006. Teaching AI through machine learning projects. In *Proceedings of the 11th Annual SIGCSE Conference on*

innovation and Technology in Computer Science Education. Bologna, Italy, June 2006. 323-323.

[9] Russell, S., and Norvig, P. 2002. Artificial Intelligence: A Modern Approach. Second Edition. *Prentice Hall*.

[10] Shapiro, L., and Stockman, G. Computer Vision. *Prentice Hall*. 2001. ISBN: 0130307963

[11] van Delden, S., and Zhong, W. 2008. Effective Integration of Autonomous Robots into an Introductory Computer Science Course: A Case Study. *Journal of Computing Sciences in Colleges*. Select papers from the Sixth Meeting of the Consortium for Computing Science in Colleges. Volume 23(4),10-19, April 2008.

[12] Wright, A., Ferrer, G., Wright, A. 2003. A Liberal Arts Approach to Teaching Robotics. In *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition*. Nashville, Tennessee.

INTERNATIONAL COMPUTING ISSUES

AS A FRESHMAN SEMINAR*

Christopher A. Healy
Department of Computer Science
Furman University
864-294-2097
chris.healy@furman.edu

ABSTRACT

This paper discusses the author's experience with a unique freshman seminar course in global computing issues, taught recently for the first time. This non-technical course is intended for a general audience, and is part of our department's aim to reach out to students who might otherwise not take a CS course. Thus, the seminar serves its role as a service course for the general college student. While it may not attract many new majors to computer science, it can expose liberal arts students to computing issues without getting bogged down with software, programming or mathematical challenges.

1. INTRODUCTION

Last year my institution revised its general education requirements to include a new category: entering students must now take two freshman seminars. These seminars are intended to be offered by any department on campus, not just those in the humanities. The computer science department aggressively embraced this program, offering five different seminar sections in 2008-09, mostly focusing on the history and applications of CS. The author felt the need to offer a seminar that dealt with international issues, such as globalism, access to IT around the world, cultural differences, and environmental sustainability, since these issues arise in upper level courses in CS, economics, and elsewhere.

Recently, several educators have noted the need for more international content in the CS curriculum. For example, Ferguson [5] examined the phenomenon of outsourcing as an impetus for departments to redesign existing courses and design new ones to make

future American programmers more competitive globally. Kurkovsky [7] introduced a new course in analyzing software dealing with sustainable development. Among other things, students employed simulation models to measure air pollution in several industrialized countries. In the area of non-programming courses: Hare [6] describes the teaching of a seminar-type course in which students read high quality articles from international news sources to examine current events and ethical issues in IT.

To the author's knowledge, this new seminar is unique: only an MBA course at American University [1] had similar content. By contrast, this seminar is intended to be non-technical, accessible to freshmen. This seminar was entitled "Global and Green Computing." The reason why the title was not simply "global computing" was that at the time the course was conceived, there was some doubt that one could fill an entire semester just covering issues of globalism. However, as reading material was being collected it became quickly apparent that there was no trouble finding enough global issues. Thus, in the end the "green" aspect of the course was condensed to 1-2 weeks. Course topics include such issues as: the accessibility of IT by ordinary citizens, to what degree the national government makes IT a priority, the cell phone and Internet infrastructure and cost issues, the quality of CS/IT education and research, cultural differences, the strength of the local IT industry and global competitiveness, as well as obstacles and strategies for doing IT in a developing economy.

Although the seminar is not intended to count towards a computer science major, the subject matter of the course does include some topics listed in the ACM/IEEE 2001 Curriculum body of knowledge [2]. In the Social and Professional Issues unit, the topics are SP2 (Social context of computing), SP7 (Privacy and civil liberties), and SP9 (Economic issues in computing).

2. APPROACH IN PREPARING THE SEMINAR

Rather than using a textbook, the reading list consisted of a large number of high-quality articles. In preparing the reading material, the author collected about 75 articles totaling 500 pages; however not all were actually assigned. Most of the articles came from the flagship journals in computing: *Computer* and *Communications of the ACM*. However, care had to be taken to choose articles that were accessible to a non-technical audience. The aim was to give students roughly 15 pages of reading for each class. Most articles were short in length (3-7 pages) but dense in material. Currency of the material was also a key consideration. Many well-written articles from the 1990s and early 2000s were noticeably obsolete and could not be used. A 1997 article on the Irish software industry is of less value than one from 2008, unless one wishes to study long-term trends in depth. The selected readings also showed some variety of article type: some were editorial pieces, one was a government document, and one was a conference research paper.

A basic goal for this course was for students to become more critical readers. On the first day of class, the instructor gave a 45-minute overview on how to analyze arguments. To start off the readings for the semester, students were assigned the first 5 chapters of the *Global Information Technology Report* [4]. This source provided enough discussion material for two weeks. Thereafter, the class launched into a routine of reading 2-3 short articles per class, centered on the same geographic region or issue.

The day before each seminar meeting, suggested questions for discussion were posted to the class Web site and also e-mailed to the students. One question basically represented about 10 minutes of class discussion. Besides mere class preparation and topical focus, these questions were intended to show students the kinds of issues they need to consider in general when reading articles: identifying the intended audience, the author's purpose, writing down terms not initially understood, points that were left out of the article, how the article would have been written differently from another country's point of view, etc.

As in the other freshman seminars offered by our department, writing is an important element of the course. In the author's seminar, the students were assigned two 2,000-word term papers during the semester, each worth one-third of the total course grade. Although the course was not about teaching the mechanics of writing, the author felt it was important to have students reinforce their skills in researching topics in the library and online, and then expressing their views orally as well as in writing. Having a CS department course that features writing as an essential element may help to foster a culture of writing throughout the rest of the curriculum. Dansdill et al. [3] noted that although a majority of CS faculty feel writing is very important for their students, they found that very few CS programs stress writing anywhere in the curriculum.

The research papers also gave students a chance to study a country or issue in more depth than class time allowed, or to study a country that was not covered at all in the readings. In a nutshell, the themes of these two papers were: "one country, many issues" and "one issue, many countries". The first essay assignment asked students to select a country, and do an in-depth study of its technological landscape. Conversely, for the second paper, students needed to select an issue, such as Internet free speech, and contrast it in several countries. The papers were graded holistically, primarily based on the quality of their content and organization.

Class participation determined the remaining one-third of the students' grade. Because of this large weight, the author desired some objective means of quantifying participation. If the enrollment is very small and class duration long, one method would be to assign a grade to each student's participation for the meeting. Since the class had 14 students and met for 75 minutes, it was more feasible to count the number of significant contributions made by each student during the seminar. Depending on the depth of a student's contribution, it was valued as a half or full point. Thus, over the period of the semester, a quantitative basis for determining the class participation was used.

3. RETROSPECT

In assigning reading material, one minor problem encountered was that several of the articles echoed many of the same points. For example, poverty-related issues in India are often duplicated in Africa and elsewhere. As a result, some articles were dropped from the list. Also, it was much easier to find articles dealing with economic issues than environmental ones in CS-oriented publications. In sequencing the reading list, it turned out to be more convenient to arrange the articles by country rather than by topic. Specifically, the countries/regions covered during class included: China, India, Africa, the Middle East, Brazil, Australia, Canada, Singapore, Malaysia, Ireland, Russia and

Finland. The class spent 2-3 meetings on larger and influential countries such as China and India. The last one-third of the course was devoted to a survey of topics (e.g. multilingual Web content) rather than a geographical focus. In all, students were asked to read 56 documents during the term.

As a rule, class discussion was especially active. During a typical class the instructor would note about 40 significant contributions from the students. The best students were often able to stimulate discussion, rather than wait for the instructor to ask the next question. Most students chose to bring a laptop to class. They found it useful to check data online during class such as the *CIA World Factbook* or to verify an article's references. Thus, students were able to critique articles. However, there was a large variance in the participation scores. The number of contributions per student per day generally ranged from 1 to 6. In practice the participation numbers were curved to yield a meaningful grade, with the intent that, for example, a "B" in class participation should connote approximately the same quality of work as a "B" on an essay.

One pleasurable aspect to teaching the course was the freedom to cover any topic under the broad umbrella of the course title. This course is not a pre-requisite to any other. There are no exams or programs to grade. Thus, there is no pressure to cover certain material, and the seminar can proceed organically at its own pace. Perhaps the most significant practical benefit of teaching a seminar such as this is the immediate increase in the department's enrollment figures. Our department saw a 30% increase in total enrollment over the previous year thanks in part to the seminars offered. Unlike most other courses in the department, they were nearly filled to capacity. And also unlike most classes in the department, the enrollment had a rather close gender distribution. The class of 14 students included 8 men and 6 women, 13 white and 1 Asian, while two of the students were computer science majors (most were still undecided).

Often, introductory level CS courses have a bimodal grade distribution due to the fact that some students already know the material from high school. In this seminar, the grade distribution was unimodal with an average of 2.41 and few extreme grades. It is unlikely that a student will come to college already familiar with concepts dealing with IT readiness in various countries. The poorer grades were generally due to students missing class, contributing little to class discussion, or writing relatively weak essays – in other words, not due to a student failing to comprehend the material. The only struggle that some students experienced was finding research material for their papers, since the assigned topics were unlike those in practically any other college course (e.g. finding IT statistics for Chile).

#	Statement	# Agree	# Disagree	# Neutral
1	I had little or no trouble keeping up with the reading load for this class.	12	1	1
2	I generally found the assigned reading straightforward to understand.	9	1	4
3	The articles assigned were appropriate for the topics of the course	14	0	0
4	The number of countries studied was sufficient for a course on international issues.	13	0	1
5	The number of different topics discussed was	11	2	1

	sufficient.			
6	I learned a lot from reading the articles.	9	3	2
7	I learned a lot during our class discussions.	8	2	4
8	I learned a lot when doing the research papers.	10	0	4
9	I have had many opportunities to express my views during class discussion.	11	1	2
10	The number and length of term papers was appropriate for this class.	13	0	1
11	The list of study questions e-mailed to the class the day before the seminar helped me to prepare for the day's discussion.	11	2	1
12	A 50-minute instead of 75-minute class period would have been better for encapsulating class discussion.	8	3	3
13	This course has opened my eyes on the role of computing in society.	7	1	6
14	As a result of this class, I am more likely to take another (technical or non-technical) course offered by the computer science department.	2	6	6
15	I have learned a lot about foreign cultures as a result of this class.	8	3	3

Table 1: Results of survey given to students.

4. STUDENT OPINION

Table 1 shows the results of a survey given to the class near the end of the semester. The purpose of the survey was to determine if students found the readings accessible, and to what degree various educational goals were realized. Some questions were also included to help plan for future seminars. Most of the responses were overwhelmingly positive. The only disappointing question was the one asking if the students were inspired to take another CS course someday. Most said they were not. However, it should be noted that the university's registration system randomly assigned many students into seminars. Students were taking this seminar because it fulfilled a graduation requirement and it was not necessarily one of their top choices. For the most part, the students in the course were a cross-section of the freshman class.

5. CONCLUSION

There is not a lot of attention paid to the CS curriculum outside the technical confines of the CS major. As a result, most students bypass taking a CS course. Offering freshman seminars can bring more students through the front door. These classes can reinforce the message that CS is not just about programming, and that IT issues impact people all over the world. The instructor's experience with a new seminar on global computing issues is part of this program. By carefully selecting high-quality accessible articles, a general student can be exposed to many issues in which IT intersects with the larger society. Besides the IT-related content, such a seminar fits well in a general education framework by teaching students how to ask the right questions when they read, and how to express themselves verbally. And it shows students that CS and IT concepts

have a human face.

REFERENCES

[1] Carmel, E., Nations, policy and information technology, http://www1.american.edu/academic.depts/ksb/mogit/initeb/ , retrieved April 17, 2009.

[2] Computing Curricula 2001 Task Force, Computing curricula 2001, http://www.computer.org/portal/cms_docs_ieeecs/ieeecs/education/cc2001/cc2001.pdf , retrieved April 17, 2009.

[3] Dansdill, T., Hoffman, M., Herscovici, D., Exposing gaps, exploring legacies: paradoxes of writing use in computing education, *Journal of Computing Sciences in Colleges*, *23*, (5), 24-33, 2008.

[4] Dutta, S., Lanvin, B., Paua, F., *The Global Information Technology Report 2003-2004*, New York: Oxford University Press, 2004.

[5] Ferguson, E., Impact of offshore outsourcing on CS/IS curricula, *Journal of Computing Sciences in Colleges*, *19*, (4), 68-77, 2004.

[6] Hare, B., Implementing a writing-intensive C.S./I.T. ethics course, *Journal of Computing Sciences in Colleges*, *24*, (1), 76-82, 2008.

[7] Kurkovsky, A., Educational aspects of sustainable development analysis: computational models and software, *Journal of Computing Sciences in Colleges*, *21*, (4), 24-31, 2006.

USING VIDEO TO EXPLORE PROGRAMMING THINKING AMONG UNDERGRADUATE STUDENTS*

Carol Wellington
Department of Computer Science
Shippensburg University
Shippensburg, PA 17257
cawell@ship.edu

Rebecca Ward
Department of Teacher Education
Shippensburg University
Shippensburg, PA 17257
rjward@ship.edu

ABSTRACT

Phenomenography is a qualitative method that primarily has employed interviews of students to explore variations in how they experience some phenomenon. This study is an experiment in phenomenography in which the basic data (the interview) is replaced by video of students working on a computer science project with the goal of exploring variations in how students approach programming activities and laying the groundwork for more extensive study. While this is a preliminary study, early analysis shows that the addition of videos enriches the conclusions that can be made and further exploration of this technique is warranted.

1. INTRODUCTION

Phenomenography is an empirical, qualitative research method designed to determine differences in how people experience specific phenomena (Marton & Booth, 1997). In other words, phenomenography is a way of looking for qualitatively different conceptualizations of an experience in a group of people. The basic structure of a phenomenographical study is formal interviews of subjects that are transcribed. Those interviews are then analyzed to look for categories in which the participants vary and exploring those variations. In computer science, this method has been used to explore student understanding of programming thinking (Eckerdal & Berglund, 2005) and how students learn to program (Govender & Grayson, 2008). Lister (2003) makes a strong

argument for using phenomenography as a way to explore to what extent students understand computer science topics. In the current study, we use phenomenography as a base from which to explore programming thinking as a developmental construct. We are particularly interested in the qualitative changes that occur in programming thinking as computer science students progress through their undergraduate programs.

A key component of phenomenography research is that data is collected from the perspective of the student rather than the researcher and involves determining varying ways that students interact with phenomena. Previous studies have used post-activity interviews (Spohrer and Soloway, 1986) or observing students working (Perkins, Hancock, Hobbs, Martin, and Simmons, 1986). While interviews are often used to collect this data after the fact, we employed audio and video technology in order to capture "real time" thinking and programming with less risk that the observation would affect the students' behaviors. Students were asked to "think out loud" during a programming assignment while their voices and computer screens were recorded. Our sample size was intentionally small in order to determine the validity of the research method and to begin a qualitative exploration of themes to be used in coding for future research.

2. EXPERIMENT DESIGN

We are interested in how video phenomenography might be used to learn about how students' problem solving skills mature throughout their undergraduate education. A better understanding of the way that problem solving abilities mature could lead to the development of curricular materials that assist that maturational process.

For this preliminary study, we wanted to study a broad cross-section of computer science majors, so we put out a call to all of our majors for volunteers. In that call, we explained the purpose of the study and asked students for one hour in which their efforts on a programming activity would be recorded. As a result of that call, we recorded students with these characteristics:

Student	**Description**	**# of CS courses completed**
Two Freshmen	A pair of female freshman working together	1
Freshman 1	A male freshman with significant pre-college experience	1
Freshman 2	A female freshman	1
Sophomore 1	A male sophomore	2
Junior 1	A male junior	9
Senior 1	A male senior ready to graduate	11

The laboratory activity for this study was a Computer Science I activity that is usually assigned about ¾ of the way through the semester. The goal of the activity is translating zip codes into bar codes and it requires conversion of ints to Strings, tearing an int into its component digits, arrays, loops, and conditionals. The instructions for the lab include information about bar codes and specific instructions that lead the students through test-driven development of the required class. For example, the lab is specific about which border cases need to be tested, but it does not give instruction on how to write the code to make those tests pass. It is important to note that all of our students

have been exposed to unit testing with JUnit, but only our freshmen and sophomores have been explicitly instructed in test-driven development.

The lab was given to the students before their recording time and they were instructed to read it before they were recorded (not all of the students did that). At their recording time, the students used a Mac and Eclipse to work on the lab and were instructed to describe what they were doing and why they were doing it. Snapz recorded the contents of their screen and their voices as video.

VIDEO ANALYSIS STRATEGY

Each of the videos was transcribed into a spreadsheet with 15 seconds for each row. In addition to what the student said, whether the student was doing each of these activities was gathered for each interval:

- reading?
- re-reading?
- following TDD?
- writing tests?
- writing code?
- running tests?
- adding debug printouts?
- using the debugger?
- tests were red?

A computer science faculty member also watched the videos and added comments about what the student appeared to be doing, any misconceptions the student seemed to have, and any other relevant comments.

4. PRELIMINARY RESULTS

Our preliminary analysis has shown a number of areas in which the strategies used by students differ with their experience.

4.1 Irrelevant changes

The lower division students made some changes to their solutions that seem to imply some basic misunderstandings of how some language constructs work.

Both "2 Freshmen" and "Sophomore 1" added extraneous else clauses when their tests failed. In both cases, they made changes like this:

Original Code Structure	Modified Code Structure

`if (condition)`
`  return val1;`
`return val2l`
	`if (condition)`
	`  return val1;`
	`else`
	`  return val2`

The addition of the "else" has no effect on the behavior of the code, but, in both cases, the students re-ran their tests hoping they had fixed the problem. This appears to show a lack of understanding of how "return" affects the flow of execution in the method.

"2 Freshmen" also tried re-arranging a sequence of nested conditionals.:

Original Code	Modification

`if (zipCode < 100000)`
`  return "0" + zipCode;`
`else if (zipCode < 10000)`
`  return "00" + zipCode;`
`else if (zipCode < 1000)`
`  return "000" + zipCode;`
`else if (zipCode < 100)`
`  return "0000" + zipCode;`
`else if (zipCode < 10)`
`  return "00000";`
	`if (zipCode < 10)`
	`  return "00000";`
	`else if (zipCode < 100)`
	`  return "0000" + zipCode;`
	`else if (zipCode < 1000)`
	`  return "000" + zipCode;`
	`else if (zipCode < 10000)`
	`  return "00" + zipCode;`
	`else if (zipCode < 10000)`
	`  return "0" + zipCode;`

Since these conditions were mutually exclusive, their order had no effect on the external behavior of their method. The students made this change and, even when it did not change the results of the tests, they did not comment on the fact that their effort made no difference. In fact, it took another three and a half minutes for them to see the off-by-one errors in their strings. Upper division students did not make these types of mistakes.

4.2 Search for Help

When students were unsure about how to proceed, they looked for help in a variety of places:

"2 Freshmen" and "Freshman 2" looked back at the lab for clues. Since the lab contained little information on syntax (except for an explanation of the modulus operator), this probably means that their troubles lay in formulating a solution more than in finding the specific syntax details. In fact, when "2 Freshmen" were trying to break the zip code into digits, they spent twelve minutes writing a variety of attempts at the solution using the syntax correctly, but were unable to put the pieces together into a solution that worked. In those 12 minutes, they went back to the lab for clues five times. In the middle of this time, one of these students gave another strategy for getting help when she said, "This is when I go to somebody else who knows more."

All of the students, to one degree or another, used Eclipse's ability to recall methods when they were stuck. For example, "Junior 1" and "Senior 1" both had difficulty remembering how to convert an int to a String. Initially, they both typed the name of the variable followed by "." and waited for clues from Eclipse. Only after Eclipse gave no possibilities did they remember that ints are primitives, so there isn't an existing method to make the conversion.

Both "Junior 1" and "Senior 1" used Google to find strategies. In fact, "Senior 1" was very quick to go to Google; when Eclipse gave no clues, he immediately brought up a browser and did a very quick search to find a solution. He went so far as to say, "Where's my Google machine? That's the best problem-solving skill. If I had to say one thing saves me every time . . ."

4.3 Off-By-One Errors

There is some evidence that teaching students to be aware of specific types of errors may help them prevent and/or repair them. In this particular lab, off-by-one errors were a common point of discussion.

"2 Freshmen" expressed an awareness of off-by-one errors by mentioning them when they were writing a loop, but they didn't draw any conclusions about whether or not they currently had such an error. Later, they made an off-by-one error where a loop goes one more time than necessary, but it had no effect on the external behavior of their method and they did not catch it.

"Sophomore 1" was clearly aware of off-by-one errors and tried almost random things to fix them. The lab required the addition of leading zeros when outputting the zip code when it is stored as an integer. His initial attempt was to check if the zip code was less than five, confusing the length of the integer (in digits) with its value. When that didn't work, he ran through this sequence of conditions: < 6 (runs the tests), <=4 (runs the test), <=5, <5 (where he runs the tests commenting that he knows that's the first condition he tries), (at this point he makes it print out the zip code which is 1234) <=4. He was so focused on the off-by-one error that it took him almost four minutes to recognize that he was looking at the integer's value instead of the number of digits it contained. He made a similar sequence of changes to a loop variable later in the lab. In both cases, the error he was making was not an off-by-one error.

"Junior 1" also wrestled with off-by-one errors. At one point in breaking the zip code into digits, he got an array out of bounds exception. He talked about the potential of having an off-by-one error: "It's not just an off by one error because at the end this will still go up to four and I'll still have an out of bounds exception." He was aware of the possibility of an off-by-one error, but could reason about why that isn't his problem without writing the code.

4.4 Code Scribbling

The most interesting thing we've seen in these videos is what we call "code scribbling." This is similar to the concept of tinkering, during which students try several small code changes in the hopes that they will succeed (Perkins, et al., 1986). Code scribbling differs from this idea of tinkering in that students do not necessarily run each program. Rather, they repeatedly type and delete before actually running the program. "Sophomore 1" is the best example of this technique. As he was trying to code his solution, he typed out a series of attempts. Often, he ran into a syntactical reason why his solution wouldn't work and then backed up and started again. However, sometimes he reached a solution that was syntactically correct, looked at what he had, said, "That won't work" and backspaced to delete that solution and try again. It appears that he has to see his incorrect code before he can predict that its behavior is incorrect.

Given our observations of "Sophomore 1," we looked for similar behavior in the other students. "2 Freshmen" also progressed through a series of solutions, but they were not as adept at predicting that the code would fail. They had to see the tests fail before realizing that the code wouldn't work.

"Junior 1" also progressed through a series of solutions. However, he described the solution (without writing any code) and predicted that the solution would work without needing to see the code written out.

These observations provide evidence of a progression in the students' ability to imagine a possible solution and evaluate its efficacy:

1. Students have difficulty predicting how a piece of code will work and depend on running the code to see what it does.
2. After they write a piece of code, they can predict that it won't work, but they have to finish writing it first. In other words, they can only imagine its behavior after they have written it.
3. Students can imagine a solution and detect that its behavior will be incorrect without writing any of the code.

5. CONCLUSIONS/FUTURE WORK

This work is a preliminary study from which few hard conclusions can be drawn. The goal was to explore the use of video phenomenography as a tool for understanding how students' problem solving skills mature. We have seen variations in problem solving strategies that appear to show a cognitive difference between students with different levels of experience. In particular, their ability to imagine the behavior of code seems to improve with experience. However, more work is required to validate these conclusions.

The videos contain information that we have not yet analyzed. For example, we have data about the percentage of time that their tests were red, whether they were following test-driven development, and their use of the debugger and debug printouts. However, we have not fully analyzed that data. We will use that data and further analysis of the videos to explore these questions:

1. How close to the actual defect do they place the breakpoint/debug printout?
2. To what extent do they rely on the instructions in the lab as opposed to going ahead based on the description of the problem?
3. Do they recognize that a strategy for solving the problem is overly complex and back up to look for a new strategy?

The early results of this study validate the potential of video in a phenomenography based study of programming thinking and, therefore, warrant a more complete, well-structured experiment. In that experiment, we would like to make two improvements: enhance the video component of the study and modify the way the students are selected and studied.

In the current videos, there are points where it is difficult to ascertain what the students are doing because they are quiet and not typing. In future video phenomenography experiments, we would like to add a camera pointed toward the student to be able to better assess what they are doing. This would also capture non-verbal communication that may signal early progress or frustration.

The current experiment has one fundamental weakness: it is assuming that problem-solving skills have matured as students progressed through our curriculum. However, we have a significant retention problem; there is a chance that the students who persist into our upper division courses enter the program with better problem-solving skills. Therefore, we would like to run a more disciplined experiment focusing on how problem-solving skills mature in the first year of collegiate programming. In that experiment, a set of freshman computer science majors would be randomly selected. Each would be filmed for one hour each week through our Computer Science I and Computer Science II courses. The goal of that experiment would be to look for trends in how their problem-solving strategies evolve through that critical year.

REFERENCES

Eckerdal, A., & Bergulund, A. (2005). What does it take to learn 'programming thinking'? *ICER'05*, October 1-2, Seattle, Washington.

Govender, I. , & Grayson, D.J. (2008). Pre-service and in-service teachers' experiences of learning to program in an object-oriented language. *Computers & Education, 51*(2), 874-885.

Marton, F., & Booth, S. (1997). *Learning and Awareness*. Lawrence Erlbaum Associates, Mahwah, NJ.

Perkins, D.N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1986). Conditions of learning in novice programmers. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, pages 213 – 229. Norwood, NJ: Ablex Publishing Company.

Spohrer, J. D. and Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, pages 213 – 229. Norwood, NJ: Ablex Publishing Company.

FACTORS IMPACTING STUDENT PERCEPTIONS OF COMPUTING AND CIS MAJORS*

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

David P. Kitlan
Pennsylvania State University
777 West Harrisburg Pike
Middletown, PA 17057
717-948-6639
dpk104@psu.edu

ABSTRACT

Students entering university arrive with predefined perceptions about computing, about CIS majors, and about the role of technology in academia and society. Understanding the sociological and experiential factors which influence these perceptions can help CIS faculty devise programs and content more likely to be successful in attracting majors. This paper will discuss a multi-campus study intended to elicit the factors which impact students' perceptions of computing and CIS majors. The results suggest that differences in perceptions persist between genders, and that parental and K-12 efforts to stress computing may be having unintended effects.

INTRODUCTION

The past decade has shown that the ubiquity of technology does not necessarily equate to large enrollments in Computer and Information Science (CIS) majors. This bleak outlook for CIS enrollments may be changing, however. After years of decline, recent evidence suggests that CIS enrollments may be rebounding [12]. This rebound coincides with several efforts to adapt CIS courses around more integrative or contextual themes (e.g. [2], [11]). These initiatives suggest that CIS must be made "relevant" to students in order to stimulate interest in these majors.

Students entering colleges and universities arrive with predefined perceptions about computing, about CIS majors, and about the role of technology in academia and society. These perceptions affect what students see as "relevant" when it comes to computing curricula. Understanding the sociological and experiential factors which influence these perceptions can help CIS faculty devise programs, initiatives, and course content which are more likely to be successful in attracting students into CIS majors. This paper will discuss a multi-campus study to identify the sociological and experiential factors which impact students' perceptions of computing and CIS majors.

BACKGROUND AND STUDY DESIGN

There has been a significant body of literature regarding the acquisition of computing skills and the formation of computing perceptions. The existing literature appears to collapse around three general influences: gender, family and peers, and prior academic (K-12) experience. Existing studies which investigate the differences in computing skills and perceptions between the genders have found that, in general, women are less confident in their skill level ([1], [3], [8]) and often employ computing technology for different tasks than males [4]. The impact of gender role socialization on perceptions is also well-documented (e.g. [7], [10]). This socialization – which occurs as societal norms and gender stereotypes are internalized – is a process aided by parent expectations, peer pressures, media images, and the K-12 experience. The presence of subtle forms of sexism in K-12 curricula and the impact of a lack of social support from peers has also been noted ([8], [10]). The result of these experiential and sociological influences – family, peers, and K-12 education – is that perceptions of computing and CIS careers are well-formed by the time students arrive at university.

In order to more closely examine the factors which influence student perceptions of computing and CIS majors, a combined quantitative/qualitative study was constructed. The study was constructed around the conceptual model derived from one of the authors' prior work (see Figure 1). This conceptual model depicts the linkage between student perceptions of computing and CIS majors based upon the influence of family (parents and siblings), peers, and K-12 education. Gender, race/ethnicity, and community types were thought to constitute the demographic "lens" through which these influences affect student perceptions of both computing and CIS majors.

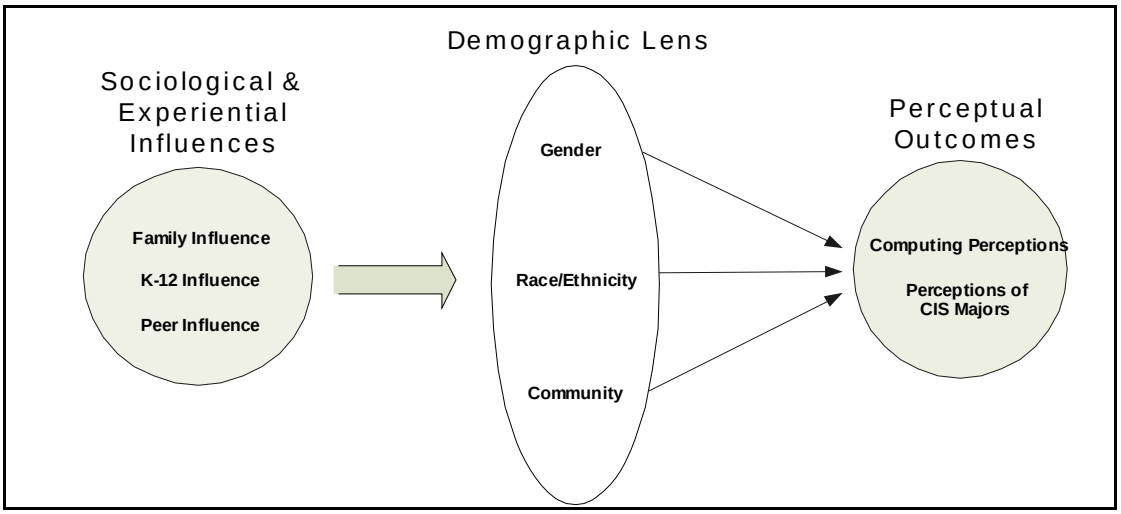


Figure 1: Conceptual Model

The quantitative aspect of this study involved the use of a custom survey administered in a pilot test and a primary implementation. The pilot test of the survey took place over two semesters during the 2007-2008 academic year. The custom survey instrument was administered to seven sections of two introductory Information Science (IS) and Management Information Systems (MIS) courses at two campuses of the Pennsylvania State University. The survey instrument was revised based on the pilot test results before primary implementation. The revised survey instrument was then administered to students attending a first-year testing and orientation session at two campuses of the Pennsylvania State University during the summer of 2008.

The custom survey instrument primarily involved likert-style and yes/no-style questions. To facilitate part of the analysis, a *Computing Perceptions* index variable was created from the responses to seven of the likert-style questions. Respondents were asked to rate their level of agreement with a series of statements using a five-point scale. These seven statements were designed to measure respondents' perceptions about the importance of computing in their present lives, in their future employment, and in society as a whole. The *Computing Perceptions* index was used as a proxy measure to gauge respondents' attitude towards computing. This measure was found to be reliable in both the pilot test ($\alpha = 0.76$) and the primary implementation ($\alpha = 0.84$) phase. The qualitative aspect of the study involved two focus groups held at one of the participating campuses during the spring of 2009. These focus groups were approximately one hour in length and involved 12 questions. The purpose of these focus groups was to delve deeper into student perceptions of computing and CIS majors, as well as experiential and societal influences on those perceptions.

PILOT TEST RESULTS

The pilot test of the survey took place during the fall 2007 and spring 2008 semesters. The survey netted 188 responses (121 in fall 2007, 67 in spring 2008) for a response rate of 69% (N=272). The respondent population was 31.7% female and 68.3% male. The respondents came from different types of communities (23.8% rural, 51.4%

suburban, 22.2% urban, and 2.6% non-US resident) and from different academic classes (26.2% freshmen, 44.9% sophomores, 21.4% juniors, and 5.9% seniors). The vast majority (97.3%) reported being between 18 and 30 years of age. Some level of racial and ethnic diversity was present (73.1% White/Caucasian, 7.5% African-American/Black, 14.0% Asian, 4.3% Latino, 1.0% other). The respondents were primarily non-Science, Engineering, Technology, and Mathematics (STEM) majors (88.2%).

Univariate Analysis of Variance (ANOVA) was performed to identify significant relationships between the *Computing Perceptions* scores and an additional series of likert-style questions. Two significant relationships were found. The level of agreement with the statement "My High School teachers and/or guidance counselors stressed the importance of Computer Technology skills" was found to be significantly related to *Computing Perceptions* ($F=2.53$, $df=4$, $p < 0.05$, partial $\eta^2=.053$) though the effect size was small, and no pairs of response categories were found to be significantly different. The level of agreement with the statement "My parents stressed the importance of Computer Technology skills" was also found to be significantly related to *Computing Perceptions* ($F=5.19$, $df=4$, $p < 0.01$, partial $\eta^2=.103$). Post-hoc Scheffe analysis found significant differences between those who strongly agreed with the statement and those who were neutral. No significant relationships involving gender, race/ethnicity, or community type were found.

PRIMARY IMPLEMENTATION RESULTS

The primary implementation of the survey took place during the summer of 2008. The primary implementation of the survey netted 346 responses, or a response rate of 39% (N=897). The respondent population was 48.8% female and 51.2% male. The respondents came from different types of communities (33.4% urban, 43.8% suburban, 22.5% rural, 0.3% non-US resident). Given the setting, all students can be considered freshmen. The vast majority (98.8%) reported being between 18 and 30 years of age. The summer 2008 respondents were more diverse in ethnicity than in the pilot phase (68.5% White/Caucasian, 21.1% African-American/Black, 4.7% Asian, 4.5% Latino, 1.2% other). Once again, the respondents were primarily non-STEM majors (65.4%). These demographics indicate a more diverse mix of student backgrounds than those found in the pilot test.

Gender was found to have a significant influence on *Computing Perceptions.* An Independent Samples t-Test found significant differences in *Computing Perceptions* between the genders ($t=-2.36$, $df=326$, $p < 0.05$), with females having a higher mean *Computing Perceptions* score than males. More specific analysis revealed that gender differences could be found in the perception of computing majors leading to more successful careers ($X^2=10.93$, $df=4$, $p < 0.05$). Females were less likely to agree that a computing major would lead to a more successful career. Significant differences were also found between genders when asked about the primary influences on their technology use ($X^2=17.48$, $df=5$, $p < 0.01$). Males tended to be more self-reliant, whereas females were more likely to be influenced by families, friends, and teachers.

Significant differences were also found between genders when asked about whether high school teachers and/or guidance counselors stressed the importance of computer technology skills ($X^2=9.94$, $df=4$, $p < 0.05$). Females were more likely to be neutral or

disagree with this statement. Females were also significantly less likely to expect that computer technology would be used regularly in their future career (X^2=14.53, df=4, $p < 0.01$). Despite these gender-based differences, the relationship between gender and the belief that CIS majors are difficult was not significant. Significant differences were found, however, between race/ethnicity and the belief that CIS majors are difficult (X^2=41.30, df=16, $p < 0.01$); a similar occurrence was found based on community type (X^2=29.49, df=12, $p < 0.01$). In general, minority groups were more likely to see CIS majors as difficult, and urban dwellers were less likely to see CIS majors as difficult. No other significant relationships based on race/ethnicity or community type were found.

Regardless of gender, the factors relating to K-12 and family influences provided significant and surprising Univariate ANOVA results. The level of agreement with the statement "My High School teachers and/or guidance counselors provided me with information on Computer majors and careers" was found to be significantly related to *Computing Perceptions* (F=5.10, df=4, $p < 0.01$, partial η^2=.060) though not in the expected direction. Post-hoc Scheffe analysis indicated that students who "disagreed" with the statements had significantly *higher* computing perceptions than those that "agreed". A similar phenomena was found in the level of agreement with the statement "My High School teachers and/or guidance counselors stressed the importance of Computer Technology skills" (F=10.94, df=4, $p < 0.01$, partial η^2=.121) and with the statement regarding parental emphasis on computer technology skills (F=5.86, df=4, $p < 0.01$, partial η^2=.068). In general, it seems that the more emphasis parents and high schools place on computer technology skills, the lower the respondents' perceptions were about computing – certainly not the intended effect.

FOCUS GROUP RESULTS

To supplement the quantitative survey results, two focus groups were held at one of the participating campuses during the spring of 2009. These focus groups were intended to further explore the influences on perceptions of computing and K-12 computing experiences among a group of students. A total of 16 students consented to participate. The volunteers were primarily STEM majors (15), freshmen (13), male (15), and white/Caucasian (9). All participants were in the 18-30 age range. Efforts to hold additional and more diverse focus groups were unsuccessful.

The most significant theme to arise from the focus groups involved the influence of students' K-12 education. Many of the focus group participants reported having taken some computer courses in high school, but the range of content and depth of coverage was mixed (e.g. programming, graphic arts, and office/keyboarding courses). This aligned with the primary implementation survey results, where 81.3% of respondents reported taking a high school computer course and 92.7% reported using computers in their K-12 classes. Some students appeared frustrated in the computing courses they were offered at the K-12 level, seeing them as "basic" and reporting a lack of computing knowledge on the part of some teachers. Several students reported that there was little reinforcement of computing skills in other K-12 subjects. Students also reported that high school guidance counselors did not provide much if any information on CIS majors or careers. The information participants had about CIS majors and careers prior to college came primarily from teachers, family, and friends.

DISCUSSION

The results suggest that the gender differences in computing perceptions persist. Females were found to have significantly higher perceptions of computing, but this does not necessarily translate into a positive perception of computing–related careers. Particularly troubling was the disparity between genders regarding the emphasis placed on computing technology skills by high school teachers and guidance counselors, though the focus group and combined results suggest that this problem may be endemic across genders. Interest in CIS majors must be sown at the middle- and high-school levels in order to be successful. This requires a concerted effort between CIS faculty, K-12 teachers, and guidance counselors – something that so far seems to be lacking. Research shows that the pipeline for students in CIS and other STEM majors is increasingly porous as students progress from middle school through college, especially for females and minority groups [8]. Efforts to reverse the gender gap cannot begin when students arrive on campus; such efforts must be longitudinal and multi-faceted in nature.

The results indicating inverse effects between the level of parental and high school emphasis on computer technology skills and *Computing Perceptions* have a few possible causes. Perhaps as technology has receded into the background – becoming a critical yet often unseen part of everyday life – young people no longer see computing skills as something they need to *learn*, but something they intrinsically *have*. Computer technology is taken as a given, and is something that most young people are expected to have exposure to throughout their formative years. This exposure is expected to translate into literacy, though empirical evidence suggests otherwise ([9]). While colleges and universities sometimes criticize the lack of specific computer skills among entering freshmen, the students themselves may not see how computing "fits" in general rather than specific terms. Certainly a lack of multidisciplinary emphasis on computing skills in the K-12 experience – and traditionally at the collegiate level – may be partially to blame.

While research confirms that computing is integral to the daily lives of many students, technology usage does not necessarily translate into being technologically literate ([5], [6], and [9]). It is important to note that the K-12 computing experience is evolving. As state governments recognize the need for students to have "21st Century Skills", many policies and programs are underway to build technological literacy in K-12 students. These programs, such as Pennsylvania's Classrooms for the Future program and Maine's Learning Technology Initiative, are attempts to better integrate technology across the curriculum and to place technology use in context. As these programs mature and notions of technological literacy evolve, it is reasonable to expect that students' perception of computing and CIS careers – as well their role in organizations, business, and society – will evolve with them. Future research into K-12 efforts at computing and technological literacy education may shed light on how – and why – the K-12 system is (or is not) producing students with better perceptions of computing and CIS careers.

LIMITATIONS

There are three observed limitations associated with this study. The focus on a single university (despite the two-campus nature of the study) limits external validity. Second, the pilot test results are potentially biased given the nature of the setting. The pilot test setting was chosen as a sample of convenience. Third, the small number of focus group participants limits the value of these results. Repeated attempts to increase the number of focus groups proved unsuccessful.

ACKNOWLEDGEMENTS

The authors wish to thank the following Pennsylvania State University students for their assistance in this implementation of the study procedures: Francine Lewis, Timothy Haughney, Frederick Maley, Matthew Laubenstine, and Rachel Bendetti. The authors also wish to thank Penny Carlson and Michael Verhagen, Division of Undergraduate Studies at the Pennsylvania State University, for their assistance.

REFERENCES

[1] Busch, T., Gender differences in self-efficacy and attitudes toward computers, *Journal of Educational Computing Research 12,* (12), 147-163, 1995.

[2] Guzdial, M., Education: Teaching Computing to Everyone, *Communications of the ACM , 52,* (5), 31-33, 2009.

[3] Madigan, E., Goodfellow, M., Stone, J., Gender, perceptions, and reality: technological literacy among first-year students, *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (Covington, Kentucky, USA, March 07-11, 2007).

[4] Hanson, K., Space Aliens? Women, ICTs, and Gender-Equitable Electronic Resources, 2002, http://www2.edc.org/GDI/publications_SR/publications/SpaceAliens.pdf, Retrieved May 01, 2009.

[5] Hilberg, J., Meiselwitz, G., Undergraduate Fluency with Information and Communications Technology: Perceptions and Reality, *Proceedings of the SIGITE 2008 Conference on Information Technology Education*, Cincinnati, OH.

[6] O'Hanlon, N., Net Knowledge: Performance of New College Students on an Internet Skills Proficiency Test, *The Internet and Higher Education, 5,* (1), 55-66, 2002.

[7] Shashaani, L., Gender-Differences in Computer Experience and its Influence on Computer Attitudes, *Journal of Educational Computing Research 11,* (4), 347-367, 1994.

[8] Stake, J., Nickens, S., Adolescent Girls' and Boys' Science Peer Relationships and Perceptions of the Possible Self as Scientist. *Sex Roles, 52,* (1-2), 1-11, 2005.

[9] Stone, J.,Madigan, E., Inconsistencies and Disconnects: A Disturbing Report on the Information and Communication Technology Skills of Incoming Freshmen, *Communications of the ACM, 50,* (4), 76-79, 2007.

[10] Tindall, T., Hamil, B., Gender Disparity in Science Education: The Causes, Consequences, and Solutions, *Education, 125,* (2), 282-295, 2004.

[11] Wilson, G., Alvarado, C., Campbell, J., Landau, R., Sedgewick, R. CS-1 for scientists, *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA, March 12-15, 2008).

[12] Zweben, S., *Computing Degree and Enrollment Trends,* Washington, D.C.: The Computing Research Association, 2009.

USING PEER LED TEAM LEARNING TO ASSIST IN RETENTION IN COMPUTER SCIENCE CLASSES*

Carolee Stewart-Gardiner
Computer Science Department
Kean University
Union, NJ 07083
908 737-3795
cstewart@kean.edu

ABSTRACT

Recent work in Peer Led Team Learning (PLTL) has resulted in development of strategies and exercises for implementation of PLTL in the first course in Computer Science as well as two higher-level undergraduate courses at a public university with a largely commuter student population. Initial results identify many reasons why students are unable to take full benefit from offered PLTL Workshops, while demonstrating the significant advantages that junior and senior Computer Science majors feel they received from the PLTL experience. Suggestions for future success with PLTL are included.

INTRODUCTION

Retention in Computer Science is not only a short term objective but also a long term goal for the academic community. But what efforts work, and where are the pitfalls? This paper will discuss the PLTL experience at Kean University with an emphasis on how it pertains to retention of Computer Science and STEM (Science, Technology, Engineering, & Math) majors. PLTL emphasizes self reliance and small group reliance in problem solving in the sciences. PLTL strongly encourages small group synergy, and passing on the understanding from one year of students to the next.

BACKGROUND AND PRIOR EXPERIENCE

In August 2006 the Epsilon Corps grant from NSF (DUE #431637), funded several faculty to visit CCNY in Manhattan NY to learn about PLTL Peer Led Team Learning, with an eye on retention and community building among STEM majors at our mainly commuter undergraduate state university. The PLTL workshop program was firmly established as a means for retention, success building for the students, and community building in Chemistry[1], which began 15 years ago at CCNY as a form of enrichment with NSF grant money. The PLTL program has grown into a for-credit course in a number of colleges, in a variety of STEM departments. Use of PLTL in Computer Science has been scarce, with the exception of one NSF funded program of a consortium of highly selective schools[2] in an emerging scholars program[3] (NSF CNS-0420343 and CNS-638510). Implementation outcomes on non-residential public comprehensive university campuses in Computer Science were not well-known prior to this work.

Kean faculty looked to improve retention in many STEM programs. Initial PLTL workshops were in the Math pre-calculus course, which all science students are required to take. The math students at Kean regarded PLTL as remedial tutoring. As a result, the program appears to have made very little impact on the Math retention in Pre-calculus.

The Computer Science faculty at Kean regarded PLTL as a potential technique for improving retention in CS0 (Computing Fundamentals w/ Java) to CS1(Computer Organizations & Programming). CS0 is required for CS and IT majors, and many STEM majors. When Computer Science majors had numbers like 800 about 20 years ago, the loss of students in CS0 was not a cause for concern. Today CS major numbers are closer to 100, and each student who drops CS0 is noticed, as that is the first step to leaving CS or other STEM majors. Therefore, PLTL was adopted by the Computer Science faculty, specifically for use in CS0 classes. The objectives are to increase retention, improve student course success, and improve a sense of community among STEM majors.

METHODOLOGY

The same PLTL-style of problems were used in workshops for both CS0 and two different required Junior/Senior classes taught in Computer Science (namely Systems Analysis and Design in Spring 08, and Distributed Systems in Fall 08) to improve the success rate in those courses. The results varied in the different classes.

Over the past 3 years, the Epsilon Corps NSF grant NSF (DUE #431637) funded the initial development of the PLTL workshop problem sets for each of 3 courses: CS0, Distributed Systems, and Systems Analysis/Design. Materials were initially solicited from other colleges, such as the Rutgers RESCS project[4], to be sample materials for use and comparison. However, many of the materials provided were for CS1 classes and were only a little helpful in a CS0 course. The PLTL group at CCNY provided materials helpful to the workshop concept[1] and structure, but the majority of Computer Science PLTL workshop materials were developed in collaboration with our students.

Student Peer Leaders were responsible for facilitating weekly workshops, and assisted in workshop design. It was found that the faculty member responsible for managing the PLTL sessions was most effective when they also taught a CS0 section. Other helpful faculty members were responsible for other CS0 sections.

WORKSHOP IMPLEMENTATION

Initially, CS0 students were reluctant to attend the workshops. The students thought the workshops, held outside of normal class time, were tutoring for failing students. To overcome this perception, an introductory workshop was held in the CS0 classes, during a visit by a Peer Leader. This "taster session" was designed to show students what a PLTL workshop exercise was like. Student attendance continued to remain at low levels.

During Spring 08, workshops for Systems Analysis/Design (a required course) were added. The faculty member overseeing the PLTL program also taught the Systems Analysis/Design course. This PLTL Workshop had high student attendance. The CS0 PLTL Workshops were enhanced as "extra credit" by all CS0 teaching faculty.

During Fall 08, a PLTL workshop was offered in Distributed Systems (a required course), featuring a recent alumnus who had taken the course just before graduation, as the PLTL leader, and student attendance also increased for that workshop.

CS0 PLTL Workshop attendance remained disappointingly low. Only a few CS0 students came, each thinking it was another form of tutoring for failing students. In May 2008, a survey was conducted to find out why more students were not coming to the CS0 PLTL workshops. *Over 60%* of the students said it was because the times were in conflict with other classes or work schedules(Figure 2), or they did not feel they had enough time.

In response to these identified concerns, another student leader was added in Fall 08 and additional workshops were offered at varying times (including Saturday), to try to draw students in at a time when they might be available.

Fortunately, the Saturday workshops worked. The majority of junior/seniors came, and freshmen/sophomore CS0 student attendance increased. With increased students, the PLTL workshops began to have the positive effects hoped for. A core of four CS0 students, with three more students less frequently attending, came faithfully on Saturday.

As an example of the positive effect experienced by the students, consider one of the core students attending. Initially, he had a "C" in the class, which began to improve shortly after his attendance at PLTL workshops. He was an older non-traditional student, and an IT major, and he became the leader. His final course grade was a "B". When the CS0 group got together, they helped each other to figure out the problems. The PLTL leader became the facilitator expected, rather than an answer-giver, as is usual in tutoring.

The Distributed Systems students came, and the alumnus PLTL leader was a real success with the students. They worked faithfully on the workshop problems, which had been prepared more oriented toward the complex concepts of the course. After the workshop, students stayed to work on their class project, with the alumnus as facilitator.

During Fall 2008, another set of surveys was given to the students, and we tried to standardize the questions. The results, illustrated in 3 Figures below, show the challenge students have in their first and second years identifying what clearly helps them succeed. By the final years of their undergraduate degrees, the merits and benefits of PLTL workshops are more clearly understood by students.

At universities with a commuter population, students struggle with commitments, beyond courses. How can we help these students? PLTL workshops both help and hinder such students. We gave surveys to the 2 Junior/Senior courses, and 2 semesters of 4

sections (20 students/section) of CS0 who are freshmen/sophomores (Figure 1). The surveys from all 4 years of students, from both semesters, showed that 80% of students who attended PLTL, felt that future students in their course should attend PLTL workshops. And 60 to 80% of students felt PLTL workshops improved the sense of community among the students. This is impressive coming from students where many do not know each other, and tend to depart campus quickly for work, or other commitments.

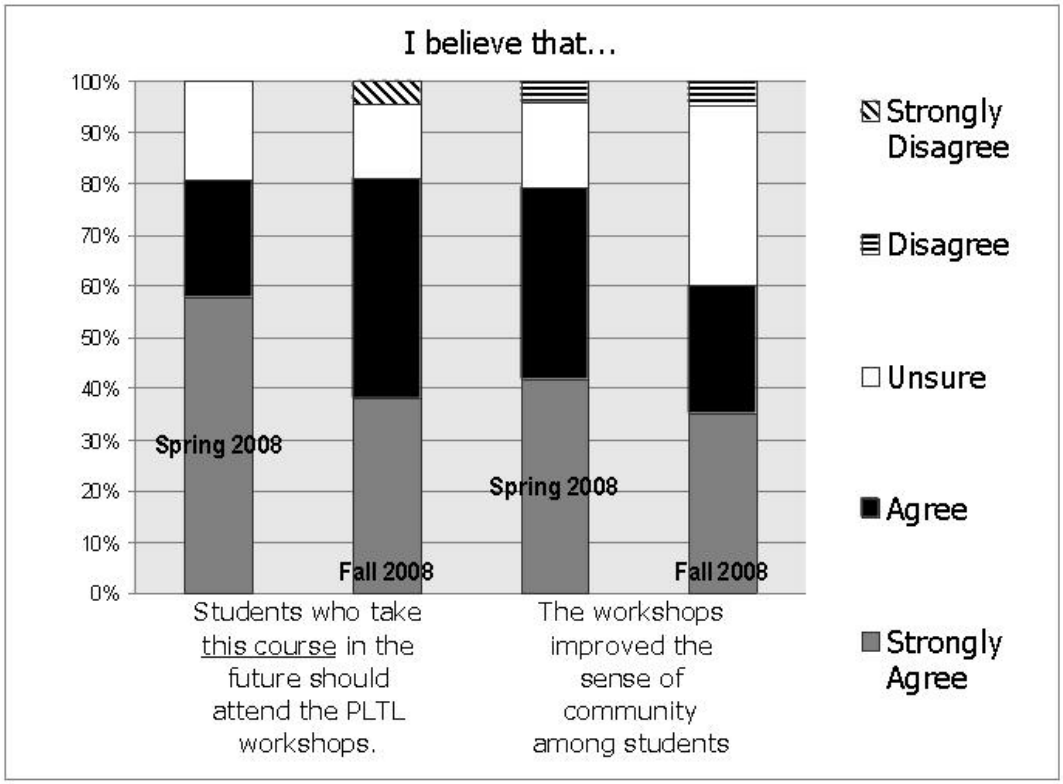


Figure 1. Student perception of PLTL Workshops

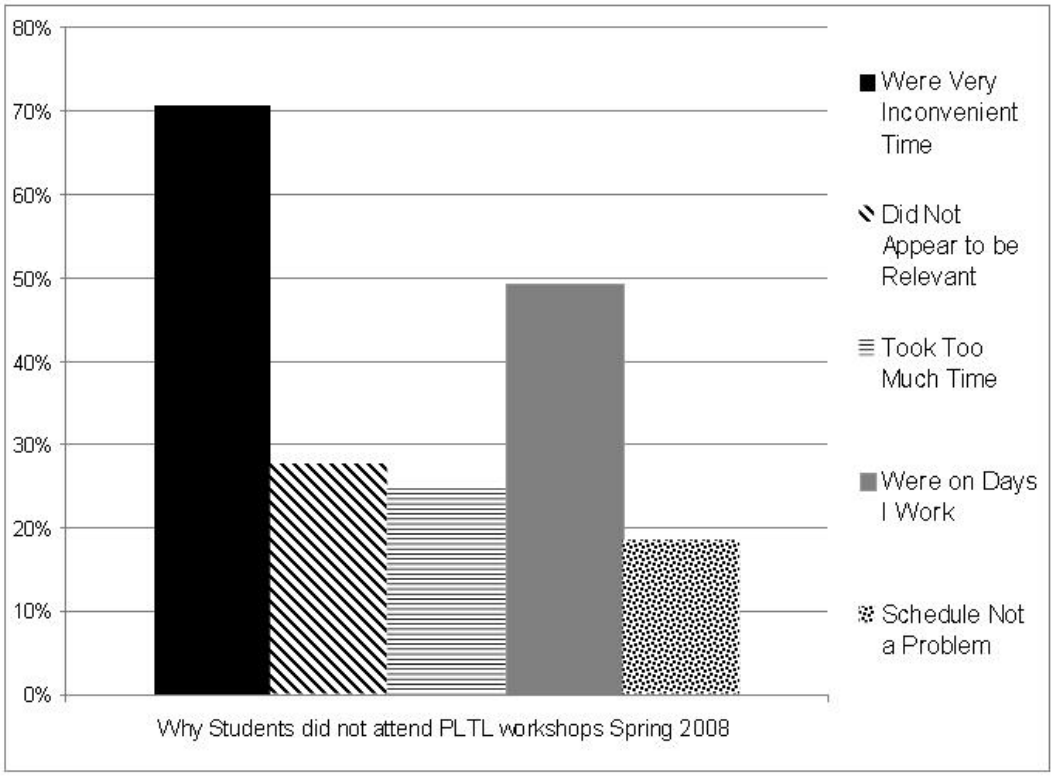


Figure 2. Summary of student reasons for not attending PLTL Workshops

Because Kean students are mainly commuters, it was not surprising that the survey of freshman/sophomores in CS0 showed the results about why they did not come(Figure 2). Clearly the reasons for non-attendance were dominated by the other commitments this student body, predominently commuters, has to deal with in their lives.

Reponses of students who found time to come to PLTL workshops, generally found it very valuable (Figure 3). Most freshmen/sophomores are unsure of the impact of PLTL on their ability to continue in STEM majors, but over half of the freshmen and sophomores agreed the workshops influenced their overall performance in the class. However, almost all juniors/seniors found PLTL workshops influenced their ability to remain a STEM major (partly because these were required courses), and ALL juniors/seniors strongly agreed PLTL influenced their overall course performance.

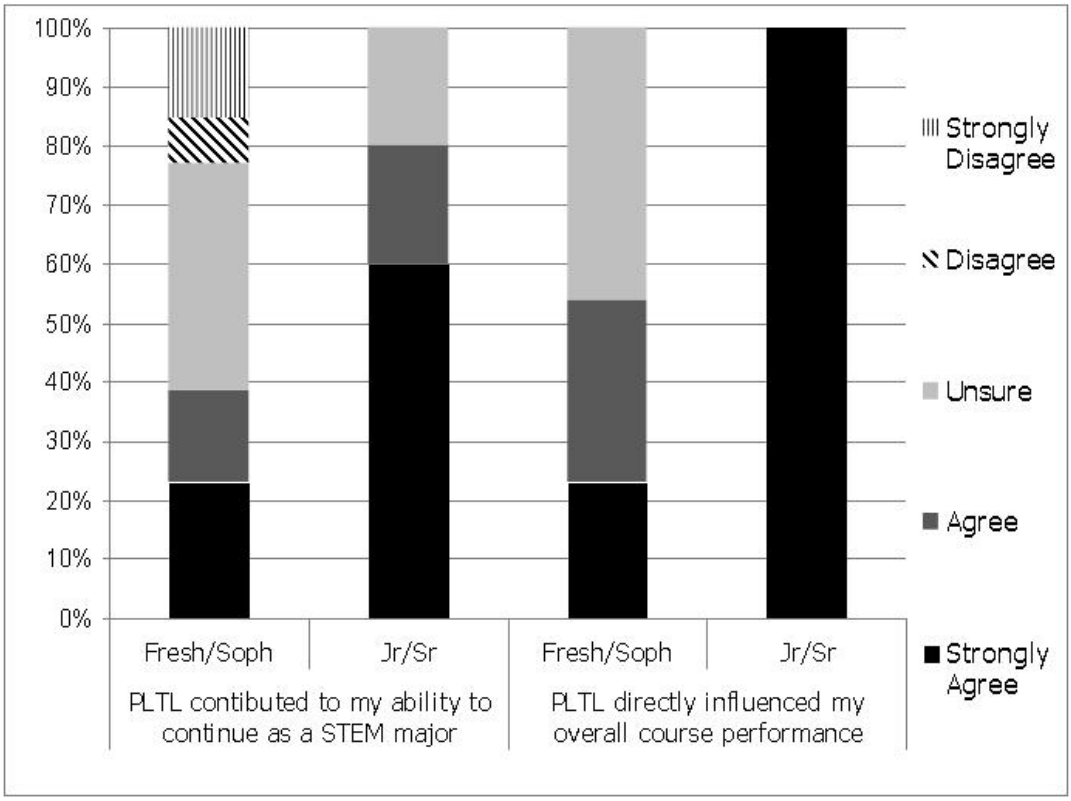


Figure 3. Comparison of perceived value of PLTL Workshops by students

CONCLUSIONS

PLTL workshops in Computer Science can improve retention and a sense of community among STEM students[5]. Initially, Kean students attend the workshops for extra credit, but then stay for the workshop content, which they see as beneficial and the success they felt in the course as a result. The outcomes of the use of PLTL workshops in Computer Science include the following "best practices":

1. The coordinator teaches the course: When the PLTL coordinator was one of the professors teaching the course, it helped to coordinate the workshops with the course.
2. Weekly reminders to attend help: When the professor reminded the students each week, that the PLTL workshops were available, and strongly encouraged the "C" and "D" students to go, attendance improved.
3. Post-midterm, PLTL attendance improves: When students were not happy with their midterm grade, and they wanted a better grade in the final exam, attendance went up. This helped some of the midterm "C" students who wanted a "B" to start attending.
4. Peers and community matter: When the small group began by coming together, and continued together, their sense of community improved. Every one of that small

group passed the course with a "C" or better, and most of them were in danger of "D" shortly after the Midterm, when the course gets more difficult.

5. Leadership and leaders are important: When one of the students rose to become a small group leader from among the students, they began to function as a group. The fact that the PLTL leader in the seniors course was an alumnus, and the juniors/seniors could see the success of the PLTL leader, encouraged them, even when the material was complex.
6. Opportunity to succeed: When the juniors/seniors who cared about their chosen future, began to see the workshops as an opportunity, instead of a requirement, they willingly came on Saturdays.
7. PLTL workshops are enrichment: It is important for faculty to stress to all students that PLTL workshop problems are meant to enable them to improve their grades, as well as the "D" students to pass. The PLTL workshops help students to get more out of small group work. The Juniors/Seniors understood this, but Freshmen/Sophomores did not, until they came to workshops more than once.

Designing the PLTL workshops to enable students to become more self sufficient, and group reliant is not an easy objective. The workshop problems must be constructed with enough leading questions to help students find the path to completion, without giving it all away. Our workshop problem sets are available on the Kean website[6]. Teaching PLTL leaders how to facilitate in workshops, and not revert to the role of answer givers on the problems students are having trouble with, as tutoring usually does, presents a real challenge, especially when PLTL leaders have not been students in PLTL workshops. The design of workshops, and the role of PLTL leaders are critical success factors of the PLTL program, since the goal is to improve the students' ability to be self and small group reliant, rather than just a better grade in the course through answers on difficult problems. Initial semesters are especially difficult, as Peer Leaders have not been through the process. Having PLTL leaders help weekly in modifying workshop problems from a student perspective, with helpful leading questions, assisted greatly in this effort.

The sense of community has some aids as well. The fact of it being a Peer Leader mattered to the students, to be able to ask what seemed to be a 'stupid' question. Having the same Peer Leaders each week, helped build a sense of community. The students often voiced how helpful the peer leader was, and groups worked together for the success of all. By working together each week Peer Leaders improved their sense of community too.

FUTURE WORK

The most significant barrier to greater success with PLTL Workshops in Computer Science at a public university, is the constraint of student time. Students commuting to school, work schedules, and lack of hours on campus remain as significant barriers to student success. One future plan is to increase the time required for CS0 by 1 hour, to incorporate a PLTL workshop once a week as a required part of class time. This is planned for 1 or 2 sections; other sections will be the same 4 hours as before. We intend to study grade assessment, at that time.

With the conclusion of the Epsilon Corps NSF Grant, which has funded this effort, future funding of Peer Leaders to conduct the workshops is in question. Internal funding

might be possible, but is dependent on the number of students positively impacted by the PLTL Workshops. Sustainability is a challenge, that the Gossser team[1] also recognizes.

Offering PLTL Workshops to students as a single credit course (as is done by Rutgers in an optional recitation style[4]) is not yet a possibility at Kean. For a few groups of students the PLTL Workshops worked well. It is not possible to generalize to all STEM majors taking CS0. This project has identified that CS0 students who attended PLTL Workshops did benefit, and students who attended workshops for advanced classes benefited more. PLTL leaders thrived on the opportunity to guide others through classes. We at Kean University hope to continue to offer PLTL Workshops in Computer Science. We hope to contribute to the collection of workshops that are at Beloit University[3].

Thanks are due to Cindi Dunn of Kansas State University, who helped with survey standards and survey data. Thanks are due to Patricia Morreale of Kean University for assisting editing this paper, and to Michael Halper, Jeffrey Cheng, George Chang, and Patricia Morreale who taught CS0 at Kean, and were helpful during this study. And many thanks go to all the PLTL leaders who worked long hours to facilitate in the success of their fellow students.

REFERENCES

[1] Gosser, D., Cracolice, M., Kampmeier, J., Roth, V., Strozak, V., Peer Led Team Learning: A Guidebook, Prentice Hall, 2001.

[2] Horwitz, S., Rodger, S.H., Using Peer-Led Team Learning to Increase Participation and Success of Under-represented Groups in Introductory Computer Science, ACM SIGCSE 2009, Vol 41, (1). pp163, 2009.

[3] Rodger, S.H., Huss-Lederman, S., PLTL in CS, www.pltlcs.org, 2007.

[4] Horwitz, S., Ryder, B., Huss-Lederman, S., Peer Led Team Learning in Computer Science Workshop, ACM SICCSE 2007, Vol 39 (1), 2007.

[5] Huss-Lederman, S., Chinn, D., Skrentny, J., Serious Fun: Peer-Led Team Learning in CS, ACM SIGCSE 2008, Vol 40 (1), pp330, 2008

[6] Stewart-Gardiner,C., PLTL Workshops, www.kean.edu/~cstewart/pltl.html , June 2009

STUDENT EVALUATION IN MONITORED TEAM PROJECTS*

Joo Tan
Kutztown University of Pennsylvania
(610)683-4413
tan@kutztown.edu

ABSTRACT

Effective communication is imperative for any team-oriented enterprise. Within the academic place, students often lack the skills to work efficiently in a team. In a one semester course, students collaborate on a client-based project in a team setting. Objective evaluation of student performance is important. Using various aspects of student evaluation, grading criteria for the course is discussed. Qualitative measures of student development were collected and are included in this paper.

INTRODUCTION

Team projects provide plenty of opportunities for students to practice interpersonal skills in the academic environment. However, they are hard to teach for various reasons – among them is the difficulty of assigning balanced workload to each student in a project team, motivating students to put forth their fair share of work, and objectively grading students on their participation/contribution in the project. Moreover, the instructor must put in extra work in managing project teams outside of the class room.

This paper is a culmination of the author's work with team project courses over a six year period. The emphasis of the course has always been on teamwork. Team skills are promoted by encouraging communication, participation, contribution, and responsible time management. Objective grading based on contribution and participation for team projects is important and has been given careful consideration. The evaluation criteria used to measure student performance is a primary focus of this paper.

COURSE GOALS

Within the context of an undergraduate environment, many factors [5] might influence the effectiveness of a project team. Our focus has been on communication, responsibility, and time management. While teaching team project courses, the author has continually injected techniques of collaborative work by introducing new and innovative ways of motivating students to meet higher standards of system development.

There are three goals for the course. First, it provides students with opportunities to improve on their soft skills. Second, it offers a different perspective of software development that students have not seen before – by going through the entire process of software development, from analysis through implementation/testing. Lastly, it lets students experience software development from within a team environment [7].

TEAMWORK

Since team projects depend on both the technical and social skills of people on the teams, opportunities for building teamwork are provided throughout the semester. To effectively meet the goals stated, the instructor instituted a number of requirements into the course. They are discussed in the following sub-sections.

3.1 Team Formation

Students are divided into project teams on the first day of class. Based on a number of factors - the instructor's knowledge of the students and consultation with the instructor's peers - the project leader for each team is first selected by the instructor. The other members of the team then gets chosen by the project leader – project manager, systems analyst, system designer, system developer, and system tester. This selection process [6] has been used for the past three years and may be a contributing factor to recent team successes (see section 4). Other methods of team selection have been discussed elsewhere [3,5]. Students' comments about this selection method were conducted at the end of the course.

3.2 Technical Writing

As part of their contribution to the team project, each student must write two technical documents. The students decide amongst themselves which document they must write – this usually corresponds to the project role that they have been assigned. In the professional workplace, technical documents are commonly written and circulated for review. To aid students in writing these documents, template outlines have been provided by the instructor. The templates have proven to be a time saver for both students and instructor.

3.3 Meetings

Weekly team meetings offer opportunities for students to engage in discussion of subject matter during a project's lifecycle; from project planning to requirements gathering and specification, discussion on system design alternatives, issues of

programming language or environment. Students were strongly encouraged to use these meetings to improve their communication skills. To make their meeting more efficient, an agenda must be posted before each meeting on a collaboration tool named Basecamp [2,4] that is provided by the instructor. After the meeting, a person in each team has to submit meeting minutes. The weekly meeting provides a setting for interpersonal communication between team members when discussing topics relevant to their project. Egos sometimes clash among people with differing expectations. When conflicts occur, the course instructor must be alert to such situations and respond quickly to resolve them.

3.4 Individual Journals

A deadline of 11:00am on Saturday was set for students to submit their personal journals each week. Three points are awarded for submission of journal on time, one point if submitted by end of day on Saturday, and no point is given otherwise. Important observations can be made from the individual journals; namely the "useful" contributions by each student. This is an important component of objective grading and is highly recommended when teaching team project courses.

3.5 Real Clients

To learn teamwork effectively, students work on real clients projects [8]. A contract between the client and the project team may be written to ensure that an agreement is made for successful completion and delivery of system. This arrangement allowed students to work on their social skills when interacting with the client as well as with end users. Since the end system must be delivered to the client towards the end of a semester, there is added incentive for teams to work harder to complete a quality product within specific time constraints.

HISTORY OF PROJECTS

4.1 Past Results

While teaching at Mansfield University, the author taught a Systems Analysis and Design course. Student teams ranged from 5 to 7 people. Table 1 shows projects that were explored in the course. Results have been classified into three categories: Success, Partial, and Failed. A Success rating meant that the system is currently in use by the client. A Partial rating is considered a successful project to the extent that they have completed at least 90% of the requirements specified by the clients. For Partial ratings, systems might not have been deployed for a variety of reasons. A Failed rating indicates that the system was either canceled or did not satisfy the requirements. In the early stages (first three years) the success rate was about 27% (3/11), with Partial success rate at 36% (4/11).

Table 1. Team Projects from Fall 2002 - Spring 2005

System	Year	Result	Status
Video Store Rental	02-03	Partial	Not Deployed
Employee Tracking	02-03	Partial	Not Deployed

Pizza Delivery	02-03	Failed	Canceled
Student Accountability	02-03	Success	In Use
Craft Ordering	03-04	Success	In Use
Bank HR Intranet	03-04	Success	In Use
Inventory Control	03-04	Partial	Not Deployed
Online Bookseller	03-04	Failed	Incomplete
Nursing Care Plan	04-05	Failed	Incomplete
Music CD Inventory	04-05	Failed	Canceled
University Curriculum	04-05	Partial	Unclear

4.2 Recent Results

More recently, projects conducted in a junior/senior level Software Engineering course at Kutztown University are shown in Table 2. In these course offerings, the rate of success is 38% (5/13) and the partial success rate rose to 54% (7/13). This success depends on a combination of many factors. Some of these factors are discussed in section 5. Data will be collected over the next three years to see how they compare to current statistics.

Table 2. Team Projects from Fall 2005- Spring 2009

System	**Year**	**Result**	**Status**
Auto Parts Inventory	05-06	Failed	Incomplete
Medical Supply Inventory	05-06	Partial	Unclear
Employee Time Management	05-06	Partial	Unclear
School Trouble Ticket	06-07	Success	In Use
Nomination and Balloting	07-08	Partial	Unclear
Dynamic Mall Directory	07-08	Success	In Use
Online Tee-shirts	07-08	Partial	Not Deployed
Used Car Inventory	07-08	Partial	Unclear
Online Advising Scheduler	08-09	Partial	Incomplete
Business Management	08-09	Success	In Use
Ambulance Dispatch	08-09	Partial	Incomplete
Online Advisement Scheduler	08-09	Success	In Use
Sports Team Geo Locator	08-09	Success	In Use

4.3 NEWER IDEAS

4.3.1 Mid-Semester Evaluation

The usual practice for these team projects has been to complete one full cycle of the software development lifecycle (SDLC) within the constraints of one semester (15 weeks). Within the past year, the instructor has added a requirement for delivery of a mid-semester prototype. A prototype of the system provides a useful milestone for assessment of project status as well as evaluation of system progress. A mid-semester presentation has also been incorporated. Clients are invited to the presentation to observe the progress that has been made on the system being built.

4.3.2 Hands-On Testing

Each semester, at least two system testing sessions are conducted in the department computer lab. Students are asked to run test cases from the test specification document in the first session. Problems found during system testing are collected and compiled into a modification request report, one per team. A modification request board (consisting of the course instructor and two members per team) meets to review the report for each team. The defects written are reviewed for proper severity level, validness, and reassignment to the correct developer. In the second session, regression testing is also conducted. Before each version of the system for testing, release notes are written for each team to inform the testers of the features that have been delivered as well as known problems at that time.

EVALUATION CRITERIA

In team project courses, the evaluation criteria must be made explicit to students. It is highly recommended that standards for evaluation be clearly stated at the beginning of the course. All the components for individual student evaluation should be included in the course syllabus. Furthermore, this should be done as objectively as possible. Wilkins and Lawhead [8] discussed potential evaluation strategies in their paper. Individual components of student evaluation are discussed next.

5.1 Personal Journals

At the end of every week, each student must write a personal journal documenting his/her contributions to the team. Points are awarded for on time submission. These journals must be posted to BaseCamp for validation by a set deadline.

5.2 Meeting Minutes

Weekly meetings held outside the class allow each student to participate in the project on a regular basis. Points are awarded each week for attendance at these meetings. Further, a person in each team must document the minutes for his/her team and submit the minutes onto BaseCamp by a specific deadline.

5.3 Technical Documents

Since each student has responsibility for authoring two technical documents, points are awarded accuracy of contents and professionalism. Since students are usually not familiar with either the structure or content of these documents, the instructor has provided outline templates for reference. These templates have proved to be useful both as a time saver for the instructor and as guidelines for students.

5.4 Contribution

Contribution refers to the relative significance of ideas, technology, and added value from each student to help make his/her project a success. Based on the instructor's observation (50% of this grade) of each student's contribution throughout the entire semester, points are awarded (using on a 4-point scale) at the end of the course. It is highly recommended that the scale used be clearly explained to students in the first week of the course. The personal journals often convey a lot of information about the contribution made by students. Peer evaluations (the other 50%) conducted at least three times during the semester are also taken into account.

5.5 Participation

Participation refers to the amount of time and energy that students exerted towards the completion of their project. Active participation at meetings and testing sessions count towards this evaluation component. A 3-point scale is used by the instructor. Students also get to evaluate their team-mates on this component during the semester.

5.6 System Completion

Students are graded on a 3-point scale based on completion of the system. The criterion for system completion consists of three parts: implementation of at least 90% of requirements specified, have an appealing and friendly user interface, and no major (severity 1 and 2) defects remain after system testing has completed. Towards the end of the semester, an acceptance test is also performed to verify the system satisfy all the specified requirements.

5.7 Mid-Term Examinations

There is always the tendency for some students to rely on other people in their team to carry the workload. Mid term examinations therefore constitute an important component of the course. The author has traditionally given four (4) midterm examinations. These exams have been used to verify understanding of concepts during the project lifecycle.

6. STUDENT REACTION

This semester, a course evaluation was conducted to get a sense of the qualitative effect of concepts learnt in such a course. Based on feedback gathered from 13 students, results of students' perceived ability are shown in Table 3. Although only anecdotal at this time, the numbers do show noticeable gain in skills while working within a team environment. The author plans to collect more comprehensive data in the future.

Table 3. Course Evaluation Survey

Knowledge of	**Start of Semester**	**End of Semester**
Project Teamwork	6.2	9.4
Software Development Lifecycle	5.0	8.3
Project Management	3.8	7.5
Risk Management	4.2	7.2
System Analysis	3.4	7.3
Software Testing	4.3	8.1

Additionally, several open ended questions were included in the survey. Two of them were:

1. What do you think of the team selection process used in the course?

2. What is the single most important thing you learned about team projects?

On question 1, 8/13 students thought it was a reasonable method to form teams. Some comments were that they rather pick who they want to work with, it was similar to what an employer would do, and it allowed people that get along well to be in the same team. Only 3/13 students did not like the team selection process used.

On question 2, 6/13 students included the word "communication" in their responses and stated that it was essential to a project's success. Also, 3/13 said that the contribution of each team member was crucial to its success.

6. CONCLUSION

In this paper, the author described his experiences in teaching team project courses over a six year period. Teamwork skills are emphasized during the course, providing students with plenty of opportunities to improve on their interpersonal skills as well as demonstration of responsibilities at various stages of system development. Objective evaluation is an important component for team projects and has been discussed. The evaluation criteria for grading students in such a course must be made explicit to students. Success rate of projects during the six year period are provided. A recent course evaluation survey to measure student development was also included. Student feedback has been positive regarding skill learned. The author is confident that future team projects can learn from these experiences.

REFERENCES

Hogan, J., Thomas R., Developing the Software Engineering Team, Proceedings of the 7th Australasian conference on Computing Education, Newcastle, New South Wales, Australia, Vol. 42, pp203-210, 2005

Lancor, L., Collaboration Tools in a One-Semester Software Engineering course: What worked? What didn't?, Consortium for Computing Sciences in Colleges, Northeastern Region, April 2008

LeJeune, N., A Real-World Simulation Technique for Forming Software Development Teams in a Capstone Course, The Journal of Computing Sciences in Colleges, pp247-253, Vol. 24, Number 1, October 2008

Project collaboration using BaseCamp, http://www.basecamphq.com/, last accessed 5/12/2009

Scott, T. J. , et al, Team Dynamics in Student Programming Projects, ACM SIGCSE Bulletin, Vol. 26, Issue 1, 111-115, 1994

Tan, J., Jones, M., "A Case Study of Classroom Experiences with Client-Based Team Projects", CCSC-Northeastern Conference, Staten Island, NY, April 2009

Whitehead, J., Collaboration in Software Engineering: A Roadmap, International Conference on Software Engineering, pp214-22, 2007

Wilkins, D., Lawhead, P., Evaluating individuals in team projects, ACM SIGCSE Bulletin, Volume 32, Issue 1, pp172-175, 2000

INTRODUCTION TO CRYPTOGRAPHY*

TUTORIAL PRESENTATION

Seth D. Bergmann
Rowan University

ABSTRACT

This tutorial is designed for computer scientists who have no prior experience with cryptography. Terminology and concepts related to computer cryptography will be described. Specific topics include private key cryptosystems, public key cryptosystems, authenticity, digital signatures, and certificates. Simplified algorithms which serve as examples of the above will be presented. Attendees should be familiar with the exclusive OR operation, the integer mod (i.e., %) operation, and the definition of prime numbers.

COOPERATIVE LEARNING FOR CS1 AND BEYOND:

MAKING IT WORK FOR YOU*

CONFERENCE WORKSHOP

Leland Beck
San Diego State University

Alexander Chizhik
San Diego State University

Cooperative learning is a well-known instructional strategy with many significant benefits for students. This workshop will help you make the most effective use of cooperative learning in your courses. We will focus on course planning, classroom management, and other concerns brought up by participants, combining a theoretical framework with actual experiences from the classroom. The cooperative learning principles and workshop activities will be applicable throughout the curriculum. Workshop participants will also receive a complete set of class-tested cooperative learning activities that have raised test scores in CS1 by 25% at our institution. No prior experience with cooperative learning is necessary.

THE ANIMATED DATABASE COURSEWARE (ADbC)*

CONFERENCE WORKSHOP

Mario Guimaraes
Meg Murray
Kennesaw State University

This workshop demonstrates a set of software animations, called Animated Database Courseware (ADbC), which are designed to support the teaching of database concepts. Areas covered include database design, SQL, transactions and database security. ADbC is freely available from the ADbC site (http://adbc.kennesaw.edu/). The animations are not tailored to any specific product or textbook nor are they intended to substitute for them. Instead, they provide a means to facilitate student learning resulting in an opportunity to include more depth or breadth to the concepts covered in a database course. The workshop will explore different ways the software may be incorporated into the classroom environment

TURNING A 14 WEEK NON-MAJOR CLASS INTO A 7 WEEK FAST FORWARD CLASS*

CONFERENCE WORKSHOP

Barbara Zimmerman
Villanova University

Challenges are faced when a full semester undergraduate non-CS major course is taught as a 7 week fast forward course. The format was downsized to one weekly 2.5 hour night session. Changing from traditional undergraduate students to adult students meant an additional goal of keeping the students interested in the materials after they have worked a full day. What worked or did not and what lessons were drawn are to be discussed. Instructors who teach or will teach adults will gain from attending this workshop. Together we will explore the challenges and solutions of condensing a course and teaching the adult learner.

POSTER SESSION

FACULTY POSTERS

Toward a Source Code Comment Mentoring System
Peter J. DePasquale, The College of New Jersey

Using Virtualization in the Computer Lab to Solve Sticky Problems
William Thomas, Juniata College

Revised Content and Format of CS Associate-Level Curricular Guidelines
Elizabeth K. Hawthorne, Union County College
Robert D. Campbell, CUNY Graduate Center
Karl J. Klee, Alfred State College
Anita M. Wright, Camden County College

Blogging: A Teaching Technique for the Twenty-First Century
Barbara H. Zimmerman and Najib Nadi, Villanova University

PALMS for CS1: Building Problem-Solving through Animated Learning Modules
Jeffrey A. Stone and Tricia Clark, Pennsylvania State University

STUDENT POSTERS

A Multithreaded Satisfiability Solver with Load Balancing
Bethany R. Branco and Andrea F. Lobo, Rowan University

INDEX OF AUTHORS

Bailie, F 53
Baliga, G 60
Barland, I 47
Barnes, T 5
Beck, L 181
Bergmann, S 180
Bhattacharya, S 120
Carl, S 40
Chizhik, A 181
Coppola, J 10
Cortina, T 6
Czejdo, B 120
Dougherty, J 9
Falkner, N 7
Feather-Gannon, S 10
Goelman, D 1
Griffin, J 9
Guimaraes, M 182
Gehlot, V 110
Healy, C 143
Henriksen, P 82
Hill, J 10
Ionescu, A 94
Jones, C 18
Kay, J 128
Kitlan, D 156
Klappholz, D 8
Kline, R 10
Kölling, M 82, 117
Lawler, J 10
Lewis, J 1
Li, P 11
Lindell, S 24
Liu, J 101
McCall, D 82
McGuire, T 118
Meinke, J viii
Michalewicz, Z 7
Miles, J 32
Mosley, P 10
Murray, K 53
Murray, M 182
Noonan, R 54
Petrovic, S 53
Pirmann, T 9
Powell, R 9
Raymond, E 5
Robinson, J 60
Ruth, M 94
Scorce, R 75
Sooriamurthi, R 7, 67
Stewart-Gardiner, C 164
Stone, J 156
Tan, J 172
Tashakkori, R 32
Taylor, A 10
van Delden, S 134
Ward, R 149
Way, T 5, 110
Weiss, L 60
Wellington, C 149
Whitfield, D 53
Zimmerman, B 183

JCSC Volume 25 Number 3 January 2010

Muhlenberg College
2400 Chew Street
Allentown, PA 18104-5586