The Journal of Computing Sciences in Colleges

The Consortium for
Computing Sciences in Colleges

Volume 34, Number 3 **January 2019**

The Journal of Computing Sciences in Colleges

Papers of the 34th Annual CCSC Eastern Conference

October 19th-20th, 2018
Marymount University
Arlington, VA

Baochuan Lu, Editor
Southwest Baptist University

John Meinke, Associate Editor
UMUC Europe, Retired

Susan T. Dean, Associate Editor
UMUC Europe, Retired

Steven Kreutzer, Contributing Editor
Bloomfield College

Volume 34, Number 3 **January 2019**

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 Susan Dean, CCSC Membership Secretary, 89 Stockton Ave, Walton, NY 13856.

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

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

Jeff Lehman, President (2020), (260)359-4209, jlehman@huntington.edu, Mathematics and Computer Science Department, Huntington University, 2303 College Avenue, Huntington, IN 46750.

Karina Assiter, Vice President (2020), (802)387-7112, karinaassiter@landmark.edu.

Baochuan Lu, Publications Chair (2021), (417)328-1676, blu@sbuniv.edu, Southwest Baptist University - Department of Computer and Information Sciences, 1600 University Ave., Bolivar, MO 65613.

Brian Hare, Treasurer (2020), (816)235-2362, hareb@umkc.edu, University of Missouri-Kansas City, School of Computing & Engineering, 450E Flarsheim Hall, 5110 Rockhill Rd., Kansas City MO 64110.

Susan Dean, Membership Secretary (2019), Associate Treasurer, (607)865-4017, Associate Editor, susandean@frontier.com, UMUC Europe Ret, US Post: 89 Stockton Ave., Walton, NY 13856.

Judy Mullins, Central Plains Representative (2020), Associate Treasurer, (816)390-4386, mullinsj@umkc.edu, School of Computing and Engineering, 5110 Rockhill Road, 546 Flarsheim Hall, University of Missouri - Kansas City, Kansas City, MO 64110.

John Wright, Eastern Representative (2020), (814)641-3592, wrightj@juniata.edu, Juniata College, 1700 Moore Street, Brumbaugh Academic Center, Huntingdon, PA 16652.

David R. Naugler, Midsouth Representative(2019), (573) 651-2787, dnaugler@semo.edu, 5293 Green Hills Drive, Brownsburg IN 46112.

Lawrence D'Antonio, Northeastern Representative (2019), (201)684-7714, ldant@ramapo.edu, Computer Science Department, Ramapo College of New Jersey, Mahwah, NJ 07430.

Cathy Bareiss, Midwest Representative (2020), cbareiss@olivet.edu, Olivet Nazarene University, Bourbonnais, IL 60914.

Brent Wilson, Northwestern Representative (2021), (503)554-2722, bwilson@georgefox.edu, George Fox University, 414 N. Meridian St, Newberg, OR 97132.

Mohamed Lotfy, Rocky Mountain Representative (2019), Information Technology Department, College of Computer & Information Sciences, Regis University, Denver, CO 80221.

Tina Johnson, South Central Representative (2021), (940)397-6201, tina.johnson@mwsu.edu, Dept. of Computer Science, Midwestern State University, 3410 Taft Boulevard, Wichita Falls, TX 76308-2099.

Kevin Treu, Southeastern Representative (2021), (864)294-3220, kevin.treu@furman.edu, Furman University, Dept of Computer Science, Greenville, SC 29613.

Bryan Dixon, Southwestern Representative (2020), (530)898-4864, bcdixon@csuchico.edu, Computer Science Department, California State University, Chico, Chico, CA 95929-0410.

Serving the CCSC: These members are serving in positions as indicated:

Brian Snider, Associate Membership Secretary, (503)554-2778, bsnider@georgefox.edu, George Fox University, 414 N. Meridian St, Newberg, OR 97132.

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

John Meinke, Associate Editor, meinkej@acm.org, UMUC Europe Ret, German Post: Werderstr 8, D-68723 Oftersheim, Germany, ph 011-49-6202-5777916.

Shereen Khoja, Comptroller, (503)352-2008, shereen@pacificu.edu, MSC 2615, Pacific University, Forest Grove, OR 97116.
Elizabeth Adams, National Partners Chair, adamses@jmu.edu, James Madison University, 11520 Lockhart Place, Silver Spring, MD 20902.
Megan Thomas, Membership System Administrator, (209)667-3584, mthomas@cs.csustan.edu, Dept. of Computer Science, CSU Stanislaus, One University Circle, Turlock, CA 95382.
Deborah Hwang, Webmaster, (812)488-2193, hwang@evansville.edu, Electrical Engr. & Computer Science, University of Evansville, 1800 Lincoln Ave., Evansville, IN 47722.

CCSC National Partners

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

Platinum Partner

Turingscraft
Google for Education
GitHub
NSF – National Science Foundation

Silver Partners

zyBooks

Bronze Partners

National Center for Women and Information Technology
Teradata
Mercury Learning and Information

Foreword

Hello, world! Here is the first journal issue I put together for CCSC. It has been a pleasure working with John Meinke to learn about CCSC publication process. John has worked tirelessly for decades as the journal editor and I will do my best to continue the legacy he had of dedicated service to the community.

I am implementing two major changes in the publication process. First, the journals will be typeset in LaTeX(as this current issue). LaTeX allows authors to typeset their final "camera-ready" copies, which are, then, compiled into a journal. Therefore, I would prefer that authors submit their final manuscripts in LaTeX. Word docs are acceptable if an author doesn't want to learn/use LaTeX, in which case a regional editor or I will need to convert the documents into LaTeX. Here is a repository that contains instructions, templates, and links to learning resources: `https://github.com/lubaochuan/ccsc-editor` The second change is to use on-demand printing. You can still get a hard-copy of the journal through your regional conference or by ordering a copy yourself on Amazon. This approach reduces paper waste and allows me to fix errors in a journal after it is published.

I am very excited about my work, but I cannot produce quality journals without authors contributing to the content. Please continue writing papers to share your ideas and experiences. The CCSC community depends on you to stay active and relevant helping all of us perfect our craft.

Baochuan Lu
Southwest Baptist University
CCSC Publications Chair

Welcome to the 34th Annual (2018) CCSC Eastern Conference

On behalf of the 2018 CCSC Eastern Conference Committee, welcome to the 2018 CCSC Eastern Conference and the new Ballston Center at Marymount University. We hope that everyone gets a chance to explore the greater DC area and see everything Arlington has to offer. We are looking forward to a great conference this year and are excited about sessions available this for students and faculty. Whether new or returning to CCSC, we think you'll find new and exciting sessions for your field of interest.

In additional to 4 faculty paper sessions this year, we will have 2 student sessions, and a poster session. This will be our first year trying student presentations. We hope this will be a quality experience for the students to present and get feedback on their work. We look forward to conferences in a few years where these students might return as faculty colleagues. The programming competition this year which will feature both college and high school teams from around the eastern region which looks to be quite competitive this year. We will have two wonderful workshops brought to us from the NSF. One on CyberPaths which aims for diversification and broadening of the STEM talent pipeline in cybersecurity and one talk on applying and obtaining grants through the NSF as smaller institutions.

We received 23 excellent faculty papers for double blind peer review and we were able to accept 12 for presentation at this year's conference. It was a very competitive year with an acceptance rate of 52%. Topics this year ranged the computing gamut including areas from natural language processing, Computer Science Education, robotics, to email security. The selected papers will be included in the Journal of Computing Sciences in Colleges.

This year we will have 4 panels and tutorials for the following: Assignments to Promote Diversity and Accessibility, Creating a culture and environment for Active Learning Success, Engaging HBCU Faculty in Project-Based Learning In Silicon Valley, and Using PLCC to implement Java interpreters in a Programming Languages course. We hope everyone is able to attend a few of these over the next 2 days.

This conference could not be done without the volunteers on the conference committee. I am grateful for their help and experience and continued support of this conference. As you mingle around the Ballston building please take time to give an extra thanks to Steve Kreutzer, Helen Wei, Vincent Cicirello, Lauren Vantalia, Yanxia Jia, Kathy Marcropol, Karen Paullet, Adnan Chawdhry, Dave Hovemeyer, John Wright, and Donna Schaeffer.

We hope you enjoy the conference and your time in Arlington. Please let us know how we did and how we could make it better in the future. If you

are interested in helping out in future conferences by hosting, participating in the committee or as a paper reviewer, please contact any committee member. I hope to see you all at the banquet and around DC this weekend!

Nathan Green
Marymount University
Conference Chair

CCSC:Eastern Region Steering Committee

Elizabeth Adams ..... James Madison University
Steven Andrianoff ..... St. Bonaventure University
Karen Anewalt ..... University of Mary Washington
George Benjamin ..... Muhlenberg College
Elizabeth Chang ..... Hood College
Vincent Cicirello ..... Stockton University
Michael Flinn ..... Frostburg State University
Sister Jane Fritz ..... St. Joseph's College
David Hovemeyer ..... York College of Pennsylvania
John Meinke ..... University of Maryland University College
Donna Schaeffer ..... Marymount University
Jennifer Polack-Wahl ..... University of Mary Washington
John Wright ..... Juniata College

CCSC:Eastern 2018 Conference Committee

Nathan Green, Chair, Registration ..... Marymount University
Steve Kreutzer, Papers ..... Bloomfield College
Helen Wei, Papers ..... Stockton University
Vincent Cicirello, Panels, Workshops, Tutorials ..... Stockton University
Lauren Vantalia, Nifty Ideas & Lightning Talks ..... Marymount University
Yanxia Jia, Posters, ... Arcadia University
Kathy Marcropol, Posters, ... Arcadia University
Karen Paullet, Student Research/Mentors ..... Robert Morris University
Adnan Chawdhry, Student Research/Mentors ..... California University of Pennsylvania
Dave Hovemeyer, Programming Contest ..... York College of Pennsylvania
John Wright, Regional Board Representative ..... Juniata College
Donna Schaeffer, Regional Treasurer ..... Marymount University
Nathan Green, Web Site ..... Marymount University
John Wright, Web Site ..... Juniata College

Reviewers — CCSC:Eastern Conference

Imad Al Saeed ..... Saint Xavier University, Chicago, IL
Rana Alabdan ..... Robert Morris University, Moon, PA
Faleh Alshameri ..... Marymount University, Arlington, VA
Ian Barland ..... Radford University, Radford, VA
George Benjamin ..... Muhlenberg College, Allentown, PA
Seth Bergmann ..... Rowan University, Glassboro, NJ
Karla Carter ..... Bellevue University, Bellevue, NE
Mónica Costa ..... Polytechnic Institute of Castelo Branco, Portugal
Lawrence D'Antonio ..... Ramapo College, Mahwah, NJ
Joshua Eckroth ..... Stetson University, DeLand, FL
Ian Finlayson ..... The University of Mary Washington, Fredericksburg, VA
Michael Flinn ..... Frostburg State University, Frostburg, MD
Reva Freedman ..... Northern Illinois University, DeKalb, IL
Alessio Gaspar ..... University of South Florida, Tampa, FL
Suvineetha Herath ..... Carl Sandburg College, Galesburg, IL
Ivaylo Ilinkin ..... Gettysburg College, Gettysburg, PA
Lindsay Jamieson ..... St. Mary's College of Maryland, St. Mary's City, MD
Yanxia Jia ..... Arcadia University, Glenside, PA
Steve Kreutzer ..... Bloomfield College, Bloomfield, NJ
Juan Jenny Li ..... Kean University, Union, NJ
Robert Marmorstein ..... Longwood University, Farmville, VA
Alex Mbaziira ..... Marymount University, Arlington, VA
Robert McCloskey ..... University of Scranton, Scranton, PA
John McManus ..... Randolph-Macon College, Ashland, VA
Patricia Morreale ..... Kean University, Union, NJ
James Moscola ..... York College of Pennsylvania, York, PA
Patrick Olson ..... National University, Campbell, CO
Karen Paullet ..... Robert Morris University, Moon, PA
Donna Schaeffer ..... Marymount University, Arlington, VA
Edwin Torres ..... Monmouth University, West Long Branch, NJ
Duo (Helen) Wei ..... Stockton University, Galloway, NJ

NSF Grant Workshop*

Mark A. Pauley
Program Director, Division of Undergraduate Education
Directorate for Education and Human Resources
National Science Foundation

This workshop is intended for faculty and supporting staff at universities and community and technical colleges who are interested in obtaining support from the National Science Foundation (NSF) for curricular and programmatic initiatives but who are unfamiliar with current funding opportunities and/or lack grant-writing experience. The workshop will provide an overview of the programs in NSF's Directorate for Education & Human Resources that focus on undergraduate education and cover the basic elements of a good proposal and strategies that can be used to effectively formulate and communicate ideas to reviewers and NSF program officers. Also discussed will be the mechanics of submitting a proposal, the review process from submission to award or decline, merit review criteria, and how to volunteer to become a reviewer.

CyberPaths*

Conference Workshop

Xenia Mountrouidou
Department of Computer Science
College of Charleston
mountrouidoux@cofc.edu

The goal of the project CyberPaths is the diversification and broadening of the STEM talent pipeline in cybersecurity in predominantly undergraduate and liberal arts schools. This is achieved by the creation of a curriculum that accommodates students of different levels of computer literacy with focus on experiential learning. This project mitigates the challenges undergraduate institutions currently face in the cybersecurity area, for example, a tight computer science curriculum and the inability to support the expensive infrastructure required for cybersecurity education. To address these challenges, first, we attract a diverse population of students by introducing cybersecurity topics through multiple paths of study and engagement. Students will be introduced to cybersecurity concepts through stand-alone course modules and laboratory exercises injected in general education liberal arts courses. Interested students can study further by taking two cybersecurity focused courses and cybersecurity capstone projects created by this project. Second, we use the Global Environment for Network Innovation (GENI) infrastructure in the development of hands-on labs and the capstone project assignments. GENI offers an affordable cloud solution to undergraduate institutions that lack the infrastructure to support high overhead computer labs. In this talk, I will present the CyberPaths project and briefly introduce the GENI labs and general education modules that we have developed. Then we will complete a couple of short GENI labs, starting from "Hello GENI" and moving to a simple "Denial of Service lab".

Using PLCC to Implement Java Interpreters in a Programming Languages Course*

Conference Workshop

Timothy Fossum
Computer Science Department
Rochester Institute of Technology
Canandaigua, NY 14424
tvf@rit.edu

This is a hands-on workshop that introduces participants to the PLCC compiler - compiler tool set to build Java-based interpreters for languages in an upper-level programming Languages course. Input to PLCC is a single specification file defining a language's lexical, syntax, and semantic structure. PLCC takes this specification and generates stand-alone Java source files that implement an interpreter for the language, including a front-end scanner, a recursive descent parser, and a read-eval-print loop. The specification file defines language tokens using simple regular expressions, language syntax using straight-forward BNF rules, and language semantics in Java classes corresponding to each of the BNF rules.

This workshop is designed for Computer Science educators who [want to] teach upper-level Programming Languages or Compilers courses in a CS program. Workshop materials will be made available in electronic format: on-line in advance of the workshop and on a USB stick at the beginning of the workshop. Participants are encouraged to install PLCC and related files used in the Workshop on a laptop prior to attending the Workshop. This material will be made available through Google Drive or from the Presenter by email in Zip format. These materials will consist of the PLCC tool set, PDF workshop slides, worked-out examples of languages, and a sequence of examples and problems to be carried out by participants at the workshop itself. Participants will work through these examples and problems that illustrate, in turn, each of the essential analysis parts of a language specification: lexical, syntax, and semantics.

Assignments to Promote Diversity and Accessibility*

Conference Tutorial

Jessica Zeitz and Karen Anewalt
Computer Science
University of Mary Washington
1301 College Ave
{jzeitz,anewalt}@umw.edu

Description

Researchers have long recognized the existence of the racial and gender gap in the technology field. Increasingly, the discipline is also acknowledging the need for inclusive access to computer science education. We will present a collection of assignments related to diversity, inclusion, and accessibility that all computer science instructors can reference. The assignments will be applicable to many standard computer science courses such as introduction to programming, data structures, object-oriented design, databases, and software engineering. The materials will include recommendations for the courses where assignments would best fit and we will discuss how to incorporate the ideas into commonly required computer science courses such as introduction to programming, data structures, and software engineering.

We believe that raising awareness is an essential step toward creating a more diverse and inclusive environment within the technology field. By positioning assignments throughout the curriculum we encourage students to confront issues of diversity, inclusion, and accessibility within various contexts, emphasizing that these issues are pervasive in the computing discipline.

In this tutorial, we will provide access to an online repository of assignments related to diversity, inclusion, and accessibility. We will highlight six of our assignments which can be used for classroom discussion or oral presentation, writing assignments, or programming and problem-solving assignments. We will also provide tips for leading classroom discussions about these difficult

issues and encourage participants to share experiences from their past courses. Participants will also be invited to share assignments and experiences within the online repository that we are compiling.

The intended audience includes faculty interested in raising student awareness of issues in diversity, inclusion, and accessibility within the technology field and promoting discussion on related issues. Participants will receive copies of at least 10 assignments that can be integrated into common computing classes (intro to programming, data structures, software engineering, ethics).

Presenters

Jessica Zeitz is an Assistant Professor of Computer Science at the University of Mary Washington. She teaches courses in programming, databases, object-oriented design, and human-computer interaction. She has experience with diversity efforts such as panels and community outreach. She was involved in a large-scale event introducing computer science to middle school girls for four years. Her research focuses on human-computer interaction and visual analytics and her work has been published in the American Educational Research Association, the Consortium of Computing Sciences in Colleges, the IEEE Transactions on Intelligent Systems Journal and the IEEE Transactions on Learning Technologies Journal.

Karen Anewalt is a Professor of Computer Science at the University of Mary Washington. She teaches course in programing, Data Structures, Object-oriented Design, Networks, Web Programming, E-commerce, Ethics, and Software Engineering and has 18 years of teaching experience. Her professional work focuses on computer science education and pedagogy and has been published in the Journal of Computing Sciences in Colleges, SigCSE, and Frontiers in Education.

Creating a Culture and Environment for Active Learning Success*

Conference Tutorial

Jesse Eickholt[1], Patrick Seeling[1], Lisa Gandy[1], Quentrese Cole[2], and Matthew Johnson[2]
[1]Department of Computer Science
[2]Department of Educational Leadership
Central Michigan University
Mount Pleasant, MI 48859
{eickh1jl,seeli1p,gandy1l,cole1qt,johns9m}@cmich.edu

Active learning and active learning classrooms have received renewed interest in recent years. Active learning can be broadly considered as any activity that involves the student in the learning process, requiring that students think about what they are doing[3]. While the positive effects of active learning on students' academic performance and perceived experience are widely reported[1], there are a number of barriers that have limited greater adoption. These barriers include limits on faculty time, institutional resources to create active learning spaces, concerns about student resistance and resentment, instructional preferences, and misunderstandings about what constitutes active learning and related benefits[2][4].

Participants in this tutorial will be presented with many options to overcome barriers to active learning. These options will span the spectrum in terms of scope, with some focused on what individual instructors can do and others looking at sparking larger departmental or institutional changes. Ample time will be dedicated to discussion of individual concerns, challenges and success stories. A variety of resources to support active learning will be presented, with an emphasis placed on economy-based active learning technology and tools. We will provide participants with an overview of frameworks and tools to convert existing classrooms into active learning classrooms that support collaboration and distribution of common artifacts. Specific examples we employ include artifacts generated by computer science students (such as programs).

The development of these tools and this tutorial was supported by a grant from the National Science Foundation (IUSE 1608043).

References

[1] Freeman, S., Eddy, S., McDonough, M., Smith, M., Okoroafor, N., Jordt, H., and Wenderoth, M. Active learning increases student performance in science, engineering, and mathematics. In *Proceedings of the National Academy of Sciences*.

[2] Michael, J. Faculty perceptions about barriers to active learning. *College Teaching 55*, 2 (2007), 42–47.

[3] Prince, M. Does active learning work? a review of the research. *Journal of Engineering Education* (2004), 223–231.

[4] Reid, P. Categories for barriers to adoption of instructional technologies. *Education and Information Technologies 19*, 2 (June 2014), 383–407.

Student Perceptions of Computing and Computing Majors*

Jeffrey A. Stone
Information Sciences and Technology
Pennsylvania State University
stonej@psu.edu

Abstract

The growth in Computer Science (CS) program enrollments over the past decade, combined with an increased recognition of the importance of computing skills in the global economy, means that educational institutions have an opportunity to attract an increasingly diverse set of students. Despite robust enrollments, students may be entering university with misconceptions about computing, computing majors, and computing career prospects, potentially limiting long-term growth in computing programs. This paper updates the author's previously published work by describing a study to elicit perceptions of computing and computing majors among a set of incoming students. The results suggest that while students have positive feelings towards computing, computing majors, and computing-related careers, significant differences exist for socioeconomic and gender groups.

1 Introduction

There has been a significant growth in undergraduate Computer Science (CS) program enrollments since 2006, with some estimates as high as almost 300% growth [10]. The increased recognition of computing and computational thinking as important skills [4], coupled with the needs of a growing job market, means that both primary and secondary level educational institutions have the opportunity and responsibility to construct curricula and recruitment plans

that attract a diverse set of students. At the university level, incoming students are often perceived as "Digital Natives" who have been widely exposed to computing technology and are proficient in its use [8]. Access to and use of computing technology is not uniform, however, and can differ based on socioeconomic status, gender, and other factors [9]; such access and use can influence self-efficacy and academic success ([5],[6]). As a result, initiatives to recruit a talented and diverse computing workforce are ongoing, ranging from primary school curricula development to policy advocacy ([7], [11]).

While the positive growth in CS enrollments is encouraging, students may be entering university with misconceptions about the nature of computing, related majors, and related career prospects. These misconceptions are often thought to be formed from a variety of sociological and experiential factors, including K-12 experiences, demographic factors (e.g. gender, socioeconomic status), societal norms, and access to information technology. For example, prior research has suggested that students' understanding of CS is often limited, perhaps due to insufficient career counseling and applications-centric high school computing curricula ([2],[3]). Reported misconceptions of computing majors and careers have included social isolation, challenging curricula, and limited career options, among others ([1],[2],[13]). These misconceptions often lead to negative attitudes towards and a lack of awareness of computing majors [4], potentially limiting the ability of university computing programs to attract and retain students.

The goal of this study was to explore perceptions of computing and computing majors among a set of incoming university students by updating the author's previous study [12]. The conceptual model for the study (Figure 1) recognizes the relationship between a variety of social and academic influences – e.g. Information and Communications Technology (ICT) access and use, peers, parents, educators, K-12 education experience – and student perceptions of computing and computing majors, filtered through the lens of group differences (gender, race/ethnicity, and socioeconomic status). The research questions for this study were: *What factors most influence students' perceptions of computing and computing majors?* and *What group differences exist in students' perceptions of computing and computing majors?*

2 Methodology

An electronic survey was administered to a set of incoming students at one campus of a public research university during the 2017 summer orientation sessions. The survey was constructed based on prior research by the author [12] and a more recent literature review. Besides basic demographic questions, participants were primarily asked to respond to a series of Likert-style statements,

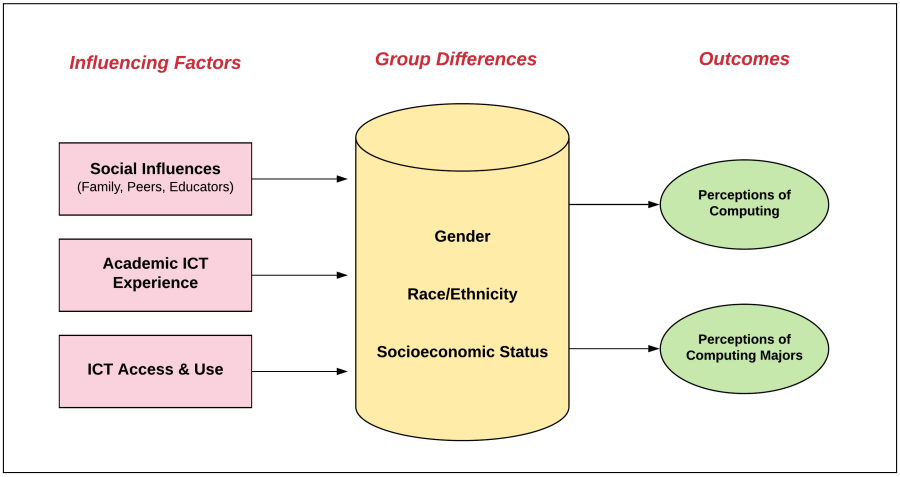


Figure 1: Conceptual Model

using a five-level scale (1=Strongly Agree, 2=Agree, 3=Neutral [Undecided], 4=Disagree, and 5=Strongly Disagree). All research protocols and instruments were approved by the Penn State University Office of Research Protections.

3 Survey Results

A total of 161 students completed at least one question on the survey (78.54%, N=205). More than half the participants were female (55.33%, n=150). A majority of participants were 18-30 years old (97.33%, n=148) and White/-Caucasian (76.00%, n=150). A majority of participants identified as commuters (62.00%, n=150) and were not considered first-generation college students (73.33%, n=150). In terms of community, 26.00% (n=150) came from a rural community, 31.33% from suburbia, 40.00% from an urban area (city/-town), and 2.67% were international students. Household income served as a proxy measure for socioeconomic status; a plurality of participants (45.83%, n=144) reported a household income of between $50,000 and $99,999, while a lesser amount reported household income less than $50,000 (38.89%). The following sections describe the survey results in light of the study constructs. Group differences are reported only when statistically significant.

3.1 ICT Access, Use, and Academic Experience

Participants most often reported using the Internet for more than 10 hours each week (60.00%, n=160), and most reported having Internet access at home (85.71%, n=161), at school (66.46%), and via mobile devices (90.68%). Most participants reported access to a computer in the home (86.34%, n=161) and at school (61.49%). Access to tablet computers (e.g. Kindle, Galaxy, iPad) was uncommon; only 35.40% reported having access to a tablet. Most participants reported using computers in high school (93.79%, n=161) and middle school (86.96%). The most common high school computing course reported by participants was a Microsoft Office-style/basic computer skills course (72.67%, n=161). Other reported high school computing courses included graphic arts, digital photography, or digital video editing (32.92%) and digital publishing (5.59%). Only 13.04% reported taking a computer programming course.

3.2 Perceptions of Computing

Most respondents agreed or strongly agreed with the statement *Computer Technology improves our society* (69.81%, n=159). The importance of computing to respondents' everyday lives was evident; a majority agreed or strongly agreed with the statement *Computer Technology plays an important role in my everyday life* (77.36%, n=159) and with the statement *Computer Technology plays an important role in the everyday life of my friends and family* (84.28%, n=159). Participants were also asked about the relationship of computing to their future careers. A majority of respondents agreed or strongly agreed with the statements *Employers will expect me to have Computer Technology skills* (77.22%, n=158) and *My future career will require me to use Computer Technology on a regular basis* (67.30%, n=159). Respondents overwhelmingly agreed or strongly agreed with the statement *Computer Technology jobs will increase in the next 10-20 years* (86.79%, n=159). See Table 1.

Table 1: Perceptions of Computing Response Summary

Statement	n	Mean	Median	SD
Computing improves our society	159	2.09	2.00	0.906
Employers expect computing skills	158	2.02	2.00	0.802
My career will require computer use	159	2.14	2.00	0.938
Computing jobs will increase	159	1.68	2.00	0.774
Computing important in my life	159	1.94	2.00	0.859
Computing important for friends/family	159	1.88	2.00	0.669

Participants were asked to respond to the statement, *How would you de-*

scribe your feelings about Computer Technology? with a five-level scale (1=Very Positive, 2=Positive, 3=Neutral (Undecided), 4=Negative, 5=Very Negative). A majority responded with very positive or positive feelings (71.07%, n=159). Significant differences between genders were found for this statement (X^2=9.73, df=4, $p < 0.05$). Females had a higher mean response (2.33 vs. 1.90), meaning females were less likely to report positive feelings towards computer technology.

3.3 Perceptions of Computing Majors

A series of statements were used to determine the perceived value of computing majors and the prevalence of traditional stereotypes about computing. Over half the respondents agreed or strongly agreed with the statement Computing majors are difficult majors (51.57%, n=159), but recognition of the strong computing career prospects were seen in responses to the statement *Students who are Computing majors are likely to have successful careers.* A majority (63.52%, n=159) agreed/strongly agreed with the statement. A note of positivity about computing majors (and diminishing stereotypes) was found in the percentage of disagree/strongly disagree responses to the statements *Students who are Computing majors are more likely to be shy and non-social* (49.05%, n=159) and *Computing majors and careers are primarily for male students* (66.04%, n=159). See Table 2.

Table 2: Perceptions of Computing Majors Response Summary

Statement	n	Mean	Median	SD
Computing majors are difficult majors	159	2.50	2.00	0.745
Students likely to be shy and non-social	159	3.45	3.00	0.905
Students likely to have successful careers	159	2.26	2.00	0.750
Computing primarily for male students	159	3.76	4.00	0.984

3.4 Social Influences

Two statements were used to assess the influence of parents and educators on the perceived importance of computing skills. A plurality of respondents agreed or strongly agreed (46.88%, n=160) with the statement *My high school teachers and/or guidance counselors stressed the importance of Computer Technology Skills.* However, a plurality of respondents were undecided about the statement *My parent(s) stressed the importance of Computer Technology Skills* (40.51%, n=158). Significant differences between the genders were found for this statement (X^2=10.82, df=8, $p < 0.05$). Females were less likely to agree with the statement (M = 3.12 vs. M = 2.72). One-way ANOVA also found

significant differences between responses to this statement and the reported household income ($F[2, 139] = 3.34$, $p < 0.05$). Post-hoc Tukey analysis indicated that those respondents reporting a household income of less than $50,000 ($M = 3.16$, $p < 0.05$) were less likely to agree with the statement than those respondents reporting income in the [$50,000-$99,999] range ($M = 2.72$).

Three other statements were designed to assess the influence of parents and educators on students' awareness of computing majors. A plurality of respondents agreed or strongly agreed with the statement *My high school teachers and/or guidance counselors provided me with information on college-level Computing majors and careers* (41.25%, n=160). A plurality of respondents disagreed or strongly disagreed (38.99%, n=159) with the statement *My parent(s) provided me with information on college-level Computing majors and careers.* One-way ANOVA found significant differences between responses to this statement and the reported household income ($F[2, 140] = 5.79$, $p < 0.01$). Post-hoc Tukey analysis indicated that those respondents reporting a household income of less than $50,000 ($M = 3.43$, $p < 0.01$) were less likely to agree with the statement than those reporting income between $50,000 and $99,999 ($M = 2.80$). Finally, respondents were asked *Who among the following people is the primary influence on your choice of major?* The majority reported that their choice of major was self-directed (70.44%, n=159), followed by family (17.61%), friends (3.77%), unknown (4.40%), and teachers (1.89%). See Table 3.

Table 3: Social Influences Response Summary

Statement	n	Mean	Median	SD
Teachers/counselors stressed skills	160	2.61	3.00	0.991
Teachers/counselors provided information	160	2.91	3.00	1.103
Parents stressed skills	158	2.94	3.00	0.979
Parents provided information	159	3.16	3.00	1.071

4 Discussion and Conclusion

The survey results show that these participants recognize the value of computing and computing majors, though it is unknown exactly how those perceptions are developed. As expected, computer and Internet access and use was very common among the study participants. Students are connected at school, at home, and on the go, and it plays a large role in their everyday lives and the lives of friends and family. Somewhat surprisingly, this does not translate into widespread enrollment in high school computing courses beyond simple MS

Office-style applications courses. Enrollment in traditional CS-related high school courses (i.e. computer programming) was also very low.

The survey results also show that participants have a positive outlook on computing. Computing technology is seen as important to daily life, and respondents were very aware of the importance of computing skills for both their future careers and employment in general. A slight majority see computing majors as challenging; more importantly, existing stereotypes of computing students – shy, non-social, and male – are not considered reality among this population. However, the results involving parental and educator influence were surprising. While a plurality agreed that teachers and/or guidance counselors stressed the importance of computer skills, a minority of respondents indicated the same for parents. Males were much more likely to agree that parents stressed these skills. This gender disparity, coupled with the fact that females were significantly less likely to report positive feelings towards computing technology than males, raises concerns that gender-based disparities may persist. A minority of participants indicated that parents provided them with information on computing majors, while a plurality indicated that their teachers and/or guidance counselors did the same. Household income was a differentiator when it came to parental influence. Respondents with household incomes less than $50,000 were less likely than respondents in the $50,000-$99,999 range to agree that parents provided them with information on computing majors or stressed the importance of computing skills. Universities should continue to provide outreach programs to attract and retain lower income and female students, though programs which can educate parents of these potential students about computing majors and careers should also be considered.

The study described in this paper provides some positive evidence that incoming students recognize the value and importance of computing, computing majors, and computing-related careers. The results are limited to a single university, but provide a basis for further exploration of both computing-related perceptions and the factors which may influence their development. The significant growth in CS program enrollments and the bright future for computing-related jobs provides universities with an opportunity to attract (and retain) an increasingly diverse set of students. By understanding some of the perceptions students have about computing, decision-makers can make more informed decisions about program offerings, marketing approaches, and retention strategies.

5 Acknowledgements

The author wishes to thank Bryan Valentine, Director of Student Affairs and Enrollment Services at Penn State Schuylkill, and Michael Koharcheck, undergraduate student, for their assistance in this project.

References

[1] Clarke, V. A., and Teague, G. J. Characterizations of computing careers: Students and professionals disagree. *Computers & education 26*, 4 (1996), 241–246.

[2] Egan, M. A. L., and Lederman, T. The impact of impact: assessing students' perceptions after a day of computer exploration. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (2011), ACM, pp. 318–322.

[3] Galpin, V. C., and Sanders, I. D. Perceptions of computer science at a south african university. *Computers & Education 49*, 4 (2007), 1330–1356.

[4] Grover, S., Pea, R., and Cooper, S. Remedying misperceptions of computer science among middle school students. In *Proceedings of the 45th ACM technical symposium on Computer science education* (2014), ACM, pp. 343–348.

[5] Hasan, B. The influence of specific computer experiences on computer self-efficacy beliefs. *Computers in human behavior 19*, 4 (2003), 443–450.

[6] Jackson, L. A., Zhao, Y., Kolenic III, A., Fitzgerald, H. E., Harold, R., and Von Eye, A. Race, gender, and information technology use: The new digital divide. *CyberPsychology & Behavior 11*, 4 (2008), 437–442.

[7] McGill, M. M., Decker, A., and Settle, A. Undergraduate students' perceptions of the impact of pre-college computing activities on choices of major. *ACM Transactions on Computing Education (TOCE) 16*, 4 (2016), 15.

[8] Prensky, M. Digital natives, digital immigrants part 1. *On the horizon 9*, 5 (2001), 1–6.

[9] Ritzhaupt, A. D., Liu, F., Dawson, K., and Barron, A. E. Differences in student information and communication technology literacy based on socio-economic status, ethnicity, and gender: Evidence of a digital divide in florida schools. *Journal of Research on Technology in Education 45*, 4 (2013), 291–307.

[10] Roberts, E., Camp, T., Culler, D., Isbell, C., and Tims, J. Rising cs enrollments: Meeting the challenges. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (2018), ACM, pp. 539–540.

[11] Scott, A., Martin, A., McAlear, F., and Madkins, T. C. Broadening participation in computer science: Existing out-of-school initiatives and a case study. *ACM Inroads 7*, 4 (2016), 84–90.

[12] Stone, J. A., and Kitlan, D. P. Factors impacting student perceptions of computing and cis majors. *Journal of Computing Sciences in Colleges 25*, 3 (2010), 156–163.

[13] Yardi, S., and Bruckman, A. What is computing?: bridging the gap between teenagers' perceptions and graduate students' experiences. In *Proceedings of the third international workshop on Computing education research* (2007), ACM, pp. 39–50.

A Comparison of Perceptions of CS Majors and Non-Cs Majors Regarding Email Security*

Virginia Eaton, Jose Cordova, Tyler Greer, and Lon Smith
Computer Science Department
University of Louisiana Monroe
Monroe, LA 71209
{eaton,cordova,greer,lsmith}@ulm.edu

Abstract

This paper describes a study designed to measure and compare the perceptions of computer science (CS) majors with those of other majors concerning the relative safety of various forms of communication, particularly electronic mail. The results indicate that, for most media, there exist significant differences between the perceptions of CS majors and those of other majors. Similar results are obtained when comparing the responses of CS majors with those of other majors regarding the importance of considering the identity of the sender when evaluating the relative safety of incoming email messages. The results suggest the need for additional training in this area for students in other fields.

1 Introduction

There is a huge market for stolen passwords on the Internet. On February 8, 2018, National Public Radio [8] broadcast a story entitled "The Market for Stolen Passwords." The story describes a site on the dark web where passwords for hundreds of companies are for sale. Passwords are being stolen despite the best efforts of companies to alert their customers to the dangers of phishing emails. For example, on April 11, 2013, a local bank sent an email to its

customers warning them about the dangers of phishing emails. Five years later, on April 18, 2018, the bank sent a similar email to its customers detailing the dangers of phishing emails. In a similar scenario, our university computing center sent an email on March 22, 2012, warning faculty, staff, and students about the dangers of phishing emails. Six years later, on April 5, 2018, a similar email was sent to all faculty, staff, and students alerting them to the university Phishing Awareness Campaign. Yet, despite these types of efforts, people continue to succumb to phishing attacks. Why? Are some people more susceptible than others? This paper is focused on one group of people, college students, and whether some students are more susceptible than others to phishing attacks. In particular, it compares the attitudes toward email security of non-CS majors with the attitudes of CS majors.

2 Background

Downs, Holbrook, and their colleagues have published two papers looking at who falls for phishing attacks. For their first paper [4], they interviewed 20 non-expert computer users to discover their strategies for dealing with possibly suspicious emails. In addition to answering questions about email security, the participants were involved in role playing related to email security. The researchers concluded that people can manage the risks that they are most familiar with, but don't appear to extrapolate to be wary of unfamiliar risks.

Four years later, they published their second paper [7], which had 1001 respondents to an online survey. In this paper, they examined the relationship between demographics and phishing susceptibility and the effectiveness of anti-phishing educational materials. Participants completed a survey followed by two tasks. The first task was to assess their susceptibility to phishing prior to receiving one of several forms of training. After the training, participants completed a second task to assess reductions in susceptibility as well as any changes in participants' tendencies to be suspicious of legitimate emails. The researchers concluded that women are more susceptible than men and participants between the ages of 18 and 25 are more susceptible.

Dhamija and her associates [3] addressed the question of why phishing works. They tested their hypotheses in a usability study with 22 participants. They showed each participant 20 web sites in random order and asked them to determine which ones were fraudulent and why. They concluded that good phishing websites fooled 90% of the participants. They also found that 23% of the participants did not look at the address bar, status bar, or any other security indicators. The researchers concluded that neither education, age, sex, previous experience, nor hours of computer use showed a statistically significant correlation with vulnerability to phishing.

Herath and her colleagues [5] looked at security services as a coping mechanism for dealing with phishing attacks. Participants in the study were all junior level undergraduate students in a required first-year course in management information. Participants were asked to complete two surveys. A total of 186 subjects responded to the first survey. Two months later, 134 of them completed the second survey. During the two intervening months participants were trained to install and use an email authentication service. The researchers concluded that users' intention to adopt an email security service is contingent upon users' perception of risk. The users' risk perceptions were found to have a significant impact on the users' perception of the usefulness of the security tool as well as their intentions to use the security tool.

Wang and his associates [9] examined what leads to overconfidence in phishing email detection. They performed a survey experiment with 600 subjects to collect empirical data for the study. The researchers collected data relying on the professional survey service company, Qualtrics. Using Qualtrics, they collected 600 valid responses from 47 States in the US. In the experiment, each subject judged a set of randomly selected phishing emails and authentic business emails. The researchers concluded that overconfidence is quite common in phishing email detection. Their findings suggest that trainers need to recognize sources of overconfidence and help devise mechanisms to reduce such bias.

Jensen and his associates [6] added mindfulness training to the traditional rule-based training to teach individuals to identify certain cues or apply a set of rules to avoid phishing attacks. The researchers argue that mindfulness techniques are critical to detecting phishing attacks but are unaddressed in rule-based instruction. They compared rule-based and mindfulness training programs in a field study at a U.S. university that involved 355 students, faculty, and staff who were familiar with phishing attacks and received regular rule-based guidance. They found that participants who also received mindfulness training were better able to avoid phishing attacks than those who received rules-based training only.

Cordova and his associates have written two previous studies comparing CS and non-CS majors regarding computer security threats [1] and password strength [2]. In the study of computer security threats, CS majors were found to be significantly more aware of computer threats than non-CS majors. In contrast, when students were asked to rate the importance of passwords in protecting email security, the study found there is essentially no difference between the mean responses of CS majors and those of non-majors. However, when the strength of student passwords was measured, the passwords of CS majors were significantly stronger than those of non-CS majors.

3 Project Goals

The aim of this research is to examine the attitudes of college students toward the security of communication media in general and email in particular. The objectives of this research are:

- Examine the attitudes of CS majors.
- Examine the attitudes non-CS majors.
- Determine whether CS majors are more aware of the need for email security than non-CS majors.

4 Methodology

Primary research was conducted via a survey that was administered to students in all majors at a regional university. The purpose of the survey was to gather demographic information as well as information about students' attitudes and behaviors regarding email security. Participation was voluntary, and no information was collected that could be used to identify any participant. To encourage participation, all participants were entered into a drawing using a randomly generated number that was provided to them after they had completed and closed the survey, to prevent any breach of confidentiality. The prizes were five $25.00 gift cards. Each participant could only win once.

5 Results

Complete surveys were received from 527 students. Of these, approximately 20 percent were CS majors. Of the CS majors, the majority was male; of the non-CS majors, the majority was female. There is no graduate program in CS. Table 1 summarizes the data.

	Male	Female	Freshman or Sophomore	Junior or Senior	Graduate Student	Total
CS Major	93	24	69	48	0	117
Non-CS Major	90	320	125	191	94	410

Table 1: Demographic Information.

The students were asked to rank the relative security levels applicable to various forms of communication using a five-point scale (1–Not very secure). A one-way ANOVA test ($F(1, 530) = 3.859$) was used to determine whether there were any statistically significant differences between the mean responses of two groups. The results are summarized in Table 2.

Communication Media	CS majors		Other majors		F value	P-value
	Mean	Variance	Mean	Variance		
Email	3.197	1.108	3.507	0.906	9.275	.0024
E-Commerce	3.145	0.660	3.276	0.923	1.814	.1786
Mail	2.957	1.162	3.322	0.971	12.006	.0006
Landline phone	3.000	1.086	3.363	1.191	10.301	.0014
Cellular phone	2.897	1.248	3.190	1.292	6.093	.0139
Texting	2.761	1.356	3.118	1.338	8.678	.0034
Television	3.051	1.359	3.377	1.238	7.681	.0058
Social Media	2.274	1.459	2.406	1.480	1.090	.2969

Table 2: Summary of Responses Regarding Security Levels of Various Communication Media.

As can be observed, the F value obtained is larger than the F criterion value (3.859) in all cases except for e-commerce and social media, indicating that the difference of opinion between CS majors and other majors is statistically significant for all other forms of communication. It is worth noting that in every case, the average response by CS majors indicates their perception that the security level of each medium is lower than that of other majors. It is also interesting that both groups identified social media as having the lowest level of security on average, while both groups identified email as having the highest level of security. Given the well-publicized privacy breaches in popular social networks, perceptions regarding social media are unsurprising. However, one would have expected students to have greater concerns about the security of email communications.

To further investigate students' perceptions regarding email security, the survey included questions designed to measure their opinions regarding the importance of email passwords as well as their consideration of the source of an email message when deciding whether the message is safe. Table 3 summarizes students' responses to a question asking them to rate the importance of passwords for providing email security on a five-point scale (1–Not very important). Although the difference between the mean responses of the two groups is negligible, previous research indicates that CS majors have stronger passwords than other majors [2].

Computer Science majors		Other majors		F value	P-value
Mean	Variance	Mean	Variance		
4.581	0.469	4.579	0.403	.0008	.978

Table 3: Perceptions Regarding Importance of Email Passwords.

6 Conclusions

This study focused on differences in student perceptions regarding various forms of communication, with particular emphasis on email communication. With the exception of social media, CS majors considered various forms of communication to be less secure when compared to students in other majors. In addition, CS majors were found to be more aware of email safety issues when considering the source of the email message (with the exception of companies with which the recipient had a commercial relationship).

Perhaps the results are unsurprising, given that CS majors are expected to have greater technical expertise and awareness of information security issues. However, it is interesting to note that the differences in perceptions regarding the safety of communication media was significant even for traditional media such as television and land line phones. More importantly, the results underscore the need for additional formal and non-formal training in email security issues for students other fields.

References

[1] Cordova, J., Eaton, V., Greer, T., and Smith, L. A comparison of cs majors and non-cs majors' attitudes and practices regarding computer security threats. *The Journal of Computing Sciences in Colleges 33*, 2 (2017), 4–10.

[2] Cordova, J., Eaton, V., Greer, T., and Smith, L. A comparison of cs majors and non-cs majors' attitudes and practices regarding password strength. *The Journal of Computing Sciences in Colleges 33*, 4 (2018), 69–75.

[3] Dhamija, R., Tygar, D., and Hearst, M. Why phishing works. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montreal, Quebec, Canada, April 22-27, 2006), 581–590.

[4] Downs, J., Holbrook, M., and Cranor, L. Decision strategies and susceptibility to phishing. *Symposium on Usable Privacy and Security* (Pittsburgh, PA, July 12-14, 2006), 79–90.

[5] Herath, T., Chen, R., Wang, J., Banjara, K., Wilbur, J., and Rao, H. R. Security services as coping mechanisms: An investigation into user intention to adopt an email authentication service. *Information Systems Journal 24*, 1 (2014), 61–84.

[6] Jensen, M., Dinger, M., Wright, R., and Thatcher, J. Training to mitigate phishing attacks using mindfulness techniques. *Journal of Management Information Systems 34*, 2 (2017), 597–626.

[7] Sheng, S., Holbrook, M., Kumaraguru, P., Cranor, L., and Downs, J. Who falls for phish? a demographic analysis of phishing susceptibility and effectiveness of interventions. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, GA, April 10-15, 2010), 373–382.

[8] Smith, S. V. The market for stolen passwords. `https://www.npr.org/sections/money/2018/02/08/584304032/the-market-for-stolen-passwords`.

[9] Wang, J., Li, Y., and Rao, H. R. Overconfidence in phishing email detection. *Journal of the Association for Information Systems 17*, 11 (2016), 759–783.

Project Based Learning in Computer Science: A Student and Research Advisor's Perspective*

John W. McManus and Philip J. Costello
Department of Computer Science
Randolph-Macon College
Monroe, LA 71209
johnmcmanus@rmc.edu, philipcostello@go.rmc.edu

Abstract

The dominant pedagogy for Computer Science education often focuses on teaching isolated skills across a set of core courses. Recently the community has adopted Senior Capstone Projects and Project Based Learning to provide a cumulating experience designed to include a clear focus on: critical thinking and problem solving; collaboration and leadership; verbal and written communications; and independent work. This paper discusses the application of project-based learning to Computer Science education, providing a student's perspective and a research advisor's perspective. An example of applying Project Based Learning is presented and the effectiveness and relevance of Project Based Learning for Computer Science education is discussed.

1 Introduction

Courses in computer science, and higher education in general, tend to teach isolated skills with limited time set aside to assist the student in assimilating the skills [5]. Assimilation is critical to the development of a deeper understanding of how the academic material fits into a broader set of knowledge, skills, and abilities [10, 13]. A conceptual understanding of the course materials is not

sufficient for students to achieve success in today's challenging environment. The field of computer science is rapidly evolving. Students require a broad range of skills beyond programming languages, algorithms, and data structures. Rice and Shannon point out that educators have been consistently told that we need more engaging, autonomous, authentic, and cooperative learning processes in our formal educational institutions [10]. A major study of what is required to succeed in college courses [2] found general "habits of mind" to be key, along with subject-specific knowledge and skills. Employer surveys show similar expectations, graduates must be able to think critically and solve problems; work well with others; communicate in a clear and effective manner; and manage themselves and their projects effectively [8, 4]. These competencies are not optional, they are required to be successful in the current environment [1]. The core competencies must be taught concurrent with the acquisition of content knowledge and understanding. Students do not learn critical thinking skills in the abstract, isolated from subject matter. Students gain competencies by thinking critically and applying the skills in conjunction with core computer science domain knowledge. Project-based learning (PBL), organizes learning around projects [6]. Students determine how to approach a problem and what activities to pursue. Their learning is connected to a "real", not academic, problem and involves core competencies such as collaboration, communications, and reflection. At the completion of their projects students demonstrate their newly acquired skills in a public forum. The students are evaluated on both "what" they have learned, how effectively they apply it, and how well they communicate it. Throughout the PBL process, the teacher's role is to guide and advise the student, and not to direct and manage student's work [9, 11, 13]. Project-based learning may be applied in individual courses or throughout a curriculum, the projects can be combined with traditional teaching, projects may be carried out as individuals or in small groups and projects can vary in duration from a few weeks up to a whole year [8]. The projects are complex tasks, based on challenging questions or problems that involve students in designing, problem-solving, decision making, or investigative activities. They provide students the opportunity to work relatively autonomously over extended periods of time; and culminate in realistic products and presentations [3, 1].

2 Methodology

Project Based Learning (PBL) is applied in two student research programs at Randolph-Macon College; the Senior Capstone course, and the Schapiro Undergraduate Research Fellowships (SURF). The programs foster student research and promote learning outside of the traditional classroom environment.

The programs have applied concepts learned at a Buck Institute for Education PBL [1] workshop to revise the improve student outcomes. The Buck Institute has developed a comprehensive, research-based model for PBL to help educators develop, assess, and improve their PBL practices [1, 7]. Experience with twenty projects over the past four years shows that projects must include a clear focus on: critical thinking and problem solving; collaboration and leadership; verbal and written communications; and self-management. We believe that these attributes are crucial stepping stones to long-term student success.

The Senior Capstone course is a required course. Students work on a project over a full semester. The student selects a project that draws on knowledge, skills, and abilities from several core computer science courses. The projects are designed to engage students in solving a real-world problem or answering a complex research question. The students demonstrate their knowledge and skills by developing a public product, a research report, and a presentation at Research Day at the end of the spring semester. The full semester projects allow students to develop deep content knowledge, critical thinking and problem-solving skills, creativity, and communication skills in the context of an authentic, meaningful project[1].

SURF fosters student research and promotes learning outside of the traditional classroom environment. The ten-week fellowship runs from early June to early August. Research projects represent a full-time assignment for the student. Each student submits a final written research report and presents their findings/conclusions at the annual SURF conference. Students present their results at Research Day at the end of the following spring semester.

3 Implementing the Project Based Learning Model

An iterative approach is applied for our PBL projects. During each phase of the project the students develop preliminary artifacts, and then refine their design. We use a gated project review process to help students develop time management skills. Students work with their faculty advisor to do preliminary research and develop the conceptual model of their project prior to the start of the Senior Capstone or SURF project. It is critical that the students have direct input in selecting their project. This student "voice" creates a sense of ownership in the project. The students are personally invested in the project, work harder, and in our experience, perform better. During the first two weeks of the project the student refines their preliminary concept and develops a full schedule for the project. Senior Capstone students give update presentations at the end of week four and week ten, revising their schedules

and scope of work if necessary. SURF students present updates at the end of weeks three and eight. The presentations provide an opportunity to reflect on the projects. Reflection on the knowledge, skills, and abilities gained helps the students internalize what they have learned and think about how it might apply elsewhere, beyond the project [1, 7, 9]. Reflecting on skills development helps each student understand their value outside an academic environment. Reflecting on the project helps the research advisers improve the outcomes of the PBL projects. The students complete their research and document their findings during the last week of their project. All students write a research paper on their project and present at a public research symposium. Providing a public forum for student work is an effective way to communicate the value of a liberal arts education and what it provides for students. The presentations allow students to demonstrate the breadth of their knowledge, skills, and abilities; which helps build understanding of the true value of their educational experience.

The co-authors have worked together on three PBL projects; two SURF projects (2015 and 2016) and a Senior Capstone project. The 2016 SURF project is detailed to demonstrate the PBL methodology applied. The project is a multi-disciplinary project, developing a Unmanned Aerial Vehicle-based flight research system to support Environmental Studies research.

3.1 Example Project. Project Orion SURF Project

This research project investigated the feasibility of developing a low-cost modular research system for Unmanned Aerial Vehicles (UAVs). Research and data collection for many disciplines can prove cost prohibitive, as the instruments and methods of collection can be expensive. There are multiple companies that produce UAVs for research. The commercial UAV's are cost prohibitive for institutions that do not have large research budgets. The goal of this project was to demonstrate that it is possible to develop a low cost, modular sensor package that maintains accuracy in data collection and can be mounted on a consumer UAV. The research system deployed is comprised of: a GPS; an Infrared Camera; and temperature and humidity sensors. All sensors are operated autonomously via a research system computer. The project is divided into two parts: the development and integration of the research system in the lab; and field testing the research system mounted to the UAV. The project developed research software that runs autonomously while collecting accurate quantitative data for the user. The project required the student to: research and select sensor packages, the flight vehicle, and the research system computer; integrate the research system computer with the flight vehicle; and develop data collection and sensor fusion algorithms. The student developed the research proposal, including a 10-week schedule and detailed budget for

the research equipment. The student was responsible for the design, build, and testing of the flight hardware system and all software.

3.2 Research Flight Computer

The student selected a Raspberry Pi (RPi) to host the research system. The RPi is a Single Board Computer produced by the Raspberry Pi Foundation. The RPi is designed as a "low-cost, high performance computer" [10] that can be used for a broad range of applications. The RPi's modularity and General-Purpose Input/Output interface pins [12] allowed the designer to incorporate diverse sensors. RPi's are also inexpensive, and easy to setup. They have a Linux based operating system, support for wired and wireless networking, and can be powered by lightweight batteries. The RPi has a compact footprint and low power consumption.

3.3 Flight Test Vehicle

Figure 1: The Flight Test Vehicle

The UAV selected as the flight test vehicle is a DJI Phantom 4 drone. The Phantom 4 has a maximum speed and altitude of 45mph and 6000m [Figure 1]. The Phantom 4 has a 4K camera that can be used in tandem with the sensor package to gather data. There are multiple applications that can process video and still images from the camera and generate a 3-dimensional map of a specific area.

3.4 Software

A critical aspect of this project involved designing software to integrate the sensors and accurately read and log the data. The software is designed to run autonomously while the research system is active. The software was designed in multiple steps, with version 1 focusing on accurately logging the data from the GPS sensor so the user can tag the longitude and latitude coordinates as the data

Figure 2: The Research System

is collected. Version 1 also dealt with error exceptions if the GPS loses signal mid-flight. Version 2 focused on the integration of the temperature and humidity sensor. Additional error handling was introduced in Version 3 to handle situations when the sensor cannot be polled or becomes inoperable during flight.

3.5 Mounting the Research System on the UAV

The main challenge of designing the research system and mounting it to the UAV was managing the overall weight and balance of the system. If the research system is too heavy, or the weight is unbalanced, it will interfere with flight stability. Preliminary testing determined that the overall weight of the system should be under 500g. Before fabrication, it was estimated that the research system would weight approximately 440.43g. The final weight of the research system was 444.8g, with the extra weight being accounted for by the rubber grommets and stainless-steel screws used to secure the components to the Plexiglas panels.

3.6 Project Orion's Research Results

Upon completion of the testing phase it was determined that Project Orion was a success, demonstrating that it is possible to design a cost-effective modular sensor platform for a UAV.

4 Conclusion

4.1 The Student's Perspective on PBL

As mentioned previously, most courses only provide students a conceptual understanding of basic Computer Science skills. Many students prepare for these classes by learning and comprehending the bare minimum to obtain their desired course grade. The PBL approach provides students with an environment to practice the skills already learned, and a chance to develop all new skills that are difficult to simulate in a classroom environment.

The PBL approach in Senior Capstone and SURF projects drives students to perform at their best due to the sense of project ownership. The BPL approach allows students to pursue projects in research areas that they select, stimulating the student's natural curiosity towards the subject. In turn, that curiosity drives to student to problem solve and generate solutions that could not have been stimulated through standard coursework. Additionally, a PBL project does not focus on one isolated problem that can be solved using a specific technique or algorithm demonstrated in coursework. PBL projects consist

of multiple inter-related subproblems. The subproblems usually have multiple approaches that can be taken to develop the solution. A solution to one problem may create an all new problem or prevent another subproblem from being solved. This forces students to think on their feet and make informed decisions during the planning and development phases of a PBL project.

PBL also helps foster the growth of professional skills and discipline with students. Meeting with advisors on a weekly basis engrains a sense of discipline as both the Senior Capstones and SURF have a compressed schedule. That personal accountability forces students to develop time management skills. Students also learn how to effectively plan their development schedule. Just like in the professional world, one can only spend so much time pursing a solution to a problem before there is a point no return and the project cannot be completed. Students must monitor their progress throughout the course of a PBL project. If the current approach to the problem is not yielding results, they must reevaluate the problem and adjust their approach. The final research paper and presentation provide students an opportunity to refine their writing and public speaking skills in an open public forum. This is a crucial aspect of the PBL approach, especially for students in Science, Technology, Engineering, and Math (STEM). STEM topics are often complicated to explain. Undergraduate STEM students who can communicate complex topics in clear and simple language have a distinct advantage over their peers.

4.2 The Research Advisor's Perspective on PBL

The PBL approach has improved the outcomes for our Senior Capstone and SURF students. The PBL projects allow students to select a project that they are interested in, and to explore it independently. The gated project development process provides a framework with sufficient structure to help ensure their success and places the responsibility for developing detailed plans and schedules on the students. This approach helps students develop time management skills. Research advisors meet with the students weekly (more often if required) to review the student's progress and provide technical support.

The project formation phase is critical to the overall success of the project. Our experience shows that students who invest time in researching potential topics prior to drafting their final proposal start from a stronger foundation and experience fewer issues completing their projects. Starting the formulation process two months prior to submitting the final project proposal allows the students time to evaluate alternative approaches and develop their preliminary research plan.

The presentations and research papers provide an opportunity for the students to reflect on their project and demonstrate the breadth of skills that they have developed / refined. The clear focus on verbal and written commu-

nications skills development helps each student understand the value of clear and concise communications. Student have multiple opportunities to practice their skills prior to their final presentations. The public forum for student to present in is an effective way to communicate the value of a liberal arts education and what it provides for our students. The presentations allow students to demonstrate the breadth of their knowledge, skills, and abilities; which helps build understanding of the true value of their educational experience. Several students have continued their research, presenting their results at national conferences.

5 Acknowledgments

The authors thank Randolph-Macon College and the Schapiro Undergraduate Research Fellowship for supporting this research.

References

[1] Gold standard PBL: Essential project design elements. `http://www.bie.org/blog/gold_standard_pbl_essential_project_design_elements`.

[2] Conley, D. *College knowledge: What it really takes for students to succeed and what we can do to get them ready.* Jossey-Bass, San Francisco, California, 2005.

[3] Dopplet, Y. Implementation and assessment of project-based learning in a flexible environment. *International Journal of Technology and Design Education 13* (2003), 255–272.

[4] Hart Research Associates. It takes more than a major: Employer priorities for college learning and student success. an online survey among employers conducted on behalf of: The association of american colleges and universities. washington, dc. `https://www.aacu.org/leap/documents/2013_EmployerSurvey.pdf`.

[5] Jaime, Arturo; Blanco, J. M., and Dominguez, C. Spiral and project-based learning with peer assessment in a computer science project management course. *Journal of Science Education and Technology 25(3)* (2016).

[6] Karaman, S., and Celik, S. An exploratory study on the perspectives of prospective computer teachers following project-based learning.

International Journal of Technology and Design Education 18(2) (2008), 203–215.

[7] Larmer, John; Mergendollar, J., and Boss, S. *Setting the Standard for Project Based Learning: A Proven Approach to Rigorous Classroom Instruction.* ACSD, Alexandria, Virginia, 2015.

[8] Mills, J. E. T. D. F. Engineering education – is problem-based or project-based learning the answer? *International Journal of Technology and Design Education 18(2)* (2008), 203–215.

[9] Moursand, D. *Project based learning: Using Information Technology.* International Society for Technology in Education, Washington, D.C., 2008.

[10] Rice, Marilyn; Shannon, L.-J. Developing project based learning, integrated courses from two different colleges at an institution of higher education: An overview of the processes, challenges, and lessons learned. *Information Systems Education Journal v14 n3* (2016), 55–62.

[11] Solomon, G. Project-based learning: A primer. *Technology and Learning 23(6)* (2003), 20–27.

[12] The Raspberry Pi Foundation. The raspberry pi. `http://raspberrypi.org`.

[13] Thomas, J. W. A review of research on project-based learning. `http://www.bobpearlman.org/BestPractices/PBL_Research.pdf`.

Expanding Communication Opportunities in Algorithms Courses*

Robin M. Givens
Computer Science Department
Randolph-Macon College
Ashland, VA 23005
robingivens@rmc.edu

Abstract

This paper describes the creation of and experience with a communication-focused project for an undergraduate algorithms course. The project was designed to provide students with a communication-based project, help students develop an understanding of algorithms in a practical context, and provide a way to discuss some algorithms that fall outside the scope of the course. Students chose an algorithm to research and delivered their findings via a paper and an in-class presentation. Papers and presentations focused on the historical, functional, and complexity details of the chosen algorithm. An example course schedule, assignment details, and algorithm list are provided. We also reflect on student outcomes and discuss potential changes for future semesters.

1 Introduction

The need for computer science students to communicate effectively has been discussed for decades. In an effort to increase communication skills, computer science programs have created dedicated communication courses [1, 5, 8, 11] or integrated writing throughout all courses, often referred to as "writing across the curriculum" [3, 4, 7]. Some of the reasons computer science programs have wanted to boost communication skills in their students are to help students

learn to communicate, to have students learn through communication, to fulfill an institutional requirement [6], or in an effort to increase teamwork [10].

Communication is not only a tool and skill desired in academia. Companies are specifically interested in hiring computer science students who can communicate ideas and solutions well. The NACE Job Outlook 2018 report found that computer science is one of the most desired degrees, communication (written) was one of the most highly desired skills, and communication (verbal) was also a top 10 sought after skill [12]. When asked by IEEE-USA Today's Engineer what employers are looking for, Ben Amaba, an executive at IBM said, "Software engineers need good communication skills, both spoken, and written" [10, 14].

A course on algorithms, which is required in most computer science programs, introduces a student to formal algorithm design and analysis. Some required algorithms courses emphasize design through data structures [16], while others emphasize analysis through complexity theory [9]. Courses may introduce algorithms by topic such as searching, sorting, and graph theory [2] or by method such as greedy algorithms, divide and conquer, and dynamic programming [13]. Regardless of structure or placement in the curriculum, an algorithms course will typically introduce, reinforce, or utilize each of these areas.

In [15], the author discussed two writing assignments in an algorithms course: an experimental research paper and a survey paper that begins with an annotated bibliography. We expand this work by examining the addition of a research project, involving a written paper and oral presentation, to an upper-level undergraduate algorithms course. The assignment presented in this paper focuses on the study of one particular algorithm and the concepts learned in an algorithms class: an algorithm's history, complexity, and relationship to other algorithms that solve the same problem. We also provide a detailed project design.

For the written and oral project, students were required to research an algorithm, write about it, and present their findings to the class. This project was added to the algorithms course in a general effort to increase student experience with both written and verbal communication throughout the computer science curriculum. The objectives of the project were to 1. provide a project focused on communication, 2. help students develop a deeper understanding of the historical significance of algorithms and their complexity and 3. provide an avenue for discussion of algorithms that may be outside the scope of the course.

1.1 Environment

Randolph-Macon College is a small, private, liberal arts college with a long history in teaching computer science, having established its Computer Science Department in 1967. The algorithms course is required for the computer science major, is an elective for the minor, and is taught in the spring semester. Algorithms is usually taken in the junior or senior year with 10-15 students and requires a minimum grade of C− to count for the major or minor. The course has a prerequisite of data structures and a corequisite of discrete mathematics, the latter of which is taught in the Mathematics Department.

Table 1: Example Course Schedule. Due dates are for the communication assignment. The semester schedule at Randolph-Macon is slightly shorter than semesters at most schools, but each class session is longer (1 hour 3-day courses and 1.5 hour 2-day courses). A 15-week calendar could allow for 2 weeks of student presentations for larger class sizes.

Week	Lectures/Exams	Due Dates
1	Intro and Discrete Math Review	
2	Discrete Math Review	
3	Algorithm Analysis and Graphs	
4	Greedy Algorithms	Topic Selection
5	Divide and Conquer	
6	Review and Exam 1	
7	Dynamic Programming	
8	Complexity: Searching	
9	Complexity: Sorting	
10	Theory of NP	Paper Draft
11	Review and Exam 2	
12	Depth, Breadth, and Best First Search	Presentation Draft
13	Student Presentations	Final Presendation
14	Final Exams	Final Paper

The algorithms course includes two weeks of discrete math review/preview which is followed by topics in algorithm analysis, greedy algorithms, divide and conquer algorithms, dynamic programming, and complexity theory. An example course schedule with due dates for the communication assignments can be seen in Table 1. Students complete seven homework assignments including an introduction to LaTeX, written summaries, and various combinations of creating, coding, analyzing, and working through algorithms. The course has one coding project, which also requires a write-up, and one research project, which requires a research paper and oral presentation. The development of the

student research project is the focus of this paper.

2 Project Details

The project was divided into five parts: topic selection (5%), draft paper (15%), draft presentation (15%), in-class presentation (25%), and final paper submission (40%). Topic selection occurred in the first half of the semester with due dates for the remaining parts in the second half, as seen in Table 1.

Students were given 16 algorithms from which to choose, as seen in Table 2, with the option to propose an algorithm for approval by the professor. However, no students proposed an algorithm. The algorithms ranged in level of difficulty. Some algorithms had been taught in previous semesters of the algorithms course or would be taught in other courses (such as cryptography), while other algorithms were beyond the scope of a typical undergraduate analysis or coding assignment. Challenging algorithms were expected to be covered in high-level terms while more manageable algorithms were to be covered in greater detail. Students were not informed of the difficulty level of the algorithms, only given a brief description of each algorithm's use.

Table 2: Sample Algorithm Topics. Students were also allowed to propose an algorithm. Underlined topics were chosen in the spring 2018 semester.

Rabin-Karp Algorithm	Fast Fourier Transform
Damerau-Levenshtein Distance	Reed-Solomon Error Correction
RSA	Lagged Fibonnaci Generator
Bloom Filters	MapReduce
Cryptographic Hash Functions	Kalman Filter
Shor's Algorithm	Hunt and Kill Algorithm
Latent Dirichlet Allocation	Radix Sort
K-Nearest Neighbors	The Schulze Method

2.1 Paper and Presentation Format

Students were required to include an introduction, background, algorithm description, complexity analysis, conclusion, and references in their paper and presentation. Drafts were expected to be complete papers without any missing parts. Students were given the following descriptions, also provided in a LaTeX template, for the sections of their paper.

- **Introduction** – Provide a summary and overview of the algorithm discussing its importance and use.

- **Background** – Provide historical information related to the algorithm. Answer questions such as: Who came up with the algorithm? When? What other important things is the creator known for? Why was the algorithm created? Why is it important? What other uses does it have? Are there any concepts you need to know before you can understand the algorithm? If so, define or explain them, provide pictures if that will help.

- **Algorithm** – Provide a description of the algorithm and how it works. What kind of algorithm is it (greedy, backtracking, divide and conquer, etc)? The type may be one not covered in class. Provide pseudocode or a very good outline of the algorithm, explain either. Provide at least one example with pictures.

- **Complexity** – Determine the complexity and discuss it. What is the complexity of algorithms that solve the same or similar problems, and is this one better or worse? What are some suggested optimizations for the algorithm, if any?

- **Conclusion** – Review the important points about the algorithms. Finish with the advantages/disadvantages of the algorithm.

- **References** – Provide at least three references that must be cited in your paper. Do not use more than one wiki-style site. If you use images created by someone else, include the reference and cite the images in your paper.

For their presentations, students were expected to cover the same concepts as the paper, dedicating at least one slide to each section. Presentations were to be rehearsed and to take 10-15 minutes, followed by questions. Feedback was provided after both the paper and presentation draft submissions, and students were required to incorporate suggested changes into the in-class presentation and final paper.

3 Student Outcomes

Algorithms chosen in spring 2018 are underlined in Table 2. Draft submissions by the students varied from needing significant changes to needing few changes. Many students needed to improve the quality and formatting of their references and the use of citations within the paper. Though examples were given, some students seemed unfamiliar with the correct way to quote or cite a reference and struggled with the appropriate amount of quotations. In particular, they

were expected to spend the majority of their paper paraphrasing and reworking their research, instead of quoting.

In the future, requiring an annotated bibliography or outline prior to draft submission, with the clear requirement of minimal quotation use, may improve the issue with proper citations and quotations. A separate, thought-out focus on the bibliography could also allow for the requirement of *no* wiki-style references. A few students found good sources with textbooks, tech sites, news articles, and some conference and journal papers. Other student struggles included difficulty in finding and providing an example of their algorithm in use, and, for the presentation, keeping bullet points succinct.

After receiving feedback, most students significantly improved their presentations and final papers. Overall, in-class presentations were well done, and provided an impetus for interesting discussion and questions from the students. Students were particularly interested to find out what would happen when a cryptocurrency runs out of supply after the presentation on cryptographic hash functions. Students were also able to discover, understand, and explain important advantages and disadvantages of their chosen algorithm. Examples include the linearity, but inflexibility, of radix sort; the difficulty in breaking RSA mathematically, but the ability of researchers to break it by listening to the CPU (acoustic cryptanalysis); and the ability to simply break RSA with Shor's Algorithm, but the unrealized potential of quantum computing.

Some algorithms chosen had been studied in previous semesters of algorithms or similar courses (though not by the students in this particular course). For algorithms that were similar in difficulty to those studied in the course, students showed an ability to research, understand, and analyze the algorithms, and provide examples of the algorithms working step by step. For more ambitious algorithms, such as the Kalman Filter and Shor's Algorithm, students were able to describe and explain the algorithms at a high-level and give examples of their use in real-world situations.

Anecdotal evidence shows that the communication-focused research project potentially had a positive impact on student understanding of algorithmic concepts. Of the eight students who completed the course in spring 2018, six students received a higher grade on their final exam than the average of their other exams, and two received a lower grade. The average increase for all students was 4.4 points with five students increasing between 3.2 and 7.7 points, one increasing 23.7 points, one decreasing 0.7 points, and one decreasing 12.3 points. From experience, it is unusual for a majority of students to perform better on the final exam than previous exams, however further work would need to be considered for conclusive results.

4 Conclusions

The research project for the algorithms course was created to provide a communication-focused project, help students gain a well-rounded understanding of algorithm importance and complexity, and provide experience with some advanced algorithms. The project was able to combine these goals by incorporating important concepts learned in an algorithms course while focusing on both written and verbal communication. Students were required to write a paper and give an in-class presentation in which they described the history, purpose, use, complexity, and other details of a chosen algorithm. Feedback on draft papers and slideshows helped students improve their work. In the future, we plan to require students to summarize their references early in the assignment to help quell the overuse of quotations and increase comfort with citations.

References

[1] Blume, L., Baecker, R., Collins, C., and Donohue, A. A communication skills for computer scientists course. In *ACM SIGCSE Bulletin* (2009), vol. 41, ACM, pp. 65–69.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*, 3 ed. MIT press, 2009.

[3] Falkner, K., and Falkner, N. Integrating communication skills into the computer science curriculum. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (2012), ACM, pp. 379–384.

[4] Fell, H., Proulx, V., and Casey, J. Writing across the computer science curriculum. In *ACM SIGCSE Bulletin* (1996), vol. 28, ACM, pp. 204–209.

[5] Havill, J., and Ludwig, L. Technically speaking: fostering the communication skills of computer science and mathematics students. In *ACM SIGCSE Bulletin* (2007), vol. 39, ACM, pp. 185–189.

[6] Hoffman, M., Dansdill, T., and Herscovici, D. Bridging writing to learn and writing in the discipline in computer science education. In *ACM SIGCSE Bulletin* (2006), vol. 38, ACM, pp. 117–121.

[7] Jackowitz, P., Plishka, R., and Sidbury, J. Teaching writing and research skills in the computer science curriculum. In *ACM SIGCSE Bulletin* (1990), vol. 22, ACM, pp. 212–215.

[8] Kay, D. Computer scientists can teach writing: an upper division course for computer science majors. In *ACM SIGCSE Bulletin* (1998), vol. 30, ACM, pp. 117–120.

[9] Kleinberg, J., and Tardos, E. *Algorithm Design.* Addison-Wesley, 2005.

[10] Lingard, R., and Barkataki, S. Teaching teamwork in engineering and computer science. In *Frontiers in Education Conference (FIE)* (2011), IEEE, pp. F1C1–F1C5.

[11] McDonald, G., and McDonald, M. Developing oral communication skills of computer science undergraduates. In *ACM SIGCSE Bulletin* (1993), vol. 25, ACM, pp. 279–282.

[12] National Association of Colleges and Employers (NACE). Job outlook 2018, Nov. 2017.

[13] Neapolitan, R. *Foundations of Algorithms*, 5 ed. Jones & Bartlett Learning, 2014.

[14] Platt, J. Career focus: Software engineering, Mar. 2011. `http://www.cis.umassd.edu/~hxu/courses/cis582/CareerFocus.html`.

[15] Weikle, D. Two concrete examples of upper-level writing assignments in an algorithms course. *Journal of Computing Sciences in Colleges 28*, 3 (Jan. 2013), 14–20.

[16] Weiss, M. *Data Structures & Algorithm Analysis in C++*, 4 ed. Pearson Education, 2013.

Scaffolding Assignments: How Much Is Just Enough?*

James Vanderhyde
Computer Science
Saint Xavier University
Chicago, IL 60655
vanderhyde@sxu.edu

Abstract

Instructional scaffolding is a teaching practice that gradually shifts the initiative in learning from the teacher to the student. In this study, we embedded different degrees of scaffolding into several CS1 and CS2 homework assignments and observed which proved most effective for student learning. We evaluated student learning by measuring scores on exam questions related to these assignments. Our initial analysis indicates that high levels of assistance embedded in a homework assignment are likely to result in reduced learning. Results related to very low levels of assistance are inconclusive. We conclude by encouraging instructors to avoid "spoon feeding" and "sink-or-swim" approaches, and instead to take great care to include appropriate levels of scaffolding in their assignments.

1 Introduction

A common misconception among inexperienced educators is that the more you can do for your students, the better. However, a learning environment that "does it all for them" is going to be poor at teaching[6]. If the instructor (or the instructor-provided material) jumps in with the answer before the student has a chance to figure it out, the instructor is "rescuing" instead of teaching. Thompson[7] asks teachers, "Who worked harder during that lesson?" If it

was the teacher, then something is wrong. Rescuing can also be called "spoon feeding," because the student is not taking initiative for learning.

At the other extreme, educators sometimes assign a homework problem and assume that if a student cannot solve it, it is because the student is not capable. However, there may be underlying causes such as a lack of confidence or lack of social support. This problem is exacerbated for students from underrepresented populations because they are particularly susceptible to being "weeded out" by this "sink-or-swim" approach[5, 3]. Therefore, if computer science educators are truly committed to diversifying the field, we are obligated to engage in effective pedagogical practices that are helpful to all and essential for marginalized students.

An effective practice that addresses both of these issues is instructional scaffolding. This is a term invented by Wood, Bruner, and Ross][9]. Terry Thompson defines it this way: "scaffolded instruction characterizes a pattern of teaching that shifts the level of responsibility for the learning from the *more knowing other* to the *less knowing other*"[8].

Scaffolding can be applied when the instructor is working closely with the student, either in class, in the lab, or in office hours. However, much student learning occurs when the student is working outside the supervision of the instructor on homework assignments. In this work we contribute to the body of work on scaffolding by focusing on the construction of effective homework assignments in CS 1 and CS 2. We believe a carefully scaffolded assignment can avoid the problems associated with either rescuing or the sink-or-swim approach. To test this belief, we gave students assignments with varying levels of assistance and measured learning through test performance.

2 Related work

In their review of literature on scaffolding research in all educational fields up to 2010, Van de Pol et al.[1] state, "With this somewhat limited body of effectiveness research on mainly one-to-one tutoring situations with mostly simple and straightforward tasks, future research might start to focus on more naturalistic classroom situations with all sorts of tasks." We have responded to their advice by extending the application of scaffolding in CS education to the arena of homework assignments. They further state that measurement of scaffolding is challenging because it is a dynamic process. Our project addresses this concern by varying and quantifying the amount of scaffolding that we use.

To illustrate the challenges in implementing effective scaffolding, we turn to Thomas et al.[6], who crafted an experiment to test the effectiveness of a particular intervention in a computer science class. The class was split randomly into two groups. One group was provided object diagrams and one was not. To

the authors' disappointment, the group given the extra scaffolding performed worse on posttests. The final conclusion was that they scaffolded the wrong task. They provided students with completed object diagrams, but they concluded that it was the students' own creation of the diagrams that resulted in learning. The authors called for further experiments to test this hypothesis. Our project addresses this call by measuring the amount of learning achieved when too much assistance is provided.

David Ginat[4] employed a scaffolded approach to learning a difficult CS skill (combining patterns in complex programming problems). The scaffolding consisted of instructor explanations and tips given at appropriate times. The author's insight was that there is a hole in the curriculum that leaves some students stranded when they try to combine programming patterns, and that careful intervention can help these students gain confidence. He expressed the "belief that the scaffolding approach may be very relevant in additional domains in computer science." Our work seeks to clarify how scaffolding can be done effectively in computer science homework assignments.

Van de Pol et al.[2] conducted their own study of the effect of scaffolding in a classroom. The study included eighth-grade social studies students across several schools in the Netherlands. The learning environment was group work in a classroom with a teacher circulating among the groups. The teachers were taught to provide "contingent" help—giving only as much help as is necessary. The authors' hypotheses were similar to those in our project. They said, "If the level of control is too high for a student, superficial processing of the information is assumed," and "If the level of control is too low for a student deep processing cannot take place." The study found (among other things) that scaffolding was good for improving student test scores, but only if applied infrequently. Test scores also improved when non-contingent help was applied, but more frequently. Our work addresses the same research questions in the context of college-level computer science students working on homework, rather than in-class work.

3 Approach

A typical homework assignment at the CS 2 level is to write a program to solve a given problem. In this context, scaffolding is anything that is provided to the learner that is in addition to the problem statement itself. A bare problem statement can be seen as sink or swim. Spelling out instructions on how to complete the assignment step by step is akin to rescuing: giving too much help and doing all the work for the learner. Effective scaffolding will fall somewhere in between. Below is an example homework assignment with an in-between level of scaffolding, provided as a series of hints.

3.1 Example assignment

Write a program that draws a pyramid of blocks of any given height, like this for height 4. Note that each block is a pair of square brackets (no space in between). There are no spaces on the bottom line. The program must use a recursive method to draw the pyramid.

```
   []
  [][]
 [][][]
[][][][]
```

Hint: The recursive method should do two things: Make a recursive call, and print one line of the pyramid.

Hint: Each line of the pyramid is a sequence of spaces followed by a sequence of blocks.

Hint: Use two parameters for the recursive method: one that holds the number of spaces to print before the blocks, and one that holds the number of blocks to print.

3.2 Hypothesis

Our overarching thesis is that the middle level is the best for student learning. You can call this the Goldilocks principle—not too much and not too little, but just right. Expanding on this, we hypothesize the following:

1. Students who completed homework assignments with no or just enough scaffolding did well on related exam questions (deep learning).
2. Students who completed homework assignments that provided too much assistance did poorly on related exam questions (shallow learning).
3. Students need to seek more outside assistance when the homework scaffolding is insufficient, or they will give up.

To address these hypotheses, we need to measure learning. For this study, we gave students homework assignments and looked at student performance on test questions related to the homework. We make the assumption that the test questions are written well, actually assess what they are intended to assess, and were graded fairly. They were written and graded by an experienced teacher. Therefore, we may further assume that if the students do well on the exam question, it is because they achieved deep learning, and if the students do poorly on the exam question, it is because they achieved shallow learning. There are always other variables involved in testing, but for this study we made these assumptions.

3.3 Experiment

In spring 2017, we created three different assignments and gave them during the second half of a second-semester introductory class (around CS 2 level).

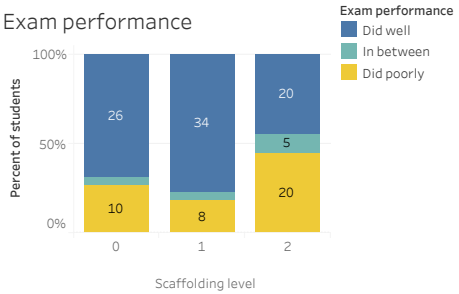
The class consisted of 29 total students who agreed to participate in the study. Each of the three assignments had a different level of scaffolding, which we call 0, 1, or 2, corresponding to none, just enough, and too much. The above example assignment was given at level 1. To be clear, the entire class was given the exact same assignments. The amount of scaffolding varied from assignment to assignment, not from student to student. The purpose of this was to control for discussion among students during the homework.

The next time the course was offered (spring 2018), there were 22 students who agreed to participate. We again created three different assignments, each at a different level of scaffolding. The assignments were similar to those from the previous year but different enough that the students could not copy existing solutions. We switched the level of scaffolding on two of the assignments (level 0 and level 2).

We collected scores on the homework assignments and scores on exam questions related to the homework assignments. We also gave the students a survey to find out if they had completed the homework on their own or used outside assistance (such as help from peers or websites).

4 Results

In this analysis, we included only the exam question scores of students who completed the corresponding homework assignment (at a level of 60% or above). The reason for this is that if they did not complete the homework successfully, presumably they did not learn from it. In that case, their exam scores are irrelevant, because we are examining the effect of homework scaffolding on exam scores.



In order to evaluate the hypotheses, we examined at each level of scaffolding what proportion of the students did well on the exam. We define "did well on the exam question" as scoring at 75% or above and "did poorly on the exam" as scoring below 50%. See figure. In the cases of the homework with no extra

scaffolding and homework with some scaffolding (scaffolding level 0 and 1), most of the students (73%) scored well on the associated questions on the exam. However, in the case of the homework with too much assistance (scaffolding level 2, rightmost bar in the figure), only some of the students (44%) scored well on the associated question on the exam, and several students (also 44%) scored poorly.

Homework scaffolding level	0	1	2
Number of students who completed the homework	38	44	45
Average exam score of the 29 students who completed all three homework assignments	77%	79%	58%

For statistical analysis, we used a repeated-measure ANOVA on the test scores for students who completed all three homework assignments ($n = 29$). See table for mean exam scores. The analysis showed that the distributions of scores on these exam questions are significantly different ($p < 0.01$). The distribution of scores and the statistical analysis support Hypotheses 1 and 2, in that the students scored better on the test questions associated with scaffolding levels 0 and 1 than they did those associated with scaffolding level 2.

There was no significant correlation between homework scores and test scores.

The survey asked the students to list the forms of external help they used on each assignment. Qualitative analysis showed that the external help fell into three categories: help from tutors, peers, and websites. We quantified the amount of external help by counting 1 for each of these categories. Thus, on a given assignment, a given student could score 0, 1, 2 or 3 on this measure. Because some students habitually seek more extra help than others, we used a Friedman test to control for differences between individual students. This required us to examine only students who completed the survey for all three assignments ($n = 14$). While the average external help score was higher on the homework with scaffolding level 0 (1.21) than the average score on the level-1 and level-2 homework (1.14 and 1.07, respectively), the Friedman test indicated that this difference was not statistically significant. Therefore, we need more data to evaluate Hypothesis 3.

4.1 Threats to validity

The following are some factors that made this experiment difficult, and how we attempted to mitigate them. First, students will help other students, thereby moving from no scaffolding to some unknown amount of scaffolding. The survey

is supposed to mitigate this in the questions about help received, but the survey questions should be improved in future studies.

Some students are going to learn, and some are going to fail, regardless of how well the homework is scaffolded. Future studies may be able to control for this factor in the analysis by using external measures such as student GPA, but we chose not to request this private information from the students involved in this study.

The distance between the assignments and the exams varied, but any effect from this should be small because exams were given frequently. Furthermore, assignments earlier in the class may naturally need more scaffolding. This effect should be minimal because the assignments are all near the end of this second semester course.

Finally, the difficulty inherent to each of the exam questions varied. This was difficult to control for in this study.

5 Future work

The survey results should be able to help us be more accurate with Hypotheses 1 and 2. In the current study, we can only examine how the level of scaffolding on the homework predicts the exam score. However, while students work on the homework, scaffolding may be provided from outside assistance as well. This was a confounding factor in our attempt to establish a relationship between homework scaffolding and test scores. If we can collect better information in the survey, then we can get a better idea of how scaffolding and rescuing in general affect test scores.

Over the next two years, we are planning to replicate this study with similar assignments but switching the amount of scaffolding provided on all of the assignments. The exam questions will be similar, but the homework assignments will be different. This will give us more data points to try to establish the relationship between scaffolding in the homework and learning as demonstrated on the exam questions. Since we are a small university, we can only control a few variables at a time, and the course is only offered once a year.

6 Conclusion

We designed an experiment to study how effective construction of homework assignments increases performance on exam questions. The results of the experiment show some indication that too much help on the homework can cause students to perform worse on exams. In other words, rescuing or spoon feeding students results in shallow learning. We also hypothesized that when students

are not given enough scaffolding on an assignment, they will either give up or seek outside assistance, but the experiment did not confirm or deny this. More data is required to evaluate this hypothesis.

We agree with Thomas et al.'s[6] conclusion: you should not scaffold something that you want the students to learn how to do. The scaffolding is the assistance the instructor gives that you do not expect the students to learn at this time.

Van de Pol et al.[2] made the point that teachers like scaffolding, but it is not something they naturally do. Part of the purpose of the current work is to encourage and empower computer science instructors to include effective forms of scaffolding in their assignments.

7 Acknowledgements

We would like to acknowledge the DEERS project for their support of this work (NSF DUE 1525373, 1525173, 1525028). We would also like to thank Jean Mehta for allowing us to modify some of the assignments in her class and her students for participating in the study.

References

[1] de Pol, J. V., Volman, M., and Beishuizen, J. Scaffolding in teacher-student interaction: a decade of research. *Educ Psychol Rev* (2010).

[2] de Pol, J. V., Volman, M., Oort, F., and Beishuizen, J. The effects of scaffolding in the classroom: support contingency and student independent working time in relation to student achievement, task effort and appreciation of support. *Instr Sci* (2015), 615–641.

[3] Gates, A. Q. The role of hispanic-serving institutions in contributing to an educated work force. *Communications of the ACM 53*, 12 (2010), 31–33.

[4] Ginat, D. Interleaved pattern composition and scaffolded learning. In *ITiCSE '09* (2009), pp. 109–113.

[5] Margolis, J., and Fisher, A. *Unlocking the Clubhouse: Women in Computing*. The MIT Press, 2001.

[6] Thomas, L., Ratcliffe, M., and Thomasson, B. Scaffolding with object diagrams in first year programming classes: some unexpected results. In *SIGCSE '04* (2004), pp. 250–254.

[7] Thompson, T. Are you scaffolding or rescuing?, 2010. https://www.choiceliteracy.com/articles-detail-view.php?id=735.

[8] Thompson, T. *The Construction Zone: Building Scaffolds for Readers and Writers*. Stenhouse Publishers, 2015.

[9] Wood, D., Bruner, J., and Ross, G. The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry 17*, 2 (1976), 89–100.

Less-Java, More Learning: Language Design for Introductory Programming*

Zamua O. Nasrawt and Michael O. Lam
Department of Computer Science
James Madison University
701 Carrier Drive, MSC 4103
Harrisonburg, VA 22807
nasrawzo@dukes.jmu.edu, lam2mo@jmu.edu

Abstract

We present Less-Java, a new procedural programming language with a simple and concise syntax, implicit but strong typing via type inference, and built-in unit testing. These features make programming in Less-Java more intuitive for novice programmers than programming in traditional introductory languages. This will allow professors to dedicate more class time to fundamental programming concepts rather than syntax and language-specific quirks.

1 Introduction

1.1 Project Goal

Introductory computer science courses lay the groundwork for future courses by teaching problem-solving techniques and fundamental programming concepts like loops, conditionals, and data structures. Many computer science departments use a mainstream language like Java in these courses, and there are good reasons to do so. Java is ubiquitous in industry, available on many

platforms, and provides an extensive standard library. Unfortunately, there are also drawbacks to using Java as an introductory language[1].

First, even simple Java programs are very verbose and unintuitive to beginners, with required class declarations, long method signatures ("`public static void main (String[] args)`"), and required semicolons. Students must write a lot of boilerplate code before writing program code. The boilerplate code is usually explained by the end of the course, but forcing students to write it without truly understanding it is disingenuous.

Second, Java requires all variables to be declared, forcing students to distinguish between declaration and initialization before they really even understand the concept of a variable itself. Many languages like Ruby and Python avoid this requirement by removing declarations and using dynamic typing. Unfortunately, these languages cannot be type-checked statically, which hurts novice programmers because it allows incorrectly-typed code to compile and run.

Finally, there is no native unit testing framework in Java (JUnit is a popular framework but it is a 3rd-party library). This means students must either do all of their testing in the main method or include a jar file in their project. The former is against software engineering best practices and the latter is unintuitive and tedious for novice programmers. Inaccessible unit testing discourages students from testing their code at all, which is damaging for the student and society at large [10].

To address these drawbacks, we present a new language named Less-Java with 1) a simple and concise syntax, 2) implicit but strong typing via type inference [8], and 3) built-in unit testing. The language also provides simple built-in standard I/O, a core set of built-in data structures (e.g., lists and maps), and basic object-oriented features (e.g., classes and objects). We aimed to retain just enough functionality to be useful in teaching introductory computer science courses without overwhelming novice programmers. To our knowledge, Less-Java is the first language to support all of these features.

1.2 Background

There are programming languages that attempt to address many of the critiques in the previous section. Scratch [9] and Snap [2] are educational languages often used in K-12 outreach efforts. They are visual, block-based languages designed to eliminate syntax errors entirely so that novice programmers can focus only on the logic of their program. Unfortunately, Scratch's limited vocabulary also makes it difficult to solve complex problems. Snap solves this problem by allowing definition of custom blocks, but both languages still re-

[1]To be clear, these drawbacks do not preclude it from being an important and useful language to learn in a more advanced course.

quire students to move blocks around. This does not necessarily translate well to college-level programming where students must use text-based languages.

Grace [4] is a more traditional procedural language that aims to "help novices at programming to learn how to write correct and clean code." Grace even goes as far as to enforce code style in the grammar; misaligned brackets or improper indentation cause errors when the program is compiled. This is nice for people that eventually need to read the code, but it is a source of frustration for students. Grace also includes advanced features (e.g., lambdas) that are unnecessary for most introductory programming courses.

Some universities use scripting languages such as JavaScript, Ruby, and Python, allowing students to begin programming with simpler syntax and less boilerplate. However, there is some evidence that students may actually struggle more when the syntax abstracts away underlying details [3]. In addition, these scripting languages are also usually dynamically typed, which can induce subtle type problems like inadvertently passing a string representation instead of the underlying object. Finally, scripting languages often include large standard libraries and a high degree of language expressivity. These features are valuable for experienced programmers, but for novices they tend to be a source of confusion especially when the students use online search engines to look for help.

2 Less-Java

We propose Less-Java, a simple language for novice-level programming [1]. In this section we describe the language itself and our reference compiler, which is implemented in Java and compiles Less-Java programs to Java code.

2.1 Language Design

Less-Java provides four native data types and three built-in collections. The four native data types are 32-bit signed integers, double-precision floating-point numbers, Booleans, and character strings. There is currently no way to explicitly cast a data type to another data type; however, operations like addition and subtraction implicitly widen an integer to a double when the two types are mixed. In addition to all the standard arithmetic operators, equality operators (==, !=) operate on all of these types. The three built-in collections are lists, sets, and maps. They can be initialized with a call to an appropriate constructor (optionally providing an existing variable to copy the elements) or by providing hard-coded data using initialization operators (e.g., brackets for lists). These collections are essentially wrappers around standard Java Collection classes.

Less-Java is strongly and statically typed but does not require explicit type declarations. Instead, the compiler assigns types to expressions based on context, a process called *type inference*. The underlying type system is composed of three kinds of types: variable, base, and object. The inference is implemented by a fixed-point process loosely based on Algorithm W for the Hindley-Milner type system [5].

The process begins with all non-literal expressions bound to a variable type (e.g., their type is unknown). These type bindings are refined iteratively using constraints derived from assignments and function calls. The constraints are resolved using a process called *unification*, which attempts to make two inferred types compatible. The type rules are as follows: 1) a variable type can be unified to any of the other types, 2) a base type can only be unified to the same base type, and 3) an object type can only be unified to the same object type. Literals can be immediately typed as base types, and constructor calls are typed according to the corresponding object type. If two types cannot be unified, the compiler reports a type error and compilation fails. The process terminates when expression types converge.

As an example, consider the assignment "a = 2.4". Initially, the variable "a" is bound to a variable (unknown) type, but the assignment constrains the type of "a" to be assignable from the right-hand side. Because the right-hand side is a literal and therefore a base type (double-precision number), the unification binds the type of "a" to the same base type. If "a" is later passed to a function, that function parameter will also be unified (and bound) to the same base type. If the assignment "a = true" appears later in the program, the unification will fail because the type of "a" is known to be a different base type than the literal "true" (which is a boolean).

If a function exposes parametric polymorphism (i.e., one or more of its parameters or its return cannot be resolved to a non-variable type), the compiler must generate multiple copies of the the function with concrete types. This is similar to how C++ handles template functions.

Finally, Less-Java includes simple unit testing using a built-in syntax. Unit tests are composed of the keyword "test" followed by any Boolean expression. During code generation, these statements are translated into JUnit test methods. This makes it easy for students to write tests or for an instructor to include tests in a project distribution.

2.2 Implementation

The Less-Java reference compiler is written in Java, which allows it to interface with the ANTLR parser generator [6, 7], facilitating parse tree traversals and code generation. Other development tools include Git/GitHub for version control and the Gradle build tool.

Lexing and parsing is handled by code that is automatically generated from the language grammar using ANTLR. After parsing the source code, the compiler converts the parse tree to an abstract syntax tree (AST) and performs several AST traversals to generate symbol tables and perform type inference. A final AST traversal generates Java target code.

During code generation, non-OOP constructs (top-level functions and unit tests) are translated to Java code in a Main class. Regular functions are emitted as public static methods with their Less-Java name and their inferred parameter/return types. Unit tests are emitted as JUnit test methods with the appropriate assertion.

After code generation, the generated Java source code must also be compiled with the Java compiler before it can be executed. The Less-Java compiler automates this process so that the user does not need to do it separately.

3 Results

3.1 Examples

The following is a simple example program in Less-Java:

```
add(a, b)
{
    return a + b
}

main()
{
    printf("4 + 5 is %d", add(4,5))
}

test add(1, -1) == 0
test add(2.5, 3.5) == 6.0
test add(1000, 1000) == 2000
```

Through a combination of implied boilerplate, simplified unit tests, and type inference, the Less-Java program is significantly shorter than equivalent Java code. Thanks to type inference, the add function doesn't need to be overloaded like it would be in Java. Instead, the function is considered generic, and a new copy of the target code is generated for each unique set of types (e.g., integers and doubles) at concrete function calls.

The following excerpt contains a more complex program, which is a computation related to the well-known Collatz conjecture [11]. This conjecture concerns a sequence where successive terms are obtained from previous terms

beginning with any positive integer. If the current term n is even, the next term is $n/2$. If the current term is odd, the next term is $3n + 1$. The Collatz conjecture posits that this sequence will always converge to 1. For our example, we wish to calculate the maximum sequence length within a (low, high) range of initial starting integers. The code demonstrates a range of language features, including functions, conditionals, loops, and unit tests. Additionally, it shows the conciseness and cleanness of the language.

```
// calculate the length of the Collatz sequence beginning at n
seq_len(n)
{
    len = 1
    while (n != 1) {
        if (n % 2 == 0) {         // even
            n = n / 2
        } else {                  // odd
            n = 3*n + 1
        }
        len = len + 1
    }
    return len
}

test seq_len(1) == 1
test seq_len(6) == 9

// find the maximum sequence length for starting values in the given range
max_3np1_seq(low, high)
{
    max = 0
    while (low <= high) {
        len = seq_len(low)
        if (len > max) {
            max = len             // new maximum
        }
        low = low + 1
    }
    return max
}

test max_3np1_seq(1,   10)   == 20
test max_3np1_seq(100, 200)  == 125
test max_3np1_seq(201, 210)  == 89
test max_3np1_seq(900, 1000) == 174
```

3.2 Preliminary Evaluation

We were unable to observe students using the language in a controlled study because of time constraints. However, we did conduct an informal experiment during a competitive programming club meeting. Students were tasked with solving a previously-approved list of problems in Less-Java (including the Collatz conjecture problem described above) without having had any prior exposure to the language. The problems were distributed as Less-Java files with some included unit tests so that the students could check themselves. A brief

demo program was posted on a projector for students to reference. The program contained many of the supported control structures, native functions, and syntax features of Less-Java. There was no formal data collection, but students were able to solve every problem with limited assistance from the language author and the club advisor. The students' solutions were reasonable in both length and complexity, and students seemed to appreciate the ease of testing.

4 Future Work

There are many avenues for future work. Some are mostly cosmetic, such as improving the error messages that the compiler produces, adding file I/O, and implementing IDE support.

More significantly, object-oriented programming in Less-Java is currently incomplete. While type inference is successful across assignments, it becomes unstable when objects are passed as function parameters. Overloading a function to handle the different parameters is insufficient for the object-oriented case; functions with many polymorphic parameters need to be emitted once for each combination of the parameters in the worst case. This cross-product property might generate unreasonably large compiled files, especially with deep inheritance hierarchies. One potential solution is to assign a list of interfaces to objects based on their methods, offloading the static analysis work to Java's interfaces. This would likely require a more extensive implementation of the Hindley-Milner type inference algorithm.

As mentioned in the previous section, an objective comparison study between Less-Java and some of the languages mentioned in the introduction would let us draw more significant conclusions in regards to language features and their impact on programming education.

Finally, the reference compiler for Less-Java has no optimization phase and the generated code hasn't been rigorously benchmarked against other languages. It should perform similarly to Java because we merely delegate most operations to the corresponding Java constructs, but a comprehensive performance benchmark may be able to expose some inefficiencies in the emitted code and address the question of whether an optimization phase in the Less-Java would help.

5 Conclusion

Mainstream programming languages remain suboptimal for introductory computer science education. Many are verbose, unintuitive, confusing, or some

combination thereof. To address this problem, we have presented Less-Java, a new programming language along with a reference compiler. Less-Java limits the constructs available to the programmer to avoid confusion and complexity while still providing all of the tools necessary to teach an introductory programming course, including a simple and concise syntax, implicit but strong typing via type inference, and built-in unit testing. This project also serves as a basis for a wide range of future work.

References

[1] Less-java. `https://github.com/zamua/less-java`. Accessed: 2018-07-24.

[2] Snap. `https://snap.berkeley.edu`. Accessed: 2018-07-24.

[3] Alzahrani, N., Vahid, F., Edgcomb, A., Nguyen, K., and Lysecky, R. Python versus c++: An analysis of student struggle on small coding exercises in introductory programming courses. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2018), SIGCSE '18, ACM, pp. 86–91.

[4] Black, A. P., Bruce, K. B., Homer, M., Noble, J., Ruskin, A., and Yannow, R. Seeking grace: A new object-oriented language for novices. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), SIGCSE '13, ACM, pp. 129–134.

[5] Milner, R. A theory of type polymorphism in programming. *Journal of computer and system sciences 17*, 3 (1978), 348–375.

[6] Parr, T. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[7] Parr, T., and Fisher, K. Ll (*): the foundation of the antlr parser generator. *ACM Sigplan Notices 46*, 6 (2011), 425–436.

[8] Pierce, B. C. *Types and programming languages*. MIT press, 2002.

[9] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch: Programming for all. *Commun. ACM 52*, 11 (Nov. 2009), 60–67.

[10] Somers, J. The coming software apocalypse. `https://www.theatlantic.com/technology/archive/2017/09/saving-the-world-from-code/540393/`, Sep 2017.

[11] Wikipedia contributors. Collatz conjecture — wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Collatz_conjecture&oldid=829100536, 2018. [Online; accessed 20-March-2018].

A Novel Events-First Approach for CS1 with Java*

Mark F. Russo
Department of Computer Science
The College of New Jersey
Ewing Township, NJ 08628
russom@tcnj.edu

Abstract

Events are integral to graphical user interfaces and other software components in modern software applications. Unfortunately, standard techniques for handling events in Java, such as implementing interfaces and overriding superclass methods, require a student to understand concepts not covered until later in the semester of a typical CS1 topic progression. This delays the ability of students to complete assignments involving event handling, such as responding to mouse interaction or a timer. This paper presents a novel simplified technique for associating student-authored methods with dispatched events in a Java program. This technique, which is based on Java *method references*, has been built into an open source object library designed for CS1. In addition to simplified event handling the library provides a variety of graphic and other classes that dispatch a range of events. The library allows programming assignments involving familiar event-based programming to be introduced comfortably within the first weeks of the semester, and it has been used successfully by hundreds of students at multiple institutions on all major operating systems.

1 Introduction

When students are introduced to a programming language for the first time they expect to create applications similar to the ones they use every day. A

frequent request of students in the first class of a typical college-level computer science curriculum (CS1) is to write a program that creates a window with a button that invokes a custom method. This often leaves the CS1 instructor providing unsatisfactory explanations to a disappointed student. When learning Java the barrier that prevents the introduction of this type of functionality stems from the complexity of using a graphical user interface framework that is almost always based on event dispatching. Techniques employed for handling dispatched events, such as implementing interfaces and overriding superclass methods, typically are not covered until later in the semester, thereby constraining lecture examples and student assignments to programs that run in the terminal.

To address this shortcoming, several efforts have been made to simplify the introduction of events in a CS1 class. For example, App Inventor [10] and Scratch [8] implement event handling in block-based languages by providing primitive event-handler "when" blocks that capture and respond to dispatched events. These specialized event handling blocks provide a simple and intuitive means by which to respond to events when a block language is appropriate. Christensen et al. [7] describe a presenter framework of prewritten Java applet superclasses which implement predefined widgets such as buttons and images. A student must author a subclass and override its methods to implement custom event handling behavior. Bruce et al. [4, 5, 3, 6] introduced an events-first approach in Java by developing a supporting library called ObjectDraw which also requires the student to subclass a WindowController superclass and override its methods to handle events. Events are dispatched by the window object only and not by individual graphic objects. Therefore, interacting with graphic objects drawn on the window must be simulated by computing whether or not mouse coordinates are within the bounds of the graphic object from within window event handler methods and tracking graphic object overlap to ensure the topmost object is the one impacted by dispatched mouse events.

This paper describes a novel simplified technique for implementing event handling in Java that avoids the need for the CS1 instructor to handwave away premature explanations of keywords like "extends" and "implements." Also covered is the way in which this technique is incorporated into an open source object graphics library called DoodlePad [9]. To complete assignments a student associates their own custom authored methods with a wide range of mouse, keyboard, timer and other events by passing a reference to a method as a parameter to one or more object methods that assign event handlers. Parameter passing is one of the first concepts introduced in CS1 and therefore can be used with confidence by students early in the semester. A variety of graphic and other classes is included in the library for students to use to complete their assignments. These classes include standard shape objects such

as Rectangle, Oval, RoundRect, Arc, Path, Line, and Polygon, as well as other objects such as Image, Sprite, Sound, Text and Pad. The event system that implements object-specific, pixel-perfect event dispatch is also described. In addition to the mouse, keyboard, timer and other events dispatched by the underlying window object, every shape object individually dispatches its own set of mouse events. This leads to a greater diversity of programming assignments that may be completed by CS1 students.

2 Method References

At the heart of the DoodlePad library's ability to implement event handling is *method references*, introduced in Java 8 [2]. Method references provide Java with an ability to reference a static or instance method, to pass that reference as a parameter to another method, to assign the reference to a variable, and to invoke the referenced method. Method references are constructed using the newly introduced "::" operator which scopes a method name by an object instance or class. The example in Listing 1 illustrates how a student may create a Rectangle object and arrange for a custom method to be invoked when the mouse is pressed on the Rectangle. The showMsg() method is associated with the Rectangle object's mousePressed event by passing the method reference "Example1::showMsg" as a parameter to the Rectangle object's setMousePressedHandler() method. This is a complete program; no other configuration is necessary.

Listing 1: A program that creates a Rectangle and associates an event handler.

```
import doodlepad.*;

public class Example1 {

  // Static method invoked when mouse is pressed on the Rectangle
  public static void showMsg(Shape shp, double x, double y, int btn)
  {
    System.out.println("The␣Rectangle␣was␣pressed!");
  }

  // Create a Rectangle object and associate an event handler
  public static void main(String[] args)
  {
    Rectangle r = new Rectangle();
    r.setMousePressedHandler( Example1::showMsg );
  }
}
```

Figure 1 shows the command for compiling and running the program as well as the resulting graphic window displaying the Rectangle object. When the

Rectangle is clicked, a message is printed to the terminal. Note that the student does not need to create a window explicitly. If a Pad object (the window class in DoodlePad) is not created by the student, the library creates one automatically. This allows the student's focus to remain on learning concepts of object oriented programming without being distracted by implementation details of the underlying windowing toolkit.

The Pad class also provides methods that allow students to invoke their own custom methods in response to an even wider range of events. Table 1 lists events dispatched by all DoodlePad Shape subclasses and the Pad class. In all cases, method references may be used to associate student-authored methods with one or more dispatched events.

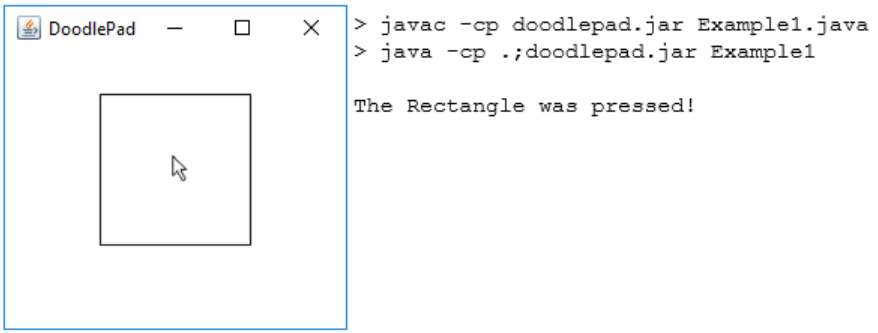


Figure 1: A Pad displaying a Rectangle with terminal commands and output.

3 Pixel-Perfect Mouse Event Dispatching

A novel feature of DoodlePad is the way in which the library selects mouse event targets. Event targets are chosen from potentially multiple overlapping Shapes as well as a Pad object, with pixel-perfect accuracy. For example, a DoodlePad program was written that creates a Pad object and a Polygon object that forms a V-shape. (See Figure 2) The mouseEntered, mouseMoved and mouseExited events of both the Pad and Polygon were associated with methods that print a message indicating the object and event that invoked the method. When the mouse is moved along the path traced by the dashed arrow in Figure 2 the statements in Listing 2 are printed to the terminal (duplicates removed). Note how the order of events handled carefully follows the trajectory, even when the mouse moves off and then back on the Polygon near the center of the Pad.

To accomplish this pixel-perfect Shape object targeting the library implements a sophisticated technique for identifying the appropriate target Shape

Table 1: Events dispatched by Shape subclass objects and the Pad object.

Event Name	Shape	Pad	Description
mouseClicked	√	√	Clicked on an object
mouseDoubleClicked	√	√	Double-clicked on an object
mousePressed	√	√	Mouse pressed down on an object
mouseReleased	√	√	Mouse released on an object
mouseMoved	√	√	Mouse moved over an object
mouseDragged	√	√	Mouse moved while pressed over an object
mouseEntered	√	√	Mouse moved into Shape or Pad window
mouseExited	√	√	Mouse moved out of Shape or Pad window
selectionChanged	√		The selected state of a Shape changed
keyPressed		√	A key is pressed on an activated Pad
keyReleased		√	A key is released on an activated Pad
keyTyped		√	A key is pressed and released on a Pad
tick		√	A Pad's timer tick event occurred
serverStarted		√	A Pad's socket server has started
serverStopped		√	A Pad's socket server has stopped
serverInfo		√	A Pad's socket server has information
serverError		√	An error occurred with a socket server
clientOpened		√	A client socket connection has opened
clientClosed		√	A client socket connection has closed
clientReceived		√	A client socket has received data
clientInfo		√	A client socket has information
clientError		√	An error occurred with a client socket

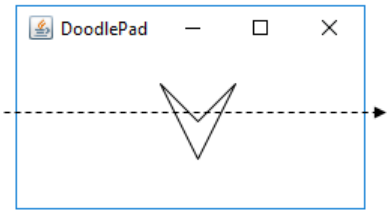


Figure 2: A Polygon shape drawn on a Pad with event handlers.

under the mouse, which is illustrated in Figure 3. Each time a shape is drawn on a Pad a second Shape is drawn on an in-memory BufferedImage object that mirrors the visible region of the Pad. The in-memory drawing differs in that shapes are drawn with a unique color for each shape's fill and stroke and with antialiasing off. After a shape is drawn on the in-memory BufferedImage its unique color is used as the key in a Map data structure that references the cor-

responding Shape object on the Pad. When mouse events occur on the window the pixel color under the identical location on the BufferedImage is read and used to look up the Shape object in the Map that should dispatch events. If no Shape object is found in the Map, mouse events are dispatched by the Pad object.

Listing 2: Output from moving the mouse along the dashed arrow in Figure 2.

```
Pad     : mouseEntered
Pad     : mouseMoved
Polygon : mouseEntered
Pad     : mouseExited
Polygon : mouseMoved
Pad     : mouseEntered
Polygon : mouseExited
Pad     : mouseMoved
Polygon : mouseEntered
Pad     : mouseExited
Polygon : mouseMoved
Pad     : mouseEntered
Polygon : mouseExited
Pad     : mouseMoved
Pad     : mouseExited
```

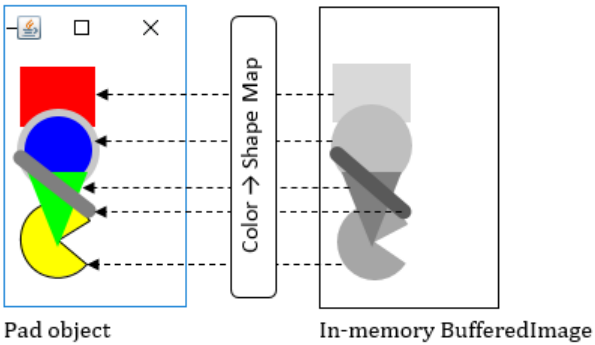


Figure 3: A Map associates unique colors of shapes drawn on an in-memory BufferedImage with Shape objects drawn on the Pad.

Using this technique, the correct target Shape object is properly identified and the event is dispatched with pixel-perfect accuracy. Students are not aware of the underlying implementation unless they choose to inspect the library source code in GitHub [9]. The result is an environment in which events are dispatched in a natural manner and therefore requires little or no explanation for students to fully master.

4 Event-Driven Assignment Examples

The enhanced event framework implemented by DoodlePad combined with its graphic and other classes expands the range and type of programming assignments that are within the skillset of students in a CS1 level class. To illustrate, following are a few sample program descriptions that have been created using DoodlePad with an emphasis on the use of its unique event system. Source code for example programs described here is available from the author upon request.

4.1 Board Games

DoodlePad comes with a built-in ability to interactively drag Shape objects on a Pad by changing a setting. This lets students build simple board games by creating Shape objects, and locating and styling them appropriately. For example, a checkers game may be created with Rectangle and Oval Shapes, where Ovals are configured to be dragged as game pieces.

4.2 Color Matching

A classic color matching game is constructed by defining a Spot class that subclasses Oval or Rectangle and adds a property with private visibility that holds a randomly selected hidden fill color. Custom Spot methods show and hide a Spot's fill color. The program starts by creating Spot objects positioned on a Pad in a regular grid. A mouseClicked event handler method for each Spot changes fill color to its hidden value. After the color of a second Spot is revealed by clicking, if the colors do not match the program pauses for a brief period of time using the Timer and then hides the colors again. When the two colors match, the Spot objects are made invisible to signify that they have been solved.

4.3 Interactive Puzzles

The keyPressed Pad event and a switch statement may be used to incrementally move a Shape object on a Pad using the arrow keys. Couple this with a recursive maze generation algorithm and students have the makings of an interactive maze puzzle.

4.4 Interactive Drawing

The Image class's ability to set individual pixel colors coupled with its own mouseDragged event is all that is necessary to create a rudimentary interactive

drawing program. Students use other Shape classes to form simple buttons that respond to mouseClicked events by setting current drawing properties such as brush color and size.

4.5 Interactive Data Visualization

Data set visualizations make interesting projects. Shape size, position and color may be used to represent data set attributes. Visualizations are made interactive using the mouseEntered and mouseExited Shape events. For example, moving the mouse over Polygons representing the 50 United States may reveal additional details by making visible associated explanatory text. As the mouse exits a Polygon, details are hidden.

4.6 Remote Collaboration

The Pad class has the ability to act as a socket client or a socket server. Using this functionality two Pad objects may establish a network connection and then exchange text messages. Received messages trigger a clientReceived event which the student can handle and translate into suitable actions. Messages are sent to remote Pad objects using the send() and broadcast() methods. An example program that uses this ability to communicate include an implementation of the classic "Battleship" guessing game.

5 Conclusion

A novel simplified technique for handling events has been built into a library of graphic and other classes designed for expanding the number and type of assignments that may be introduced in CS1. The event handling technique depends on a new language feature introduced in Java 8 called method references [2]. This library, called DoodlePad [9], is open source and freely available under a GPL3 license [1]. It has been used by hundreds of students at multiple institutions on Linux, Mac and Windows operating systems and has proven to be a solid platform in all cases. Refer to `http://doodlepad.org` for further instructional materials, examples, links to source code and complete Javadocs.

References

[1] GNU general public license. `http://www.gnu.org/licenses/gpl-3.0.html`.

[2] The java™tutorials. `https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html`.

[3] Bruce, K., Danyluk, A., and Murtagh, T. Event-driven programming is simple enough for cs1. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education* (New York, NY, USA, 2001), ITiCSE '01, ACM, pp. 1–4.

[4] Bruce, K., Danyluk, A., and Murtagh, T. A library to support a graphics-based object-first approach to cs1. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education* (New York, NY, USA, 2001), SIGCSE '01, ACM, pp. 6–10.

[5] Bruce, K., Danyluk, A., and Murtagh, T. Event-driven programming facilitates learning standard programming concepts. In *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (New York, NY, USA, 2004), OOPSLA '04, ACM, pp. 96–100.

[6] Bruce, K., Danyluk, A., and Murtagh, T. *Java: An Eventful Approach*. Pearson Education, Upper Saddle River, NJ, 2006.

[7] Christensen, H., and Caspersen, M. Frameworks in CS1 – a different way of introducing event-driven programming. In *Proceedings of the 7th annual conference on Innovation and technology in computer science education* (New York, NY, USA, 2002), ITiCSE '02, ACM, pp. 75–79.

[8] Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., and Kafai, Y. Scratch: Programming for all. *Communications of the ACM 52*, 11 (2009), 60–67.

[9] Russo, M. DoodlePad: Next-gen event-driven programming for CS1. *Journal of Computing Sciences in Colleges 32*, 4 (2017), 99–105.

[10] Turbak, F., Sherman, M., Martin, F., Wolber, D., and Pokress, S. Events-first programming in APP Inventor. *Journal of Computing Sciences in Colleges 29*, 6 (2014), 81–89.

Using Computer Programming Activities and Robots to Teach Generalization of a Geometry Concept*

Mark G Terwilliger, Jay L Jackson, Cynthia L Stenger, and James A Jerkins
Computer Science and Information Systems Department
Mathematics Department
University of North Alabama
Florence, AL 35632
{mterwilliger,jljackson3,clstenger,jajerkins}@una.edu

Abstract

Computer science and math education researchers have long believed that a symbiotic relationship exists between their disciplines [7]. In fact, in its early days, computer science education programs were often co-located in a math department. Stenger et al. [13] developed an instructional treatment that uses computer programming as an explicit method for teaching abstraction and generalization in the STEM classroom. This instructional strategy uses computer programming to explore the essential characteristics of a mathematical concept and to push learners to advance in levels of abstraction. In this study, results are shown from a professional learning session using computer programming activities, mathematical arguments, and programming on an S2 robot to push middle and high school computer science, math, and science teachers (N=25) to improve their level of generalization over area expansion of a triangle with respect to the expansion of the sides of the triangle. The programming activities served as a laboratory to expose and explain what happened in the minds of learners as they explored and learned to generalize this geometry concept. The researchers used an initial genetic decomposition to evaluate the learner's level of abstraction. Follow up interviews

were conducted with 6 participants. The analysis, using APOS as a framework, categorized mathematical behaviors at the Action, Process or Object level. The data demonstrated how computer programming activities influenced teachers' mental images and pushed them to higher levels of abstraction.

1 Introduction

The ability to generalize is an essential skill for the sciences. Existing literature supports the idea that mathematics courses, such as discrete mathematics and calculus, help to develop these skills [6]. The belief exists among math and CS education researchers that computer programming pushes the learner to higher levels of abstraction [13]. Specifically, many STEM education researchers believe that using computer programming activities designed to parallel the construction of an underlying mathematical process may stimulate or accelerate the development of the associated mathematical construction [5, 8].

Stenger et al. [12] have developed an explicit method for motivating students to generalize mathematical concepts into constructions using computer programming exercises and proof writing, based on the theoretical perspective of APOS theory. Others have also used computer programming to fortify mathematical concepts and as a tool to assist in abstraction [10, 11]. When students write computer programs to show mathematical patterns, the generalizations of those patterns appear in the code, strengthening students' conceptual knowledge and improving their ability to generalize. This particular study investigates whether computer programming activities regarding the perimeter and area of triangles influence students' mental constructions toward higher levels of abstraction in this subject area.

2 Theoretical Framework

The theory of reflective abstraction was described by Piaget [9] as a two-step process, beginning with reflection on one's existing knowledge, followed by a projection of one's existing knowledge onto a higher plane of thought. Further, Piaget [9] and Dubinsky et al. [1] wrote that during the process of cognitive development, reflective abstraction could lead to the construction of logico-mathematical structures by the learner. The conviction that reflective abstraction could serve as a powerful tool to describe the mental structures of a mathematical concept led Dubinsky to develop APOS theory.

In APOS theory, the mental structures are Action, Process, Object, and Schema. A mathematical concept develops as one acts to transform existing physical or mental objects. Actions are interiorized as Processes and Processes

are encapsulated to mental Objects, but APOS practitioners hold that learners move back and forth between levels and hold positions between and partially on levels. This nonlinear behavior and the resulting mental structures may explain the different ways learners respond to a mathematical situation [1].

In this study, a genetic decomposition was developed as a conjecture of the mental constructions – Actions, Processes, and Objects – that may describe the construction of mental schema for the concept of geometry as it develops in the mind of the learner. The genetic decomposition motivated the design of this research study as well as the analysis of the results. It was also the basis for the computer activities in the lessons that were developed. The pervasive impact of the genetic decomposition is consistent with an APOS theoretical framework [1].

3 Instructional Treatment Overview

Stenger et al. [8] developed an explicit approach to teaching abstraction and generalization in the mathematics classroom using computer programming exercises. The instructional treatment is grounded in APOS theory and considers the mental processes by which abstract concepts are acquired and utilized in mathematics [4]. Dubinsky is an advocate of students writing computer programs where the constructs in the program parallel the constructs of a mathematical topic under investigation. By using computer programming exercises where the programming activity specifies the procedure for the computer, the student is motivated to reflect upon the enactment of the concept. Dubinsky states that the act of programming is a generic process which carries out what may be viewed as a more general construct in specific cases and induces the student to move towards a generic abstraction of the concept [5]. The instructional treatment is built upon this notion that programming is a vehicle for building abstractions in the mind of the learner. Numerous researchers in APOS theory have shown that computer programming is a viable tactic for teaching a variety of topics in undergraduate mathematics [2]. Consequently, it is commonly held that computer constructions are an intermediary between concrete objects and abstract entities [1, 3].

4 Methodology

The researchers applied the instructional treatment to the concept of area expansion in triangles. The investigation was carried out with 25 middle and high school teachers in a 2 week professional development. These instructors taught computer science and math (80%), science (16%), and English (4%). Nineteen

of the teachers were female and six were male. Earlier in the training week, teachers had investigated whether, given three side lengths, the combination of those lengths would form a triangle. This current lesson built on that material. Each subject participated in a complete lesson including the pre-test, response sheets, and post-test. The format of the lesson was as follows. A brief introduction to the Python programming environment was given along with the code template in Figure 1.

PROGRAM:

```
a=30
b=50
c=70
print ("a", "b", "c", "Perim", sep="\t")
perim = a+b+c
print (a, b, c, perim, sep="\t")
perim = 2*a+2*b+2*c
print (2*a, 2*b, 2*c, perim, sep="\t")
```

OUTPUT:

```
a     b     c     Perim
30    50    70    150
60    100   140   300
```

Figure 1: Computer Programming template for lesson

A brief lesson on how to determine the area of a triangle after knowing only three sides was discussed and included the derivation of Heron's formula. Learners wrote programs to derive the perimeter and the area of a triangle after inputting three side lengths. Learners were encouraged to experiment with their computer programs and make observations about any relationships. Once this initial table was constructed, the participants were ushered through a series of program modifications and written responses. For example, they were asked to add rows to their programs to show how perimeter and area were affected when side lengths were multiplied by a particular factor, such as doubling, tripling, and halving the lengths of the sides. Written responses to questions and reflections on their observations were recorded by the participants on their response sheets including generalizations of behavior. Observations on the relationship between side length and perimeter/area were solicited as general expressions and participants were taught how to denote the general expressions in mathematical language. For example, participants might observe that if the side lengths were tripled, then the perimeter would also triple, or that if the side lengths were doubled, then the area would increase by that factor (2, in this case) squared. Later, instruction was provided to show how to calculate area when two sides and an included angle of a triangle are known. After this, an additional column was added to the program which allowed participants to input two side lengths and an included angle and calculate the

area of the resulting triangle. The instructional treatment was designed so that repetition with various program modifications would stimulate the desire to generalize the observed behavior and make conjectures about the mathematical construct. The final stage of the lesson involved making conjectures and convincing arguments. Participants were shown how to use general expressions to support, or refute, a conjecture using mathematical language. They were then asked to attempt their own convincing arguments with the general expressions they recorded during their inquiry. As a part of the instructional strategy, an activity is used to further push the learner to encapsulate the process to an object.

In this project, we had the participants program Parallax Scribbler 2 (S2) robots in order to continue exploring the relationship between the triangle sides and area. All of the participant's responses were collected on response sheets during the lesson. Additional data was collected in the form of interviews. Recorded interviews with five of the participants were transcribed and analyzed. All of the collected data was reviewed and scored using APOS theory. A ranked set of scores was devised to denote pre-action, action, process, and object levels based on the genetic decomposition and recorded for each subject's pre-test, response sheets, post-test, and (where applicable) interview data. In the event that authors disagreed, a discussion and further analysis of the data was used to reach consensus.

5 Results

The ratings prior to and after instruction showed that 23 out of 25 teachers started at the pre-action or action level of abstraction. Twelve teachers were rated at the process or object level after the instruction.

6 Influence of Computer Programming on Generalization

The influence of writing computer programs was demonstrated by teachers on the Python and robot activity response sheets. In the Python activity, participants began with a small program as shown in Figure 1. You can see in this example, the sides of a given triangle are doubled and the sides of a new triangle and its corresponding perimeter are displayed. Participants added a third row in which the triangle sides tripled. A new column was added to each row to display the area for each triangle. As the participants were writing these programs, they observed patterns and began coming up with general expressions on their own that helped clarify the geometric relationships being

studied. Teacher T0010 modified his program to allow the user to type in a scale factor and used this in the general formulas for perimeter, area, and angle measures. He added instructions at the end of the table that displayed the ratios for perimeter and area of the scaled triangle over the original triangle. Sample output is in Figure 2.

```
Enter side 1:  12
Enter side 2:  14
Enter side 3:  20
Enter your scalar:     3
a       b       c       perim   area    Angle A
12.0    14.0    20.0    46.0    82.65   36.18
36.0    42.0    60.0    138.0   743.85  36.18
The new perimeter is 3.0 times the original perimeter
The new area is 9.0 times the original area
```

Figure 2: Computer program T0010

In the follow-up interview, T0010 described how he developed a mindset of generalizing during the process of writing code: "The example of 30 50 70 made it hard to see how the areas and perimeters are related without a calculator. Who knows what 649.4 x 4 or 649.9 is right off the top of their head? I generated the lines at the end to compare the initial and the scaled triangle. We were looking to make generalizations in order to form a conjecture. We did two and three, but what about 100, 500 or .025? I know that you could change the code, but after you get the math, I like to generalize the code. I found it more difficult to see the relationship between the first (scalar of 1) and last row (scalar of 3) when looking at them. I thought about putting a loop for multiple scalars, but did not feel it was necessary. I compared one original to one scalar." Teacher T0006 noticed the pattern and immediately added a loop using the generalized formula she observed. She iterated a loop over the triangle scale factor and then added a column to display that scale factor, along with the ratio of the scaled triangle over the original triangle. Sample output is shown in Figure 3.

```
Length of side a 14
Length of side b 8
Length of side c 12
a       b       c       AngleA Perim   Area     S-Fac A-Fac
14.0    8.0     12.0    86.4   34.0    47.9     1
28.0    16.0    24.0    86.4   68.0    191.6    2     4.0
42.0    24.0    36.0    86.4   102.0   431.2    3     9.0
56.0    32.0    48.0    86.4   136.0   766.5    4     16.0
70.0    40.0    60.0    86.4   170.0   1197.7   5     25.0
```

Figure 3: Computer program T0006

In the follow-up, T0006 discussed what prompted generalization while cod-

ing: "I assumed that we were going to work toward using a loop to generate a table as we had been doing..., so I thought I was working ahead. Plus, I'm naturally inclined in all areas of my life to see what I can make a computer do so that I don't have to do it myself. That same mindset came into play here as well. I didn't have any instructional motivation until I added the columns to show the ratios of the perimeter and then area to the factor multiplied by each side. I thought having those columns would help students see the relationship more easily."

7 Influence of Robot Activity on Generalization

An activity was used to push the learner to encapsulate the process to an object. In this project, we had the participants program Parallax Scribbler 2 (S2) robots in order to continue exploring the relationship between the triangle sides and area. Initially, the participants were given an 18x24-inch poster with a triangle printed on it. Their goal was to have the robot trace the triangle by writing code using the Spin programming language. A primer was given that allowed the participants to write programs with just two commands: (1) turn_by_deg(deg), which rotated the robot deg degrees counterclockwise, and (2) go_forward(d), which moved the robot straight ahead by d units, where a unit is 0.5 millimeters. After drawing a triangle on the poster, participants then modified their program to have the robot draw a similar, smaller triangle on an 8.5x11-inch sheet of paper. Measurements were taken of the two similar triangles and observations were made about the ratios between the side lengths, perimeters, and areas. Although a scale factor was not suggested, it turned out that 11 of the 25 participants used a factor of ten. Other participants used a scaled factor of 2. Participants that successfully completed the robot activity demonstrated higher APOS levels when tested immediately after the activity. Four of the thirteen (30.7%) at Pre-Action and Action level successfully completed the robot activity while ten of the twelve (83.3%) at Process or Object level did.

8 Conclusion

The Python and robot programming activities motivated students to generalize, influenced their mental images, and pushed them to higher levels of abstraction. Some teachers who started at the action or process level were able to transition to more general triangle properties and to use these in their computer programs. Teachers naturally turned to their computer programs to help them generalize over the concept. The programming activities influenced

students and served as a catalyst to move from purely English descriptions of their conceptions to using mathematical symbols, and the activities put them into a mindset of generalizing. The results of this study can facilitate further analysis of using computer programming to overcome cognitive obstacles in teachers' understanding of geometry.

8.1 Acknowledgements

This work is supported by a U.S. Department of Education Math/Science Partnership grant administered by the Alabama State Department of Education.

References

[1] Arnon, I., Cottrill, J., Dubinsky, E., Oktaç, A., Fuentes, S., Trigueros, M., and Weller, K. *APOS Theory: A Framework for Research and Curriculum Development in Mathematics Education.* Springer New York, 2013.

[2] Asiala, M., Kleiman, J., Brown, A., and Mathews, D. The development of students' understanding of permutations and symmetries. *International Journal of Computers for Mathematical Learning 3*, 1 (1998), 13–43.

[3] Dubinsky, E. Writing programs to learn mathematics. `http://www.math.kent.edu/~edd/ICMITechPpr.pdf`.

[4] Dubinsky, E. The cognitive effect of computer experiences on learning abstract mathematical concepts. *Korkeakoulujen Atk-Vutiset 2* (1984), 41–47.

[5] Dubinsky, E., and Tall, D. Advanced mathematical thinking and the computer. In *Advanced mathematical thinking.* Springer, 2002, pp. 231–248.

[6] Epperson, F. Agile methods and interaction design: Friend or foe? In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (New York, NY, USA, 2009), EICS '09, ACM, pp. 209–210.

[7] Jenkins, J., Jerkins, J. A., Stenger, C., and Stovall, J. The Influence of Stereotypes on STEM majors in a Problem-Based Curriculum Investigation: Computer Science vs. Mathematics Majors. In *Proceedings of the 53rd Annual ACM Mid-Southeast Conference* (2011).

[8] Jenkins, J. T., Jerkins, J. A., and Stenger, C. L. A plan for immediate immersion of computational thinking into the high school math classroom through a partnership with the alabama math, science, and technology initiative. ACM-SE '12, ACM, pp. 148–152.

[9] Piaget, J. The equilibration of cognitive structures: The central problem of intellectual development (t. brown & kj thampy, trans.). *Chicago: The University of Chicago* (1985).

[10] Sigurdson, N., and Petersen, A. Student perspectives on mathematics in computer science. In *Proceedings of the 17th Koli Calling Conference on Computing Education Research* (2017), ACM, pp. 108–117.

[11] Simonot, M., Homps, M., and Bonnot, P. Teaching abstraction in mathematics and computer science-a computer-supported approach with alloy. In *CSEDU (2)* (2012), pp. 239–245.

[12] Stenger, C., Jerkins, J. A., Jenkins, J., and Stovall, J. Using Computer Programming to teach Mathematical Generalization and Proof. In *Proceedings of the 13th International Congress on Mathematical Education (ICME-13)* (2016), International Commission on Mathematical Instruction.

[13] Stenger, C., Jerkins, J. A., Stovall, J., and Jenkins, J. An APOS Study on Undergraduates' Understanding of Direct Variation: Mental Constructions and the Influence of Computer Programming. In *Proceedings of the 21st Annual Conference on Research in Undergraduate Mathematics Education (MAA SIGMAA on RUME 2018)* (2018).

Cross-Lingual Genre Classification using Linguistic Groupings*

Hoang Nguyen and Gene Rohrbaugh
Department of Computer and Information Science
Messiah College
Mechanicsburg, PA 17055
{hn1174,grohrbau}@messiah.edu

Abstract

In this paper we present a method for classifying texts by genre using machine learning applied in multilingual corpora. We show that by training on languages for which training data is readily available, a stable classification model can be built that effectively classifies documents in languages for which training data is not available. Using two distinct languages in the training phase results in better accuracy than using a single language, while additional languages provide no benefit. Finally, we show that language relatedness – whether the training languages are related to the testing language – does not impact accuracy.

1 Introduction

Knowing the genre of a document (editorial, academic journal article, news release) can facilitate a variety of tasks including part-of-speech tagging, statistical machine translation[8], and automated text summarization[3]. Since such genre information may not be explicitly encoded in documents, there is a need for automated genre classification.

Prior studies have demonstrated genre classification is beneficial to topic labeling[4], sentiment analysis[7], authorship detection[1], and language function[10]. Our research focuses on building a stable cross-lingual genre classification model applicable to languages for which little training data exists,

referred to as less commonly known languages (LCKL). We explored the effect of language relatedness and the number of languages used in training on the success of the resultant model.

1.1 Background

A variety of genre classification approaches have been used since the 1990s, including discriminant analysis based textual parameters readily available from part-of-speech tagging and syntactic analysis[5], and the "bag-of-words" approach[2]. These studies, however, worked with monolingual corpora, and specifically languages for which large annotated datasets already existed.

Rather than focusing on one single language, cross-lingual genre classification (CLGC) builds genre classification models across multiple languages. Semi-supervised learning methods such as co-training[12] or domain adaptation methods have been explored. However, these methods are still heavily dependent on statistical machine translation, which is realistically unavailable to low-resources languages and costly in terms of memory usage and storage in multilingual settings.

1.2 Project Goals

Our research focuses on improving genre classification prediction for low-resourced languages by using annotated data from well-resourced languages without resorting to machine translation or part-of-speech tagging. We focused on answering two crucial questions: (1) Does training on related languages improve the predictive ability of the model? (2) What is the optimal number of languages needed to build a stable cross-lingual genre classification model for LCKLs?

2 Methodology

We used two multilingual parallel corpora: the Europarl corpus, which contains parlimentary proceedings in 21 European languages; and JRC Acquis, which comprises legal documents translated into 23 different languages[9]. To ensure the integrity of the dataset, we consider only original language texts, excluding translated texts. Since the same number of text documents exist in each language, we randomly selected the text documents from the equally distributed text document corpus[8].

Our dataset included 15 languages in three groups: Slavic, Germanic, and Romance. Within each group, the language with the fewest documents was

identified as the LCKL and set aside as testing data, and the remaining languages were used to train the classification model. As shown in Figure 1, Danish (da) was the LCKL for the Germanic group, which also included English (en), German (de), Dutch (nl), and Swedish (sv). The Romance group had Romanian (ro) as LCKL, along with French (fr), Italian (it), Spanish (es), and Portuguese (pt). Finally, for Slavic, the LCKL was Bulgarian (bg) with Polish (pl), Czech (cs), Slovak (sk) and Slovenian (sl) as the remaining languages.

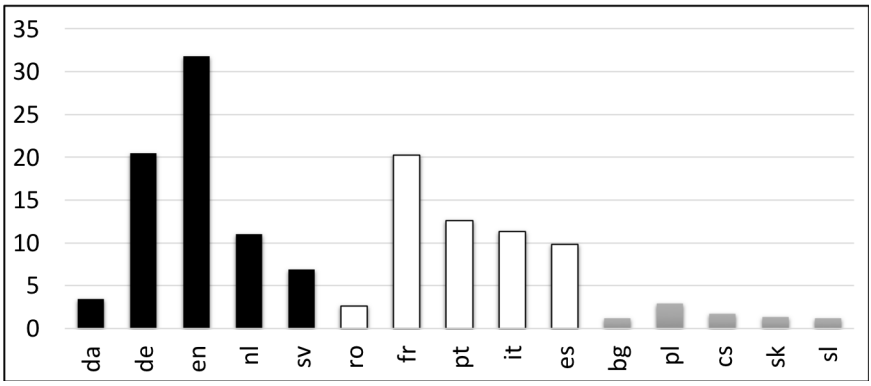


Figure 1: Document Count by Language (Thousands)

2.1 Data Processing

We utilized the Punkt algorithm[6] from NLTK[11] framework to extract 15 linguistic features from individual text documents. The features chosen were identified by Petrenz & Webber[8] as being reliable indicators of genre, and included measures such as colon frequency, document length, sentence mean length, and single-sentence paragraph count. These became the input features for a Support Vector Machine (SVM) genre classification model, implemented from the scikit-learn library with Linear Kernel.

In order to compare the effects of in-group and out-group languages on each of the LCKLs, we performed genre classification with given features and analyzed the confusion matrix generated by the models. Different metrics including accuracy, precision, recall, and F-1 measure were calculated based on the confusion matrix. This paper presents the accuracy measures, but we found that in every case, the F-1 measure followed a similar pattern.

To identify the optimal number of training languages, we used training datasets including from one to twelve languages. The sequence was determined by the number of documents available in each language. Starting with

the language with the most documents, we added languages one by one to the training dataset until it contained 12 languages (excluding all of the LCKLs). By finding a peak in accuracy, we can determine the optimal number of languages to use in training.

3 Results

As shown by Figure 2, within the Slavic group, training on other Slavic languages resulted in higher accuracy than training on non-Slavic languages, regardless of the number of languages used for training. Surprisingly, the Romance group showed the opposite trend: training on non-Romance languages gave better results than training on Romance languages (see Figure 3).

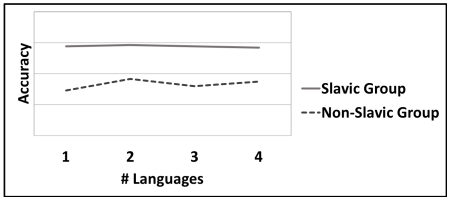


Figure 2: Slavic Accuracy

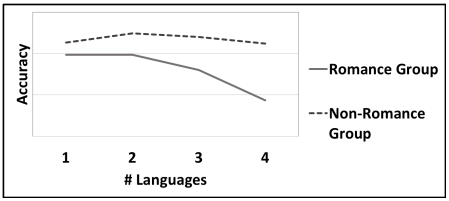


Figure 3: Romance Accuracy

Finally, within the Germanic group, performance of Germanic versus non-Germanic training ultimately depended on the number of training languages (see Figure 4). For all three language groups, the Europarl F-1 measure and the JRC F-1 measure show the same trend as accuracy.

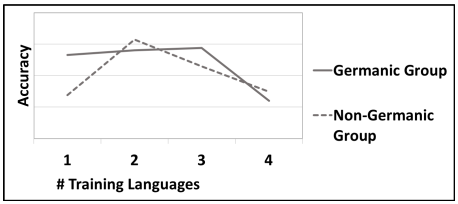


Figure 4: Germanic Accuracy

Regarding the question of how many training languages would be optimal, Figure 5 shows that the highest accuracy values occurred when the training dataset contained two languages.

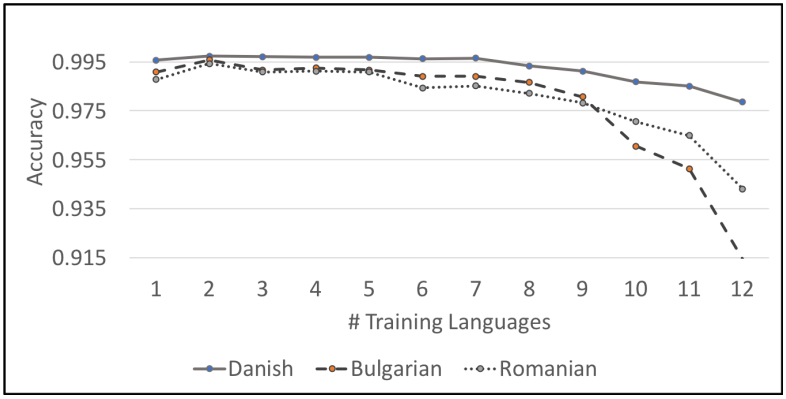


Figure 5: Training Languages

4 Conclusions

Our research did not reveal a consistent trend in the relationship between training on related versus less-related languages. Slavic performed better with related languages; Romance performed better with less-related languages; and Germanic showed no consistent trend. Further research regarding effects of interactions between languages within each linguistic grouping is necessary so that a more thorough understanding of the effects of linguistic relatedness. A more significant result of the research was identifying the optimal number of training languages, ignoring relatedness. Specifically, we found that training with two languages provides the most robust genre classification across linguistic groups.

References

[1] De Vel, O., Anderson, A., Corney, M., and Mohay, G. Mining e-mail content for author identification forensics. *ACM Sigmod Record 30*, 4 (2001), 55–64.

[2] Freund, L., Clarke, C., and Toms, E. Genre classification for ir in the workplace. *Information Interaction in Context, Copenhagen, Denmark* (2006).

[3] Goldstein, J., Ciany, G. M., and Carbonell, J. G. Genre identification and goal-focused summarization. In *Proceedings of the sixteenth*

ACM conference on Conference on information and knowledge management (2007), ACM, pp. 889–892.

[4] Hingmire, S., and Chakraborti, S. Topic labeled text classification: a weakly supervised approach. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval* (2014), ACM, pp. 385–394.

[5] Karlgren, J., and Cutting, D. Recognizing text genres with simple metrics using discriminant analysis. In *Proceedings of the 15th conference on Computational linguistics-Volume 2* (1994), Association for Computational Linguistics, pp. 1071–1075.

[6] Kiss, T., and Strunk, J. Unsupervised multilingual sentence boundary detection. *Computational Linguistics 32*, 4 (2006), 485–525.

[7] Melville, P., Gryc, W., and Lawrence, R. D. Sentiment analysis of blogs by combining lexical knowledge with text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (2009), ACM, pp. 1275–1284.

[8] Petrenz, P., and Webber, B. Robust cross-lingual genre classification through comparable corpora. In *The 5th Workshop on Building and Using Comparable Corpora* (2012), p. 1.

[9] Steinberger, R., Pouliquen, B., Widiger, A., Ignat, C., Erjavec, T., Tufis, D., and Varga, D. The jrc-acquis: A multilingual aligned parallel corpus with 20+ languages. *arXiv preprint cs/0609058* (2006).

[10] Wachsmuth, H., and Bujna, K. Back to the roots of genres: Text classification by language function. In *Proceedings of 5th International Joint Conference on Natural Language Processing* (2011), pp. 632–640.

[11] Wagner, W. Steven bird, ewan klein and edward loper: Natural language processing with python, analyzing text with the natural language toolkit. *Language Resources and Evaluation 44*, 4 (2010), 421–424.

[12] Wan, X. Co-training for cross-lingual sentiment classification. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-volume 1* (2009), Association for Computational Linguistics, pp. 235–243.

Revisiting an Educator's Dilemma: Using Natural Language Processing to Analyze the Needs of Employers and Inform Curriculum Development*

Nathan Green, Xiang Liu, and Diane Murphy
Marymount University
Arlington, VA, 22207
{ngreen,xliu,dmurphy}@marymount.edu

Abstract

Employers in technology are constantly changing their job advertisements and it is a challenge for academic programs to produce graduates who can meet these job requirements and employer expectations. The Information Systems (IS)/Information Technology (IT) educators have been facing the conundrum of whether an emerging technology is a "game-changing" development or something more transient so as to avoid "bloating" the curriculum. This study examines the body of knowledge as represented in our IT/IS program course syllabi and the recent job postings in five of our specialization areas including data science, computer science, healthcare IT, information systems and cybersecurity using natural language processing techniques. One of our goals is to identify the major overlaps and gaps between the two entities systematically by employing quantitative methods. The major contributions of this study lie in that it demonstrates how such data-driven analysis and mining approach informs clarifications to the wording of existing course syllabi, modifications to existing course contents, or the introduction of new courses into the curriculum. Lastly, the future research directions are delineated which this knowledge base can be applied to enhance university graduates' employability by analyzing students' resumes and presenting which jobs most closely match their knowledge, skills, and abilities.

1 Introduction

College graduates entering the workplace today are expected to be equipped with a gamut of skills including technical, subject matter-related skills along with practical experience and essential competencies like problem-solving and leadership [13]. However, there have been growing concerns among business leaders across industries, policymakers as well as higher education administrators and educators about the skills gap in the current workforce [11].

Moreover, a multitude of latest surveys and studies on the job vacancies and workforce readiness corroborate such concerns. For example, CareerBuilder's latest studies on the effects of the skills gap on the U.S. labor market revealed that 68 percent of 2,380 employers surveyed currently have open positions for which they cannot find qualified candidates [2]. What makes the situation more worrisome is that the gap between the number of jobs posted each month and the number of people hired is growing larger as employer's struggle to find candidates to fill positions at all levels within their organizations. According to the CareerBuilder report, two in three employers (67 percent) are concerned about the growing skills gap and 55 percent of the employers have seen a negative impact on their business due to extended job vacancies.

A research program undertaken by LinkedIn and Capgemini examined the digital talent gap that affects all areas of the business [6]. 50 percent of the organizations participating in this study acknowledged hampering impacts of the widening digital talent gap on their digital transformation programs and agreed that a shortage of digital talent has cost them a competitive advantage. Another example is the latest research conducted by Udemy, which disclosed that the nearly 80 percent of Americans feel the U.S. is facing a skills gap, and 35 percent state that it affects them personally in the face of constant changes [14].

However, no consensus has been reached as for what underlies this mismatch or what causes a widening shortage of skilled workers in the marketplace [4]. Different stakeholders have experimented various approaches to close the skills gap. For instance, some argue that companies should develop their own talent pipeline program as the remedy [12]. The government has recognized the need to connect individuals with careers from an early age and is trying new approaches such as the Department of Labor's Registered Apprenticeship Program. On the other hand, some propose to develop labor-market intermediaries such as employment agencies or trade associations, employer relationships with technical colleges or other institutions, and employer-provided training to bridge the supply and demand side [15].

How to prepare workers for the future has also largely focused on what educators can and should offer through the academic programs. Colleges and universities have been striving to close the gap using different approaches and

strategies such as the co-op, partnership program run jointly by the employers and university and the connected program or curriculum programs using connections between the various stakeholders throughout a student's educational journey [5, 17]. However, such efforts mainly address the soft employability skills gap instead of hard employability skills. As university faculty, we are facing challenges to ensure students have both technical skills and soft skills that employers will eventually want and to prepare students to be career-ready and competitive in a global economy. Therefore, this study aims to tackle the challenge from a different angle by examining the body of knowledge as represented in our IT/IS program course syllabi and the recent job postings using natural language processing (NLP) techniques.

The rest of the article is organized as follow. The next section revisits an IT/IS educator's dilemma in curriculum development. The third section focuses on research methodology and presents the detailed steps involved in data collection, data cleaning, and data analysis using several information retrieval and data mining tools and techniques. The discussion section explores how the result informs clarifications to the wording of existing course syllabi, modifications to existing course contents, or the introduction of new courses into the curriculum. The last section presents our conclusions along with several future research directions.

2 Motivation: Resolving an Educator's Dilemma

The skills gap is particularly conspicuous in the technology field due to the many fast-paced, disruptive innovations. From voice assistants to self-driving cars to robot caregivers, automation and artificial intelligence (AI) are becoming more and more pervasive. Such phenomenon engenders a stream of controversial debates and conversations regarding the impact of those emerging technologies on the labor market [3, 7]. It is estimated that graduates from the disciplines such as computer science, IT, and IS today will find those skills they have learned from school out of date within six years [8]. There is a pressing need for university educators to examine the current curriculum to identify and adjust those that do not match industry demand to adequately prepare students for the work of the future [1].

However, given the fast rate of change in the employment landscape, the IT/IS curriculum can easily become bloated. An educator's dilemma is to find a balance between accommodating new material from the discipline, teaching fundamental concepts and maintaining a curriculum with the fixed number of credits in accordance with accreditation requirements. A holistic model was proposed to tackle such dilemma [16]. The authors [16] came up with a conceptual framework consisting of two models: the strategic model for "when" to

incorporate new technology topics into the curriculum and the tactical model for "how" to insert new courses into the existing curriculum. The major limitation of this framework is that "when" and "how" decisions are subjective and to some extent "ad hoc" instead of "data-driven". Further, no formal assessment of the effectiveness of this framework has been conducted. Thus, this study attempts to tackle the dilemma using a more flexible and agile approach. One of our goals is to identify the major overlaps and gaps between the two entities (i.e., course syllabi and job advertisement) systematically by employing quantitative methods.

3 Methodology

This section covers details of data curation, data cleaning and analysis techniques that were used to conduct an initial explorative study.

3.1 Data Curation

The process for conducting our study requires two data sets: a set of job postings for an industry and the current syllabi used by the program or department for the relevant majors. To get the required job skills for an industry, we created a web scrapper for one of the leading online job seeking and recruiting websites, indeed.com. This scrapper retrieved all textual information on a job posting while removing all html and JavaScript code. Doing this we lost the structure of the post but retain all text in the posting. To scrape selected jobs, we restricted the search to a location, entry level jobs only, and a search term which is required to be in the post. For instance, our scrapper can pull the first 100 pages of a search with 15 jobs per page (1500 jobs) for selected parameters such as Entry Level Jobs in Washington DC for "Information Technology". The results for each one of these data scrapes was then saved for further analysis.

To collect what skills the university currently covers in its curricula, we collected the last syllabus used for each course, 99 in total. Using data extraction techniques with Python scripting, we extracted all the textual information for the syllabi. The extracted text from each syllabus was saved in its own file so each individual class could be compared to industry needs.

3.2 Data Cleaning

An average job posting from indeed.com contains many boilerplate sentences, which is irrelevant for the goal of our analysis to see if the university's courses contain all words that the job posting contains. For instance, basic language and frequent words (the, is to, a, an) are not particularly useful for our analysis.

These words are often called "stop words" and these stop lists come with NLP kits. We remove all stop words from the job postings using the Natural Language Toolkit's (NLTK) stop word list for English [9]. With the basic stop words removed, most job postings still contain some common job-related terms. Most of these deal with employment and Human Resources such as Equal opportunity, LLC, headquartered, experienced, etc. To remove these terms from our evaluation, we hand curated a list of more than two hundred common employment terms. After removing both the stop words and the common employment terms, we obtain a list of terms that mostly relate to job skills needed to fill the position.

Many of the data issues associated with the indeed.com data are present in our syllabi data as well. For example, syllabi also come with boilerplate information as the template is given by the university. We conducted the same data cleaning procedure to remove all stop words from the syllabi. The ultimate goal for analysis is to get rid of noise information about school policies (late assignments, quizzes, test, drops, and withdraws) and extract the skills taught by each course. We crafted a list of 200+ academic terms as stop words and removed those from each syllabus as well. In the end we were left with only the key concepts in each syllabus.

3.3 Analysis Techniques

To compare the skills taught in a class with industry needs, we created a process that would rank the relevancy of each course to a specific industry. First, we extracted the key terms that represent an industry and represent each syllabus respectively. We conducted this extraction automatically using the common information retrieval technique of term frequency–inverse document frequency (tf-idf). Tf-idf ranks the importance of a term to the target.

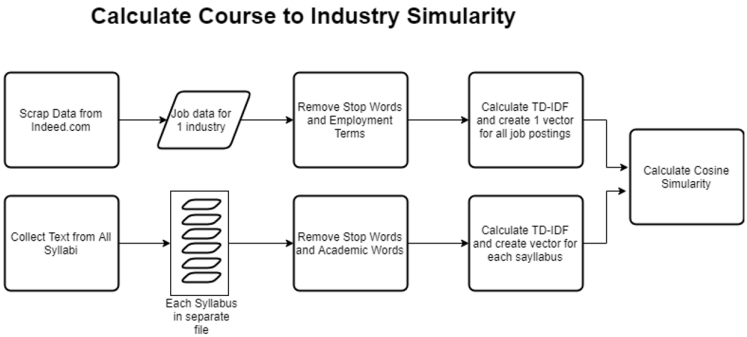


Figure 1: Flowchart of the Research Methodology

Having a value for each term, we turned each document into a vector of values representing the terms in the document. To find out if two documents were similar we took the cosine similarity measure between the two vectors. If the cosine value is equal to 1, they are the same document, if the cosine value is 0, the documents share no terms in common [10]. We conducted cosine similarity analysis on both vectors and generated one file for each industry. In each file the courses are ranked by how closely they match the industry job postings based on their cosine similarity value. For instance, if we want to inquire about the top five courses offered in our program that cover the skills matching the Data Science industry job needs most closely, we would see: IT390, IT385, MSC385, MSC325, IT820. These courses accurately reflect the data science components of our undergraduate data science specialty in the IT major as well as a course in the doctorate in cybersecurity program. Figure 1 illustrates the steps involved in our research methodology.

In addition, we had the goal of seeing what we should be teaching that industry needs and what we are currently teaching that is not relevant to industry. These features are created as a byproduct of the above tf-idf analysis. Industry needs not currently covered in a class can be discovered by taking the set difference between terms in the keywords in industry and the keywords in the syllabi, represented by A in Figure 2. Topics covered in courses but not mentioned in job postings are covered by the reverse, the set difference between terms in the syllabi and terms in industry, represented by B in Figure 2. The

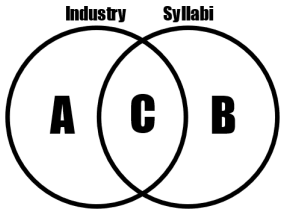


Figure 2: Venn Diagram showing the relationships between industry needs and course topics

code and analysis for this research has been scripted using Python. To give a high-level view of how we are doing as a school each semester, we calculate a coverage ratio. This is defined as the percentage of terms in industry that we cover in the syllabi. We calculate this in order of rank, so we can see how we do on the top 10 industry terms, as well as top 25, 50, 100, and 1,000. If all terms are examined, regardless of rank, this would be the ratio of C/A in Figure 2.

4 Result Analysis and Discussion

In this section several different analyses on the preliminary results are presented. The purpose is to examine gaps and overlaps between our curriculum and industry job needs from various angles in order to achieve a holistic picture.

4.1 Analysis 1: Concepts in Syllabi Not in Job Advertisements

Figure 3 is a list of the top 50 terms that were identified as frequently occurring in our syllabi but not appearing in the job description.

programmed	portfolios	rfid	globalization, globalized
literacy	recursion, recursive	tracer, ipconfig	outsourcing
webpage	traversal	deterministic	polynomial
macintosh	watermarking	cybercriminals, cyberterrorism, cybercrime cyberstalking	backdoor
ciphers	authenticated	elliptic permutation	asymmetric
randomness, pseudorandom	cryptanalysis, cryptosystem	trapdoor	decryption
sha md5	symmetric	highperformance	traversed
asynchronously, synchronizing	hotspot	cyberphysical	usenix
enigma	multithread, threads	deadlock	pentester
cyberinsurance	motherboards motherboard microcomputer	webapps storyboard	tuples
conditionals	concurrency	cyberattacks	honeynets honey pots
ransomware	unmasking	infections	backdoors
singleton	antispyware	corpora cordova	tokenization

Figure 3: Top 50 Terms in Syllabi Not in Job Advertisements

Most of these terms cover fundamental concepts in the core areas of computer technology (e.g., motherboards), software development (e.g., threads), networking (e.g., synchronizing), databases (e.g., tuples), and cybersecurity (e.g., ciphers). These reflect the depth of coverage in our syllabi. RFID is the only technology that is mentioned in the syllabi and not in the job advertisements selected. Some terms need to be edited in the syllabi, such as "pentester" to "penetration tester", webapps to "web application" and "honey pots" to "honeypots".

4.2 Analysis 2: Skills Required by the Industry, but Not Covered by Each Concentration

While tf-idf and cosine similarity gives us a ranking of our courses per industry, it still does not let us know if our curriculum is missing any key concepts. To analyze this, we also generate a list of key terms that occur in the indeed.com postings that do not occur anywhere in our syllabi. The top 5 terms in each specialization are shown in Figure 4.

Cybersecurity	Data Science	Computer Science	Information Systems	Healthcare IT
genetics	geospatial	physics	geospatial	chiropractic
implements	analysts	developer	gis	rehab
diagnostics	scientist	geospatial	feature	sport
analysts	osint	custom	sap	spine
architecting	geographic	analysts	analysts	patients

Figure 4: Top 5 Terms Not Covered by Each Specialty

These are the unfiltered top 5 terms. Geospatial, GIS and geographic terms occur multiple times in three specialties and would appear to be a possible addition to our program. Our coverage in cybersecurity appears more than adequate and the highly occurring terms are more descriptors than skills. However, "genetics" as a term in cybersecurity will be researched further as this might be an emerging area. The use of "developer" rather than "programmer" will result in a review of our software development syllabi. Similarly, the use of "analysts" will be reviewed. Open source intelligence ("osint") is covered in the cybersecurity syllabi but we will further investigate its use in data science, outside of cybersecurity.

The Healthcare IT words are not technology related but show how Healthcare IT has migrated outside the medical offices and hospitals. It is now being implemented in chiropractors, rehab clinics, spine clinics and sports facilities.

4.3 Analysis 3: Coverage Ratios

The coverage ratio represents the percentage of top keywords that exist in a course. Figure 5 shows ratios for the top 10 to 1000 keywords in each of the five specialties.

Concentration	10	25	50	100	1000
HEALTHCARE IT	0.6	0.8	0.86	0.88	0.726
INFORMATION SYSTEMS	1	0.92	0.88	0.8	0.711
DATA SCIENCE	0.9	0.92	0.92	0.94	0.726
CYBERSECURITY	1	0.96	0.92	0.94	0.811
COMPUTER SCIENCE	1	1	1	0.99	0.865

Figure 5: Coverage Ratios by Specialty

For the top 10 words in Information Systems, Data Science, and Cybersecurity specialty there are at least one course that mentions each of them (100%). For Data Science we have 90% (9 of the 10 keywords) and for healthcare 60%. Further analyses show that this number changes as the number of keywords

increases. Computer Science is fully covered for the first 100 keywords. Cybersecurity coverage is also good through 100 keywords. For Healthcare the value goes up as we go down in keywords, so the syllabi are covering many of the mid-term words. Further analyses will include coverage for when a keyword happens in at least 2 courses, 3 courses, and so on.

4.4 Analysis 4: Programming Languages Requested in Job Advertisements

Learning to program is part of the IS/IT curriculum, and academics must select one or more languages to teach basic and more advanced concepts of software development. Figure 6 shows the five most common programming languages for each specialty.

Healthcare IT	Information Systems	Data Science	Cybersecurity	Computer Science
java	java	python	python	java
cobol	javascript	java	java	python
javascript	spark	spark	shell	javascript
	puppet	bash	ruby	perl
	python	javascript	perl	ruby

Figure 6: Most Common Programming Languages in Job Advertisements

Java and Python are the primary languages requested across all specialties. This reinforces our curriculum decisions where Java I and II are recommended for the Computer Science specialty and Python (introduced in 2016) for the Data Science and Cybersecurity specialties. Web development, including the JavaScript and Perl languages, is also offered and recommended for the Information Systems specialty. Apache Spark is an important addition for the Data Science and Information Systems specialties and should be added to the programming environment for these options. Cobol was unexpected and indicates the reliance on older software in the healthcare field.

5 Conclusions

This was an initial analysis of the two datasets: curriculum scope and job advertisements in the IT/IS field. While there was substantial overlap, the initial analysis showed that there were some gaps. In several cases this can be overcome by a careful review of the syllabi to ensure that they reflect the current terminology in the field and that variations on terms are used (e.g., analysts not just analysis). In other cases, a new course might be needed. In this initial review, geospatial/GIS seems a good addition to the curriculum and will be researched further.

There are many additional analyses possible now that we have the datasets in place. We can run these analyses every semester to identify changes in the job advertisements as well as any curriculum changes that we have made. We will also do additional analyses looking for other gaps over time. For example, break down the syllabi by level (undergraduate, masters, and doctorate) and tie this into job advertisements and their education requirements. Further research will also be performed with a third data set, applicant resumes, to determine whether it is possible to use these data to promote certain job positions for individuals in the program as part of our job readiness activities.

References

[1] *What should we be teaching the next generation of computer scientists?* Times Higher Education, Jan. 2016.

[2] CareerBuilder. The skills gap is costing companies nearly $1 million annually, according to new CareerBuilder survey. `http://press.careerbuilder.com/2017-04-13-The-Skills-Gap-is-Costing-Companies-Nearly-1-Million-Annually-According-to-New-CareerBuilder-Survey`.

[3] Frey, C. B., and Osborne, M. The future of employment: how susceptible are jobs to computersation?, 2013.

[4] Fuller, J. B. Whose responsibility is it to erase america's shortage of skilled workers?, 2015. `https://www.theatlantic.com/business/archive/2015/09/whos-responsible-for-erasing-americas-shortage-of-skilled-workers/406474/f`.

[5] Fung, D., and Mathews, T. *Connected Curriculum for Higher Education.* UCL Press, 2017.

[6] Institute, C. D. T. The digital talent gap—are companies doing enough?, 2017. `https://www.capgemini.com/wp-content/uploads/2017/10/dti_the-digital-talent-gap_20171109.pdf`.

[7] Institute, M. G. Jobs lost, jobs gained: Workforce transitions in a time of automation, 2017.

[8] Koc., E. Is there really a skills gap?, Feb. 2018.

[9] Loper, E., and Bird, S. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies*

for Teaching Natural Language Processing and Computational Linguistics - Volume 1 (Stroudsburg, PA, USA, 2002), ETMTNLP '02, Association for Computational Linguistics, pp. 63–70.

[10] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[11] Mills, K. G., F. J. B., and Rivkin, J. W. Growth and shared prosperity: Havard business school- u.s. competitiveness project, 2015.

[12] of Commerce Foundation: Center for Education, U. C., and Workforce. Managing the talent pipeline: A new approach to closing the skills gap, 2014.

[13] Piliouras, T., Yu, R., Villanueva, K., Chen, Y., Robillard, H., Berson, M., Lauer, J., Sampel, G., Lapinski, D., and Attre, M. A deeper understanding of technology is needed for workforce readiness. playing games, texting, and tweets aren't enough to make students tech-savvy. In *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education* (April 2014), pp. 1–8.

[14] Udemy. Skills gap report, 2017.

[15] Weaver, A. The myth of the skills gap, 2017.

[16] X., L., and D., M. Tackling an is educator's dilemma: A holistic model for "when" and "how" to incorporate new technology courses into the is/it curriculum. SAIS.

[17] X., L., and D., M. Constructing a connected program in it/is: A non-r1 university case. SAIS.

Tarot: A Course Advising System for the Future*

Joshua Eckroth and Ryan Anderson
Math & Computer Science Department
Stetson University
DeLand, Florida 32723
{jeckroth,randerson1}@stetson.edu

Abstract

Course advising plays an important role in a student's college experience. Uninformed decisions about which courses to take during which semesters can cause a student to fail to graduate on-time or struggle with a major not ideal for the student. Likewise, faculty and administration routinely ask questions about when courses should be offered, how many teachers are needed to cover the courses, and how changes to prerequisites or major requirements impact schedules. However, course advising is challenging because it can demand complex planning years into the future while considering numerous rules about major requirements, course prerequisites, maximum course loads, and course offering schedules. We have developed Tarot, a course advising system that uses a planning engine to develop multi-year course schedules for complex scenarios such as double majors, study-abroad semesters, course overrides, early graduation, and transfer credits. This paper describes Tarot's design and operation and distinguishes its capabilities from existing course advising tools.

1 Introduction

Course advising plays an important role in ensuring a student's success in their college career. This is particularly true for students attending private,

often costly colleges and students on academic or athletic scholarships. In these cases, time-to-graduation is a hard constraint on their ability to afford their education. Likewise, colleges and universities are ranked according to 4-year graduation rates, further emphasizing the importance of ensuring students finish their degrees on-time. Without a strong course advising infrastructure, including software tools and well-trained advisors, student success and the institution's ranking may suffer.

We have developed a new software tool called TAROT to assist advisors and students. TAROT is designed to help with the complex constraints and rules inherent in planning multi-year course schedules. Although many academic departments design prototypical two- or four-year schedules to help guide entering students, not all students follow the same path or come from the same background. Once a student deviates from this pre-defined plan, e.g., they bring transfer credit, or add a second major, or study abroad, or need a course override, or must finish in 3.5 years, etc., then the pre-defined plan is useless. Now, the student and advisor must think through the intricacies of major requirements, course prerequisites, and offering times to find a plan for this student's particular circumstances. This cognitive burden introduces the possibility of mistakes and precludes any more advanced forecasting such as finding the best time to study abroad or finding every possible major elective that would satisfy graduation requirements.

Yet handling constraints and managing complex interactions is the raison d'être of planning engines from the field of artificial intelligence. TAROT is a planning engine designed specifically for course advising. Its use cases focus on developing a student's schedule across multiple semesters rather than scheduling courses for times/days in the week, rooms, etc. We implemented TAROT in Prolog and exploit this language's ability to perform backtracking search to find solutions that satisfy arbitrary constraints. TAROT is also designed for efficiency, and time-to-answer is reported for each use case described below. At this time, TAROT handles just a few majors (math, computer science, computer information systems, and cybersecurity), and while we have plans to extend to more majors, we do not plan to include every available university course. General education requirements, for example, are marked as "gen. ed." slots in a schedule since the courses are mostly interchangeable and available in any semester.

In order to best explain how TAROT differs from other course advising systems, we define four levels of sophistication in terms of the kinds of questions students and/or advisors may wish to answer.

Level 0: Questions about the present. At the most basic level, students and advisors wish to know: What courses has this student taken? What is the student's current GPA, and how many credits have they earned?

Level 1: Questions about a possible future. This level introduces forecasting, i.e., establishing a schedule for the student's future. Relevant queries include: What courses should the student take, and in which semesters, in order to graduate on time? What is a sequence of courses that allows the student to achieve a double major? How does the schedule change if the student wishes not to take more than two major courses per semester? How does the schedule change if the student studies abroad during a certain semester or takes a semester off? (Note, we assume students are unable to transfer their study abroad credits, though of course this is not always the case in practice.)

Level 2: Questions about all possible futures. This next level considers more than one possible future, or often, all possible futures. For example: When should this student study abroad to ensure on-time graduation? Which semester should a particular course be taken to remain maximally flexible? What grade must the student earn in a certain class to ensure a desired GPA?

Level 3: Questions about the rules. Finally, staff and faculty may also ask about the rules themselves. For example: How different are two majors in terms of shared and distinct courses? How many teachers are required to cover the courses that students may take to complete their degrees?

In the remainder of this paper, we will show how TAROT is able to answer questions from all four levels. We begin by discussing related works and how they fit in terms of these levels. We then describe TAROT's internal implementation and showcase how TAROT handles a variety of questions.

2 Related Work

Several systems have been designed to help with academic advising. For simple advising, i.e., our Level 0, the use cases involve the present and the near future. For example, a student comes to an advisor after one or two semesters and wants to know his or her eligibility for some specific classes in the upcoming semester. Engin et al. [4] created a system that attempts to answer those questions by comparing courses to a set of rules created using *Oracle Policy Automation* (OPA). The student's level (freshman, sophomore, etc.) can be determined using credits earned, or the student could check his or her eligibility for a course in terms of prerequisite courses. Another system from Vincenti and Bennett [7] is designed for the student to use on his or her own ("self-advising"). Once the student selects courses for the current semester, a list is generated showing which courses will be available and relevant in the following semester. Guerin and Goldsmith [2] created an academic advising system that uses a dynamic Bayes net. This system suggests courses based on the academic background of the student. For example, students are directed to different sequences of Computer Science courses depending on whether they were stronger

in mathematics but weaker in programming, or vice versa. Dodson et al. [3] use Markov decision processes to likewise recommend next-semester courses based on the student's history. Systems like these are effective when the student and/or advisor have a clear understanding of the student's desires in the near term, but are not capable of developing customized concrete plans multiple semesters into the future.

The systems described above operate mostly on Level 0 since they do not generate full schedules. Now we consider systems that operate at Level 1. The IDiSC+ system by Mohamed [6] uses a sophisticated algorithm to consider several factors while forecasting semesters. This system, like those in Level 0, considers prerequisites and availability, but the IDiSC+ system can also work with constraints such as ensuring minimum credit hours per semester, number of courses per semester, required electives, minimum GPA, course priority, and even the student's budget. A system with this type of complexity is effective for planning the archetypal undergraduate education, but falters when even more complex factors – double majors, when to study abroad, or if a semester will be missed – are introduced.

To our knowledge, no systems are available that operate at Level 2 or 3, i.e., finding all possible schedules and making inferences on this set, and asking questions about the major requirements and multitude of course schedules themselves.

These prior efforts demonstrate a reasonable ecosystem for assisting students and advisors in general advising scenarios. Students have access to databases and planning engines that allow them to lay out an appropriate schedule, perhaps for their entire undergraduate career. Unfortunately, more complexity exists in real-world academic advising. This added complexity is where Tarot is most unique. In particular, Tarot can identify all possible scenarios meeting user-provided constraints, rather than just the most likely. Consider an example where a first-year student is certain that she will study abroad for one semester. She asks her advisor which semester taken abroad would delay her graduation the least, assuming she is unable to transfer any credits (the safest assumption for the hypothetical scenario). An advisor would normally have to spend a substantial amount of time working this issue out, taking into consideration long term questions like course frequency/availability, GPA, and double majors; and this would have to be repeated for each potential semester abroad. For Tarot, this kind of search can easily be constructed by specifying just a few constraints. Tarot is designed to work with arbitrary constraints, some of which may introduce exhaustive search to find all possible solutions to a question.

In the following sections, we comment on Tarot's implementation and then evaluate its capabilities and efficiency across all Levels 0 through 3.

3 Implementation

TAROT is implemented in Prolog, whose backtracking search paradigm is an ideal fit for TAROT's use cases. Course information, including major requirements, prerequisites, and course offering times (Fall/Spring; odd/even years) are stated as Prolog facts or, in some cases, more complex rules that check for specific conditions. For example, our computer science major's required senior research course has complex prerequisites: students must have completed three 300-level courses, two of which must be computer science (CSCI) courses, one of which may be a CSCI or a computer information systems (CINF) course.

For most advising questions, TAROT internally fills out a multi-year schedule. In some cases, TAROT also proceeds to find all possible ways of building the schedule in order to answer questions from Levels 2 and 3. The schedules are built by establishing a blank schedule full of "free slots" and then recursively replacing free slots with courses. Prolog's backtracking allows us to specify arbitrary constraints on the schedules, such as maximum number of major courses to take each semester, without having to define code that specifically handles these constraints. Any schedules that fail to meet the constraints will be thrown out and the search for satisfying schedules will continue. As a case in point, finding a schedule for a student who wishes to double major is as simple as finding a schedule for the first major, then starting with this schedule (and its remaining free slots) and filling in the requirements for the second major using the same code path that filled in the first major's courses.

Due to space constraints, we refrain from showing much Prolog code in this report. However, it is worth noting that considerable effort has been focused on efficiency while developing TAROT. For example, the system avoids excessive backtracking by first scheduling senior-level courses in the last or second-to-last semester. Next, junior-level courses are scheduled, and so on. This ensures introductory courses that serve as prerequisites for later courses, but which may be taken any time since they have the fewest prerequisites, are scheduled under the most constrained environment, when most courses have already been scheduled. This design follows the general design paradigm of constraining the search as early as possible to avoid generating results that are ultimately thrown away by subsequent constraints.

Presently, TAROT suffers from a lack of a proper user interface. Advising questions must be specified as Prolog queries on a command line. In future work, we plan to investigate an appropriate web-based interface that best exposes TAROT's capabilities in a form that is clear and efficient for non-programmers.

4 Evaluation

We evaluate Tarot by demonstrating some example questions and Tarot's responses across the three levels.

Level 0: Questions about the present. Our university uses Ellucian Degree Works [1] to show information about a student's class history and required courses. Tarot has a built-in parser for Degree Works' class history format. We can read this class history and then ask Tarot to calculate the student's GPA:

```
readStudentFile('classHistoryH1.txt', Taken),
calcGPA(Taken, GPA, Credits, AllCredits).
```

The result is a table of the student's class history, including grade earned and individual course credits, followed by their GPA and overall earned credits. We note that the classes and grades shown below are simulated. The query completes in 0.002 seconds on a laptop with an Intel i5 processor and 8 GB RAM.

Transfer	astr180 (Tr, 3)	chem110 (Tr, 4)	csci111 (Tr, 4)	
2016 Fall	csci142 (B+, 4)	csci211 (A-, 4)	rels390 (P, 4)	
2017 Spring	csci201 (B, 4)	csci221 (A-, 4)	hlsc219 (A, 4)	math141 (C+, 4)

GPA: 3.33, Credits: 24, All credits (including transfer): 39.

Level 1: Questions about a possible future. Using this same simulated class history, we can next ask for a schedule that would allow the student to complete his computer science degree in two years. The following query reads the class history, generates four semesters covering the next two years, fills in a schedule, and further ensures there are at least five general education courses in the schedule.

```
readStudentFile('classHistoryH1.txt', Taken),
generateBlankSemesters(2018, 4, Semesters),
finishDegrees([csci], Taken, [], Semesters, PlanSemesters),
ensureGenEdCount(PlanSemesters, 5).
```

The result is a table that gives the proposed schedule. Note that major requirements, course prerequisites, and course offering times are considered while developing the schedule. The query completes in 0.006 seconds.

2018 Fall	cinf401	csci311	csci321	math142
2019 Spring	csci331	csci431	gen. ed.	gen. ed.
2019 Fall	csci498	phys141	gen. ed.	gen. ed.
2020 Spring	csci301	csci499	phys142	gen. ed.

The student may also wish to limit the number of CSCI and CINF and MATH courses to take in the same semester to, say, two courses. Tarot fails

to find a successful schedule (in 1.742 seconds), indicating the constraint is not possible to achieve. Adding another semester does succeed (in 0.437 seconds):

```
readStudentFile('classHistoryH1.txt', Taken),
generateBlankSemesters(2018, 5, Semesters),
finishDegrees([csci], Taken, Semesters, PlanSemesters),
ensureCourseLoad([csci,cinf,math], PlanSemesters, 2),
ensureGenEdCount(PlanSemesters, 5).
```

2018 Fall	csci321	math142	phys141	gen. ed.
2019 Spring	csci331	csci431	phys142	gen. ed.
2019 Fall	cinf401	gen. ed.	gen. ed.	gen. ed.
2020 Spring	csci301	csci498	gen. ed.	gen. ed.
2020 Fall	csci311	csci499	gen. ed.	gen. ed.

Now consider a new student, with no class history, who wishes to study abroad (which we assume involves no transferrable major courses) during 2020 Spring semester. TAROT can plan around this missing semester to find a successful four-year schedule (in 0.006 seconds):

```
generateBlankSemesters(2018, 8, Semesters),
setMissingSemester(Semesters, 2020, spring, Semesters2),
finishDegrees([csci], [], Semesters2, PlanSemesters).
```

Double-majors are also trivial to specify, for example, a MATH+CSCI double major (completed in 0.009 seconds):

```
generateBlankSemesters(2018, 8, Semesters),
finishDegrees([math,csci], [], Semesters, PlanSemesters).
```

Level 2: Questions about all possible futures. If a student wishes to know *when* to study abroad, we can write a query that forces TAROT to find all schedules for all possible study abroad semesters, and then tally the number of schedules that work for each different study abroad selection. We use Prolog's 'findall' feature for this purpose. Naturally, as we develop a more friendly user interface, direct use of Prolog code will be entirely avoided.

Assuming the student is a freshman starting college in Fall 2018, the system finds the following number of working schedules for each possible study abroad semester:

2019 Spring – 47,340 (20%)	2019 Fall – 51,549 (21%)	2020 Spring – 35,469 (15%)
2021 Spring – 32,870 (14%)	2021 Fall – 45,092 (19%)	2022 Spring – 28,026 (12%)

Thus, the student has the most flexibility in her schedule if she studies abroad in 2019 Fall. Note that 2020 Fall is missing due to the frequency in which some required major courses are offered (Fall/Spring; odd/even years).

A student may wish to know all the possible semesters to take a particular course, for example, our department's artificial intelligence course. To answer this question, we first generate all possible schedules with 'findall' starting in Fall 2018, then tally all the different semesters the course appears. The query completes in 112 seconds and reports that the only semester available to take the course is 2021 Spring if the student starts as a freshman in our introductory programming course. If the student has prior programming experience and starts in CSCI 141, the AI course may be taken in 2020 Spring in 20% of possible schedules or 2021 Spring in the other 80% of possible schedules.

Sometimes students wish to know what grades are required in certain courses to achieve a certain GPA. This is straightforward to request from Tarot. We can simply request Tarot to generate a schedule (with unknown grades in classes not yet completed), calculate their GPA with the query 'calcGPA(Taken, GPA, Credits, AllCredits)', and then add a constraint like 'GPA >= 3.5'. We use the CLPR library (constraint logic programming in real numbers) [5] to efficiently establish constraints on real-valued variables, in this case, course grades. CLPR uses a solver for linear equations rather than backtracking search.

Level 3: Questions about the rules. Finally, Tarot can report how similar two majors are by finding schedules for each major separately and then tallying how many courses are in common. This operation takes as much time as generating all possible schedules, twice (about four minutes).

Tarot can also help staff and faculty determine how many teachers are needed to cover all possible schedules for freshmen entering in odd or even years (assuming one section of each course). This information can serve as evidence in a proposal to support a new tenure line – i.e., an argument can be made that at least some number of teachers are required to cover a new major within an existing department. The query involves finding all distinct possible schedules for each relevant major, for freshman starting in an odd year and even year separately. Then, Tarot calculates the number of distinct courses in each semester and divides that number by the course load for teachers (e.g., 3/3). Naturally, this is Tarot's most time-consuming query, requiring 9 minutes, but it is also likely to be the least frequently executed query.

5 Conclusion

This paper described Tarot, a tool that aids students and advisors in everyday, yet sophisticated course advising scenarios. By making use of an integrated planning engine, Tarot is capable of developing long-term course schedules while respecting major requirements and prerequisites. Tarot can also handle arbitrary constraints such as maximum course load, study abroad semesters,

and double majors. By producing and analyzing all possible schedules for a given set of constraints, TAROT can also provide information about the best time to take a certain course or the minimum number of teachers required to cover all courses. In future work, we plan to develop an appropriate user interface for students and advisors that best exposes TAROT's advanced features.

References

[1] Ellucian Degree Works. https://www.ellucian.com/Software/Ellucian-Degree-Works/.

[2] DEKHTYAR, A., GOLDSMITH, J., LI, H., AND YOUNG, B. The bayesian advisor project i: Modeling academic advising. Tech. Rep. 323, University of Kentucky Dept of Computer Science, 2001.

[3] DODSON, T., MATTEI, N., GUERIN, J. T., AND GOLDSMITH, J. An english-language argumentation interface for explanation generation with Markov decision processes in the domain of academic advising. *ACM Transactions on Interactive Intelligent Systems (TiiS) 3*, 3 (2013), 18.

[4] ENGIN, G., AKSOYER, B., AVDAGIC, M., BOZANLI, D., HANAY, U., MADEN, D., AND ERTEK, G. Rule-based expert systems for supporting university students. *Procedia Computer Science 31* (2014), 22–31.

[5] HOLZBAUR, C. Ofai clp(q, r) manual. Tech. Rep. TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna, 1995.

[6] MOHAMED, A. A decision support model for long-term course planning. *Decision Support Systems 74* (2015), 33–45.

[7] VINCENTI, G., AND BENNETT, V. A JSON-based self-advising system. *Journal of Computing Sciences in Colleges 32*, 3 (2017), 39–45.

Using Bayesian Spelling Correction to Improve Students' Skills in a Data Structures Class*

Reva Freedman
Department of Computer Science
Northern Illinois University
DeKalb, IL 60115
rfreedman@niu.edu

Abstract

This paper describes the adaptation of a natural language processing algorithm, Bayesian spelling correction, for use in teaching data structures. First I describe the algorithm and its mathematical basis. Then I describe the conversion to a programming assignment. Finally I describe what students learned from doing the assignment, lessons learned from this project and alternatives that could be used in assigning this project again.

1 Introduction

The purpose of this project was to investigate what it would take to use an algorithm from natural language processing (NLP) as the basis of a real-world size programming assignment in a data structures class. There were multiple motivations for undertaking this research. First, it would enable students to learn an interesting piece of NLP that does not require any background in linguistics. Second, it uses a large enough data volume that efficiency actually matters and students can't just eyeball the data to see whether their answers are correct. Most importantly, the assignment benefits from high-level data and control structures, including maps or associative arrays (dictionaries in

Python) and streams or lambda expressions (list comprehensions in Python). Students also get to practice some pieces of infrastructure that will be useful in future real-world endeavors, including the use of a serialized singleton class to store data between programs, and how to instrument a program for analyzing speed.

The algorithm chosen for this purpose is the Bayesian spelling correction algorithm. A variant of this algorithm is used in most word processors today, yet most people who have not studied it generally have no idea how it works. Students generally assume that spelling correctors look up the problem word in a dictionary. They can readily see that looking up the word in a dictionary will tell you only whether the word is correct. For words not found, however, students have generally not noticed that the possible corrections provided by the word processor are sequenced from most to least likely.

In the first section of this paper I describe the algorithm and its mathematical basis. Then I describe its conversion to a programming assignment. Finally I summarize what students learned from this project and alternatives that could be used in assigning this project again.

2 Background

The algorithm described here was first suggested by kernighan. Although it was first published in 1990, it is still the basis of the spell checker in most commercial word processors. The algorithm is based on Bayes' Law, which was originally proposed in [1], and found today in most statistics or AI textbooks, e.g., [140]jurafsky-martin[404-405]russell-norvig. The classical Bayesian problem can be described as follows: Given one feature of an object, how can we determine what is the most likely class it falls into? For example, suppose there are three possible causes for a rash — a cold, chicken pox or the Ebola virus. If a person has a rash, which of these diseases are they most likely to have? Bayes' Law can be used to answer this question when all we know is the inverse data, i.e., for each of those diseases, we know what percent of sufferers usually have a rash.

I initially present this formula to students using a non-NLP example for two reasons. First, many of them have learned it in a statistics class, and I want to remind them of it. Second, I want them to see that this algorithm is not NLP-specific, but can be useful in many contexts.

Bayes' Law tells us that $P(x|y) = P(y|x)P(x)/P(y)$, where $P(y|x)$ represents the probability of y given x. For example, suppose that 10% of people with a cold have a rash, 90% of people with chicken pox have a rash, and 50% of the people with Ebola have a rash. (Note: these numbers are not medically accurate. I was not able to obtain accurate medical information, and in any

case, I like to warn students about "medical students' disease", where the latter are always subject to worrying about whatever issue is being discussed in class.)

Now suppose that for every 100 people who visit a doctor with a rash, 90 have a cold, 9 have chicken pox and only one has Ebola (which would of course be high, but integers keep the math easier). Suppose Kim has a rash. In that case we know that $P(rash) = 1$, so we can calculate the probability that Kim has each disease under consideration as follows:

$$\begin{aligned} P(cold|rash) = P(rash|cold)P(cold) &= .10 * .90 = 0.0900 &= 9.00\% \\ P(pox|rash) = P(rash|pox)P(pox) &= .90 * .09 = 0.0081 &= 0.81\% \\ P(Ebola|rash) = P(rash|Ebola)P(Ebola) &= .50 * .01 = 0.0050 &= 0.50\% \end{aligned}$$

In other words, although a rash is less common as a symptom of a cold than of the other diseases, Kim is most likely to have a cold simply because colds are so much more common. This insight is the key to understanding the spelling algorithm.

This observation can be applied to misspelled words by framing spelling correction as follows: Given a string of characters s, which word w in the dictionary has the highest $P(w|s)$, i.e., the probability that the word the user intended to type is w given that the user actually typed the string s. Since Bayes' Law converts $P(w|s)$ to $P(s|w)P(w)/P(s)$, and $P(s)$ is constant for a given string, we only need to calculate $P(s|w)P(w)$ for each word in the dictionary. In Bayesian theory, $P(w)$ is called the prior probability of w and $P(s|w)$ is called the likelihood of w.

3 Implementation

The implementation is based on using a large corpus to obtain frequency information about both letters and words. The larger the corpus, the more accurate the results. Kernighan used the 1988 Associated Press (AP) corpus, which contains about 44,000,000 words. We used a version of this corpus [4] originally prepared for the annual TREC information retrieval competition [5] sponsored by the National Institute of Standards and Technology (NIST). The corpus is now available from the Linguistic Data Consortium (LDC).

The implementation contains two programs, one which only runs once, to preprocess the corpus, and one which is used for actual spell checking. The preprocessing program starts by replacing every character that is not an Ascii letter by a space. Although that will split abbreviations into two words, e.g., $doesn + t$, this is a common approach in NLP software. After that, students

need to create dictionaries of the frequency of each character and of each character pair. Then they need to build a dictionary of words in the database with their frequencies. They also need to save the total number of words in the database and the number of distinct words. All of these data will be used by the spell check program.

Regular expressions and stream or lambda processing can be useful for this process. Since the data volume is so large, there is a large difference between the most and least efficient implementation. For this reason we ask students to time their program. We also ask them to serialize the result so that the data collection does not have to be repeated.

The second program uses the generate and test model, which is a common AI approach [7, 139-143 ff.], although it is usually new to students. Under this paradigm, the program contains two sections, one to generate possible words and the other to evaluate them. After that, students need to sort the evaluated words and display them in descending order of probability in order to model what commercial spell checkers do. We only run the algorithm on words that are not in the dictionary that the first program has built, since we assume that words in the corpus are correct. Looking at the algorithm, students believe the actual spell check program will be inefficient, but it actually runs very rapidly.

The spell check program needs to calculate $P(s|w)P(w)$ for every possible correction. Fortunately, we do not need to look at every word in the dictionary as the math above might indicate. *kernighan [3] suggest that most spelling errors belong to one of four types:

a) Add any letter at any position (including at the beginning or end of a word).
b) Delete the letter at any position.
c) Substitute the letter at any position with any other letter.
d) Transpose any two consecutive letters.

Kernighan calls each of these single-character errors, including the last. We can accept their research that these are the most common kinds of errors without necessarily considering transposition to be a single-character error. In any case, students can see by experiment that these are the categories used by Microsoft Word, for example, although Microsoft Word does include one additional category, namely the insertion of a space inside a string to create a two-word unit. The first step is to make a list of the potential good words that the four rules will give us for evaluation purposes. In every case what we want to calculate is the probability that a good word w will be converted to the given bad string s using one of the four procedures. We always view the change from the point of view of the good word, e.g., if the bad word is *acress* and we are considering the good word *actress*, we look at what it takes

to convert *actress* to *acress*, i.e., deleting the *t*. It is necessary to explain this with examples, as students find it unintuitive.

To do this, students need to process the given string four times. Items *a)* and *c)* involve two nested loops, an outer loop to maintain the position of the letter in the error string and an inner loop for the correction letter. Items *b)* and *d)* can be handled with a single loop. For each item, we need to keep track of the type of correction and which letters were involved. We also tell students to keep track of where in the word the correction was located to make debugging easier.

In the second step, we prune the list by looking up each possible correction from each of the four categories in the dictionary. We drop any possible correction that is not in the dictionary, i.e., any string which is not a word or is so rare that it does not appear in the corpus at all. Most of the potential corrections are dropped at this point.

For each of the four categories, we ask students to print the number of possible corrections and those that are left after pruning. In addition to being a useful debugging aid, this shows students that the number of words they need to evaluate is extremely small compared to the number of words in the dictionary. For example, Table 1 shows the results for *acress*.

Table 1: Potential Corrections for *acress*

Category	Count	Pruned
add	6	3
delete	182	1
substitute	156	4
transpose	5	1

Now that we have a short list of corrections, we need to calculate $P(s|w)$ and $P(w)$ for each word. To obtain $P(w)$, we can count how often the word w occurs in the corpus, then divide by the total number of words in the corpus. It is common to use smoothing so that no count will be zero even though a word may not be found in the corpus. Kernighan uses Good-Turing smoothing, which adds .5 to every word count. To keep the numbers proportional, one also needs to add an equivalent correction to the denominator, so that instead of $P(w) = count(w)/N$, where $count(w)$ is the number of occurrences of w and N is the total number of words in the corpus, we have $P(w) = (count(w)+.5)/(N+.5V)$, where V is the number of distinct words in the corpus.

We can get $P(s|w)$ by using the heuristics suggested by *kernighan [3]. We assume that a particular letter substitution is independent of the word it is located in and its place in the word. Therefore we just need to know how

common each particular error is. Kernighan provides this data in his paper in the form of four 2-D tables called confusion matrices. I scanned these tables and provided them to students. The four tables are defined as follows:

$$\begin{aligned} del(x,y) &= \text{how often } y \text{ was deleted after } x, \\ &\quad \text{i.e., the user typed } x \text{ instead of } xy \\ ins(x,y) &= \text{how often } y \text{ was inserted after } x, \\ &\quad \text{i.e., the user typed an extra } y \text{ after } x \\ sub(x,y) &= \text{how often the user typed } y \text{ instead of } x \\ trans(x,y) &= \text{how often the user typed } yx \text{ instead of } xy \end{aligned}$$

These four tables can be used to calculate $P(t|c)$ as follows, where $P(t|c)$ represents the probability that the typo letter t was used instead of the correct letter c:

$$P(t|c) = \begin{cases} del(w[p-1]), w[p])/bigrams(w[p-1], w[p]) \\ ins(w[p-1], w[p])/unigrams(w[p-1]) \\ subst(s[p], w[p])/unigrams(w[p]) \\ trans(w[p], w[p+1])/bigrams(w[p], w[p+1]) \end{cases}$$

where p is the location in the good word where the error happened and *unigrams* and *bigrams* are the counts of single letters and letter pairs calculated by the corpus analysis program. If a deletion or insertion happens at the beginning of a word, $p = 0$ and $p - 1$ refers to an imaginary slot before the first letter.

In other words, the probability that t was deleted before c is the ratio of the frequency of the error pair tc to the total frequency of tc in the corpus. The probability that t was added before c is the probability of the frequency of the error pair tc compared to the frequency of the letter c. The probability that t was substituted for c is the ratio of the error pair tc to the frequency of c. Finally, the probability that t and c were swapped is the ratio of the error pair tc to the total frequency of the letter pair tc. Basing the choice of the previous character instead of the next character is arbitrary, and students have difficulty with that concept. They also have difficulty with inserting letters at the beginning of a word, so examples are useful here too.

We now ask students to build a chart like Table 2. $P(total)$ is $P(x|word) * P(word)$, multiplied by 10^9 for easier reading. The italicized words are incorrect words that are present in the corpus. This issue is discussed later in the paper. To create the chart, students need to save all the data for each line in a data structure such as an object, make a list or other type of collection

of the objects, then sort the items so that the chart prints in descending order of probability. To use the sort function provided by the runtime system (e.g., in Python or Java), they need to write a comparison function to compare two objects. Many students are not used to the idea of building a temporary data structure just to format output correctly. In addition, although they have studied sort comparators in a previous course, most are not comfortable with the concept.

Table 2: Sample Output for *acress*

Cand	Type	Pos	C	E	x|w	P(x|word)	P(word)	P(total)
actress	del	2	t	c	c|ct	0.0001165315	0.0000584117	6.806804
across	subst	3	o	e	e|o	0.0000073322	0.0002452899	1.798504
acres	ins	4	–	s	es|e	0.0000198875	0.0000818941	1.628671
acres	ins	5	–	s	ss|s	0.0000172377	0.0000818941	1.411670
acrss	ins	2	–	e	re|r	0.0000079273	0.0000000420	0.000333
adress	subst	1	d	c	c|d	0.0000021981	0.0000000981	0.000216
agress	subst	1	g	c	c|g	0.0000014294	0.0000000701	0.000100
caress	trans	0	ca	ac	ac|ca	0.0000000000	0.0000001261	0.000000
access	subst	2	c	r	r|c	0.0000000000	0.0000665661	0.000000

The tables were provided as two-dimensional arrays. Students were encouraged to convert this format to a two-dimensional associative array for easier lookup. Although this requires only a few lines of code and avoids the need to convert between letters and numeric subscripts, it requires a sophisticated use of list comprehensions and many students did not bother.

The outputs have been designed to make the project easy to grade. As part of the real-world approach to the assignment, integer columns in the output must be right-aligned and real numbers must be decimal-aligned. Both rows and columns must be labeled. The outputs of the data analysis program include the unigram table, the bigram table, a few items from the frequency dictionary (e.g., the totals and the ten most common words and their frequencies), and the timing information. For the spelling correction program proper, the program prints summary data about the creation and pruning of possible corrections in addition to the correctly sorted final table. In addition, the second program will not run if the serialization file has not been correctly implemented.

4 Lessons Learned

This assignment was tested in an upper-level elective course containing 35 students. The course included a short introduction to Python and had as pre-

requisite our regular data structures course, which is taught in C++ using the STL. Even with the detailed directions provided, about 20% of the class needed help understanding the assignment in office hours. In addition, it was necessary to provide smaller test files with answers to enable independent student debugging. Python *append* recopies the list for every new element. Several students found it hard to believe that using this function instead of linear list processing, thus converting the algorithm from $O(n)$ to $O(n^2)$, was the cause of their program not running in the time allocated by the department server.

In annual surveys, our graduates express the desire for larger and more unstructured assignments. Our regular data structures course gives students weekly practice with new data structures. This assignment gives them an opportunity to integrate that material in a larger project. Students provided very positive evaluations of the project, with several putting it on their resume as a project they had done. However, given the class size, detailed directions are required, i.e., most of the students are not yet ready to design a program from the description of the problem. Further evaluation is required to see whether giving one or more such assignments in a course would increase students' design ability, and/or whether the design skills could be taught explicitly.

Evaluation of the code shows a wide variety of coding skill, from professional level to code that would horrify anyone who teaches programming. Many students did not use list comprehensions except where they were explicitly required. Several students used unnecessarily complex library functions that the instructor had not expected, and several would have preferred to write their own sort, even though writing a sort comparator function is a matter of a few lines.

There are advantages and disadvantages to giving the assignment as a two program set connected by a serialized file. It gives students a better idea of how the whole system works, and helps them realize that if they don't create the data in the first program correctly, they won't be able to run the second program. The flip side of that argument is that it is more difficult to grade the second program if students' first program doesn't work. For that reason it might be better to define the layout of the serialization file and give students who need it a copy of that file so that they can start from scratch on part 2.

It would be possible to add library usage to the software engineering goals for this assignment by putting the tables in a library instead of simply giving them to students to copy and paste. One could give the tables out as 2-D associative arrays so that students get practice using that format, or one could precede this unit by a unit specifically designed to teach list comprehensions, and then require their use in this assignment.

Although an advantage of using the AP '88 corpus is that one can come close to Kernighan's original numbers, use of that corpus requires a paid license

because it is not open source. Since we cannot duplicate Kernighan's exact corpus in any case, and any large corpus should give similar numbers, I would use an open source corpus next time.

In the real world, it would be more appropriate to add together the two ways to get *acres* and print one line with the total probability, but leaving it as two lines and mentioning the issue in class is preferable for two reasons. Students can see that there are two ways to generate that word, and it makes debugging simpler.

Something that makes students uncomfortable is that the algorithm can produce invalid possible corrections if there are typos in the original corpus, such as the italicized words in Table 2. Our version of the AP '88 corpus had 161,817 unique words, meaning that just using a commercial spell checker on the unique words alone would be a significant project, as the corpus contains a large number of proper nouns, common nouns that are capitalized (either the first letter or the whole word) because of context, expressions such as "aaah", and other cases that need to be individually adjudicated. So we left the incorrect words in the corpus and discussed the issue in class instead.

Dropping words that are not in the corpus from the list of possibilities removes most of the incorrect words, but also removes a correct but rare word like *cress*. That removes one of the points of using Good-Turing smoothing. In addition, we do not adjust for zero terms in the confusion matrices, so that, for example, *caress* and *access* in our table have probabilities of zero even though they are valid words that the user could have typed.

It will be noted that our results do not match Kernighan's because we were not able to obtain the exact corpus that he used. His paper [3] gives the following words in sequence, as does Jurafsky's popular NLP textbook [2, 163-167], which is based on Kernighan's paper:

actress, *acres* (twice), *across*, *access*, *caress*, *cress*

5 Conclusions

Bayesian spelling correction has been a successful addition to the standard assignments in a data structures course. Students recognize the task but have no idea how it's done. No individual step is hard, but students need to do many steps correctly to accomplish the result. The algorithm has a solid mathematical base. Once students understand it, they can appreciate differences in the results from different commercial spell checkers, and they tend to have fairly good intuition about which words are more common.

This algorithm is useful for teaching students something about natural language processing without having to spend class time on basic linguistics. It is

also useful as an assignment where the database is large enough that students can't just eyeball the data to get the answers. Additionally, it is CPU-intensive enough that an efficient implementation really makes a difference, something that has become less common as machines have become faster. There is an opportunity to use many of the concepts that need reinforcing, such as associative arrays, regular expressions, streams or lambda processing, serialization, and custom sort orders. Finally, the program is not available in the standard repositories. Although a few students will discover the blog post [6] where Peter Norvig solves the problem in a few lines, that only provides a small portion of the code needed for this assignment.

References

[1] Bayes, T. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London 53* (1763), 370–418.

[2] Jurafsky, D., and Martin, J. H. *Speech and Language Processing*, 2 ed. Prentice Hall, Upper Saddle River, NJ, 2008.

[3] Kernighan, M. D., Church, K. W., and Gale, W. A. A spelling correction program based on a noisy channel model. In *COLING '90* (1990), vol. 2, pp. 205–211.

[4] Linguistic Data Consortium. Tipster v. 2, 2018. `https://catalog.ldc.upenn.edu/LDC93T3C`.

[5] National Institute of Standards and Technology. Text retrieval conference, 2018. `https://trec.nist.gov`.

[6] Norvig, P. How to write a spelling corrector, 2016. `https://norvig.com/spell-correct.html`.

[7] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 3 ed. Pearson, New York, 2009.

Courses of Blockchain Technology to Teach at a Career-Oriented College*

Poster Session

Penn Wu
Computer Information Systems
DeVry University
Sherman Oaks, CA 91403
pwu@devry.edu

Blockchain technology has evolved from a hype to a technical reality and is quickly becoming a demanded skill. Career-oriented colleges are facing the challenge to incorporate blockchain technology into existing curriculum. While lacking instructional and curriculum models, educators must (a) identify the knowledge domains of blockchain technology, (b) classify their topical areas, and (c) place them in suitable academic subjects (programming, management, or business laws), before being able to determine the appropriate level of instructions and pedagogies. This paper describes a model designed and used to evaluate what topical areas of blockchain technology are practical to teach at a career-oriented college. The evaluation is based on seven factors: (1) maturity of technology, (2) impacts on management, (3) real-world applications, (4) subject classification, (5) knowledge prerequisites, (6) textbook readiness, and (7) recommended pedagogies. The evaluation results are presented by a rating scale ranging from 1 to 5 indicating the degree of "appropriateness" of instruction at college level. A set of territorial indicators also help to clarify the subject territory (computer science, business, or law). This paper suggests to: (1) offer all majors with an introductory course for students to understand what blockchain is and how it applies to business, (2) teach a cryptocurrency-centered course that engages students of technical majors with in-depth discussion of sciences and technologies that drive blockchain, and, if instructionally feasible, (3) guide CS and CIS majors through a project-based programming course for experiencing how to code blockchain applications.

Promoting Conscientious Discussions in Computer Science Ethics Instruction*

Poster Session

Deborah G. Tatar, Michael Stewart and Chandani Shrestha
Computer Science
Virginia Tech
Blacksburg, VA 24060
chandani@vt.edu

This research addresses the difficulty that the instructors and students in Computer Science (CS) have in conducting conscientious discussions in the classroom. Having conscientious discussion is a prerequisite for effective instruction in ethics in CS as mandated by the CS accreditation board and professional organizations. The "draft, depict, depose" pedagogical strategy works in conjunction with ThoughtSwap, a web application. An instructor shares a prompt that students answer through real-time, anonymous writing, thus drafting their "thoughts". The submitted collection of thoughts can be redistributed, so that each student will receive another student's thought, which may be contrasting or similar to the original thought. At this step the students can analyze the received thought in small group discussion and represent it, thus also depicting the thoughts that do not belong to them. Finally, small group discussions and thoughts from drafting and depicting process may be presented to the whole classroom, thus deposing the summarized thoughts. The "draft, depict, depose" learning pattern supplemented by ThoughtSwap, permits movement from individual anonymous thought sharing in written form, to larger group discussion. ThoughtSwap is a practical pedagogical tool. Students can join at any stage. The instructor may reorder thoughts, draw one particular thought to the attention of the class and even, if appropriate, remove thoughts (though we do not recommend this). Our research examines the overarching question of whether ThoughtSwap and the associated learning pattern, "Draft, Depict, Depose" can promote conscientious discussions in classroom among CS students. Student-level outcomes are based on demographic distributions, self-reported openness to discourse and discussion with others, and observed participation through and outside the tool.

Analyzing Relationship: Twitter Tweet Frequency With the Stock Prices of Telecom Companies*

Poster Session

Amrita Shelar and Ching-yu Huang
Computer Sciecne
Kean University
Union, NJ 07083
chuang@kean.edu

Twitter is a widely used online social media and it has been used to discover the relationship between data. We investigate whether the daily number of tweets that mention any of the 4 telecom companies - Verizon, T-Mobile, ATT and Sprint vis-à-vis stock prices. The datasets we studied are Twitter frequency (tweets that mention the name of either of the companies) and stock prices for a period of 14 days. We carried out ETL (Extract, Transform, and Load) process to obtain clean and refine data sets for analyzing. We have discarded the data of ATT and Sprint later because their names were shown in another event during that period and not reflected the real data of these two companies. The Tweepy library of python is used to connect to Twitter Streaming API and download the tweets from Twitter.

The study focuses on correlating data sets of Twitter tweet frequency with the stock prices of telecom Companies; using Statistical Methods: Z-score and Chi-Square – Test of Independence with data visualization. Our results demonstrate the relation between daily numbers of tweets is correlated with that of stock price for Verizon and T-Mobile. Our preliminary results also demonstrate the relation of frequency of tweets with stock prices of each day. Furthermore, it appears that the daily number of tweets is not correlated with the stock prices. The result may vary depending on if the data set is captured for a longer period, say months or a year to have a more concrete analysis and study their relationship.

A Strategic Approach for Adapting the CUDA Course at Computer Science Departments Within United States Universities*

Poster Session

Imad Al Saeed
Computer Science Department
Saint Xavier University
Chicago, IL 60655
alsaeed@sxu.edu

The purpose of this study was to encourage United States universities to offer CUDA courses as one of the main courses within the computer science department. This study included a target population of professors of both genders who were teaching CUDA at 100 universities. The general findings indicated that CUDA is an important technology and should be taught as one of the main courses within all computer science department at United States Universities. This study provided a business plan that addressed all the resources needed to create new CUDA courses. It also offered an effective solution for the financing problems that might prevent other schools from offering CUDA courses.

Integrative Curriculum for Teaching Databases*

Poster Session

Ching-yu Huang
School of Computer Science
Kean University
Union, NJ 07083
chuang@kean.edu

In today's age of big data, database (DB) is a particularly critical subject in Computer Science because data management is a core business for many companies, regardless of industry. Many software developers, data analysts, and other positions that need to access data and manage users require applicants to have a solid DB foundation. Before a student can take a DB class, they should at least have an intermediate knowledge of programming and set operations. Therefore, many colleges require students to first take Discrete Math and Data Structures before taking DB. This means that students typically only get DB exposure late in their college education; however, in order to better shape their career paths, students should be getting earlier DB exposure. Additionally, the curriculum of a traditional DB course focuses on relational DB and Structure Query Language (SQL), even though the NoSQL DB is gaining much more popularity in the workplace. Furthermore, in the workplace, web-based applications are in high demand because they are platform independent and can be accessed worldwide. For web database application developers, it is important to learn how DB works with different technologies. Traditional DB curriculum neglects this practical aspect, and often solely focuses on theory. This paper proposes an integrative curriculum for teaching DB that interweaves essential theoretical DB concepts, as well as hands-on practical DB experience.

Teaching Artificial Intelligence for Undergraduate Students: A Project Based Approach*

Poster Session

Weidong Liao and Osman Guzide
Computer and Information Sciences
Shepherd University
Shepherdstown, WV 25443
wliao@shepherd.edu

Artificial Intelligence has advanced rapidly in recent years due to the progress in hardware performance and algorithmic studies. The job market for artificial intelligence has been promising as many practical applications have become viable nowadays. Over the years, many universities and colleges offers an introductory artificial intelligence course to undergraduate students. With the progress in the area of artificial intelligence and machine learning, one introductory AI course is insufficient and an approach to integrating artificial intelligence across undergraduate curriculum is essential and appealing. In this poster, we present our project based approach to integrating artificial intelligence and machine learning concepts to a variety of undergraduate courses. A number of diverse projects will be discussed, ranging from hardware based projects used in our Computer Organization course, to deep learning applications used in our Web Programming course.

What GDPR Means for Data Privacy*

Poster Session

Susan S Conrad and Mohammed Alghamdi
School of Business
Marymount University
Fairfax, VA 22030
susansconrad@gmail.com

Data privacy/personal data has drawn a lot of attention from people in sectors of commerce, technology, real time data processing, information technology and the banking industry. With the adoption of the European Union General Data Protection Regulation (EU GDPR) data privacy policies must be modified to meet the new requirements. It is estimated that more than 50must modify business practices to meet the requirement (www.inc.com, 2018) The days of collecting data and selling it without easy-to-read user consent is no longer allowed under GDPR. Data portability allowing users to request a report of data stored about them in a readable format, becomes a key concern for companies such as Google, Yahoo, Microsoft, Amazon and others.

This poster provides some context to what personal data and data privacy is interpreted by multiple organizations and countries. It discusses what types of information are considered to be personal information and the issues associated with keeping this information private. It briefly discusses how the EU GDPR impacts the definition of the personal information and how the right to be forgotten will impact privacy and data storage

Engaging HBCU Faculty in Project-Based Learning in Silicon Valley*

Panel Discussion

Faculty that teach computer science at historically black colleges and universities (HBCUs) have unique experiences and challenges that impact the way they create curricula. This special session will discuss challenges facing HBCUs and how industry- academia partnerships can assist in creating curricula that is not only engaging but helps to involve the students in their own education using project-based learning. We will also explore how HBCUs can work together to develop interactive, creative shared knowledge that will help prepare students for the future; whether that be careers in Silicon Valley or graduate school. This panel includes participants from the Google-sponsored workshop Faculty In Residence (FIR) Program that was held in the Summer of 2017.

Faculty in Residence (FIR) program was envisioned to assist faculty at HBCUs bridge their prior experience with curriculum development and allow them to be immersed in Silicon Valley culture and practices. The six-week experience permitted approximately thirty faculty from HBCUs to work alongside Silicon Valley engineers to create curricula that is designed for teaching introductory programming courses, applied data structures, mobile applications, machine learning, or cybersecurity. Participants engaged in workshops related to active learning, flipped classroom, code reviews, and technical interview do's and don'ts. Also, faculty were educated about the successes of existing programs like Google's Computer Science Summer Institute, Hampton University's Applied Computer Science Summer program, and the HBCU Google In Residence Program.

Prior to attending the FIR Workshop, faculty members were asked to create at least two goals that they would work to achieve as a result of the program. These goals had to relate to curriculum development, research connections, and/or work products that could be used directly in the classroom for the upcoming Fall semester. Additionally, faculty were asked to think out of the

ordinary curriculum box and participate in design thinking to create interactive, fun, yet challenging learning experiences.

To support the work being performed by the FIR participants; Google also delivered events and opportunities that faculty could take back to inform their students on the technical interview process. Faculty were informed of Google's "Skill Builder Series" that delivers information on topics such as interview preparation, maximizing an internship, and technical skills (language specific, platform specific development, software version control).

Panelists include faculty from Howard University, Shaw University, and Coppin State University each with their own unique experiences from FIR. Questions to be addressed during this panel include:

- What are current strategies (specific to Howard University, Shaw University, etc.) that have been working at HBCUs to increase diversity in Computer Science?
- What was the experience of the students coming into the Courses?
- What Was the expectation (if any) of the faculty coming into the FIR Workshop?
- What techniques were faculty able to leverage successfully in their classroom after the FIR Experience?
- What are the lessons learned or improvements that you think Google and HBCUs can leverage for future FIR activities?

Panelists

The faculty chosen to participate on this panel in no way represent all the innovative persons that participated in the six-week thought-provoking workshop. However, they represent the varied missions and cultures of HBCUs today showing that these institutions are not homogenous. These faculty biographies are provided below.

Legand Burge III, Ph.D. is a Professor of Computer Science at Howard University in the Electrical Engineering and Computer Science Department (EECS). Dr. Burge was instrumental in creating various academic-industry partnerships for Howard University, including the Google In Residence program[1]. The GIR program is designed to bring Silicon Valley engineers into HBCUs to teach programming courses from an industry perspective. This program and its success has spread to approximately ten HBCUs and continues to gain popularity. Additionally, Dr. Burge was a key architect in helping to think through the FIR program with Google thought leaders. He realized that one company

alone cannot reach all the 110 HBCUs in the U.S. He knew that teaching the faculty industry practices could help underrepresented students gain a leg-up in the technical interview process and in Silicon Valley practices. Additionally, Dr. Burge was one of the key architects of establishing and outlining the Howard-West initiative with Google that started immersing HBCU students in Silicon Valley culture, projects, and problems.

Dr. Burge's research interests lie in the field of distributed computing and in global resource management in large-scale distributed systems. Dr. Burge is currently the director of the Distributed Systems Research Group (DSRG) and associate director of the Center for Applied High Performance Computing at Howard University. Dr. Burge is also interested in Computer Science Education and Diversity, and Tech Innovation and Entrepreneurship. Dr. Burge is a AAAS Fellow, and Fulbright Scholar recipient.

Leshell Hatley, Ph.D. is an Assistant Professor of Computer Science at Coppin State University in Baltimore, MD and is the Founder and Executive Director of Uplift, Inc., a nonprofit STEM education organization. Uplift offered the first mobile application development course to middle and high school students in the US and is one of the first organizations to offer after school courses in Lego Mindstorms Robotics in Washington, DC. She and the students in her research lab at Coppin State, the Lab for Artificial Intelligence and its Applications (LAIA), won the 2016 USA White House HBCU (Historically Black Colleges and Universities) Maker Innovation Challenge and she recently led the first Coppin State Team to compete in the NASA Swarmathon. She is a passionate computer engineer, educator, and researcher who continuously combines these three attributes to create innovative approaches to teaching STEM concepts to students between that ages of 3 and 73. With over 20 years of teaching experience, Dr. Hatley leads teams of enthusiastic students, dedicated volunteer instructors, and teams of engineers to achieve award winning success, national news coverage, and innovative technology product designs. Gloria Washington, PhD

Gloria Washington (Moderator), PhD is an Assistant Professor of Computer Science at Howard University. At Howard, she runs the Affective Biometrics Lab and performs research with her students on human-centered computing and computer science education. She currently teaches the introductory programming courses at Howard using the C++ programming language. Through her work in creating engaging curricula for these classes; she is also researches technologies that can measure the engagement of the students and adapt lessons as needed. She led the first woman graduate from Howard's newly minted Computer Science PhD program. Additionally, she runs a coding and tech entrepreneurship program called #WatchMeCode Tech Innovation Experience. Through this program, she works with local high schools in the

Washington, DC area to assist teachers with introducing coding to their students. With the #WatchMeCode program, she hires currently enrolled Howard University comp sci. majors to teach introductory, fun subjects to young black males such as Coding Robotics Using Spheros and Block Chain Programming for High-Schoolers. This helps the Howard students take ownership of the work they are learning in their classrooms.

Before coming to Howard University, she was an Intelligence Community Postdoctoral Research Fellow in the Department of Computing Science at Clemson University. She performed research on identifying individuals based solely from pictures of their ears. Dr. Washington has more than fifteen years in Government service and has presented on her research throughout industry. Ms. Washington holds M.S. and Ph.D. in Computer Science from The George Washington University, and a B.S. in Computer Information Systems from Lincoln University of Missouri an HBCU.

Lloyd Williams, Ph.D. is an Assistant Professor at Shaw University in Raleigh, NC. He received a B.S. in Philosophy from Vanderbilt University. After completing a Ph.D. in computer science from NCSU, he joined the CS faculty at Shaw University in 2011. He brought a passion and love of students to teaching that drew his students to computer science. He was promoted to Department Chair for Computer Science two years later and Program Director for Science and Technology Innovation three years after that. He has personally mentored minority computer scientists who have gone on to work at companies including NASA, Intel, Google, Glaxo, US DOD, US DHS, Fidelity Investments, Cisco, ATT, Lenovo and IBM at starting salaries as high as $80,000. He was recognized as the 2017 STEM Educator of the Year by the Research Triangle Park Foundation's STEM in the Park Initiative.

Dr. Williams personally created the Shaw Innovation Laboratory, taking a former storage room and creating a lab that has been awarded over $600,000 in funding in 2016 alone, including a recent $400,000 NSF award to create the Shaw University Center for Computer Science Living, Learning and Research. His Shaw Innovation Lab has brought cutting edge to students both at Shaw and from K- 12. Dr. Williams founded the Shaw Computer Club in 2011 (www.shawcc.com). The club has conducted STEM outreach with K-12. The club has striven to create strong ties with industry, including Citrix, Red Hat, Cisco and HP. He has hosted Google engineers to teach app development to Shaw students for two semesters and he worked as a Faculty in Residence at Google's Mountain View headquarters in Summer 2017 where he created an innovative virtual reality project that he has incorporated into his software engineering class for Fall 2017 and hopes to deploy at other schools as well.

Acknowledgment

This work is supported by all thirty faculty members that participated in the FIR program. The participating institutions included Bennett College, Johnson C. Smith University, Fort Valley State University, Dillard University, Xavier University, Prairie View AM University, Claflin University, Coppin State University Norfolk State University, Hampton University, Shaw University, Spelman College, Morehouse College, Morris College, Howard University, and Harris Stowe University.

References

[1] Washington, A. N., Burge, L., Mejias, M., Jean-Pierre, K., and Knox, Q. Improving undergraduate student performance in computer science at historically black colleges and universities (hbcus) through industry partnerships. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (2015), SIGCSE '15, pp. 203–206.