The Journal of Computing Sciences in Colleges

Papers of the 21st Annual CCSC Northeastern Conference

April 12-13, 2019
University of New Haven
West Haven, CT

Volume 34, Number 6 **April 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
Mercy College

Foreword

Welcome to the 2019 issues of our journal for the CCSC spring 2019 conferences: Southwestern (March 22-23), Central Plains (April 5-6), South Central (April 5), Mid-south (April 12-13), and Northeastern (April 12-13).

Please plan to attend one or more conferences, where you can meet and exchange ideas with like-minded computer science educators. Each conference covers a variety of topics that are practical and stimulating. You can find detailed conference programs on the conference websites, which are listed on the CCSC conferencecalendar: `http://www.ccsc.org/regions/calendar.`

From January 2019, this journal will be published electronically on the CCSC website and links to the journal issues will be sent to CCSC members via email. Those of you who would like hard copies of journal issues can order them from Amazon. Simply search for "CCSC Journal" to find available issues. The journal will continue to be available in the ACM Digital Library.

As an author, you may post your papers published by CCSC on any website. Please make sure to use the PDF versions of your papers with CCSC's copyright box. Such PDFs can be downloaded from the ACM Digital Library or extracted from our electronic journal.

Please feel free to email me directly at `blu@sbuniv.edu` if you notice any issue with our publications.

Baochuan Lu
Southwest Baptist University
CCSC Publications Chair

Welcome to the 2019 CCSC Northeastern Conference

Welcome to West Haven, Connecticut and the University of New Haven, for the Twenty-Fourth Annual Consortium for Computing Sciences in Colleges Northeast Region Conference. The conference is held in cooperation with the ACM SIGCSE and Upsilon Pi Epsilon Honor Society.

Our program features two distinguished invited speakers, Pete Wurman, Vice President of Engineering at Cogitai and Jia Chen, Offering Leader of Blockchain Solutions for Healthcare and Life Sciences at IBM's Innovation and Solution Incubation Team. The conference has a diverse and engaging program that includes paper presentations, lightning and encore talks, workshops, tutorials, and faculty and student research poster presentations. On Friday morning, we are hosting our traditional programming contest. On Friday afternoon, we have two student-focused sessions: a student "unconference" and a programming problems discussion session to allow participants and organizers of the programming contest to review and analyze problem solutions.

Our thanks go to a remarkable conference committee and highly invested board. Their inspiring and diligent work has ensured the success of this conference. We are also very fortunate to have worked with dedicated and thorough reviewers, enthusiastic session chairs, and outstanding student and staff volunteers at University of New Haven. The conference continues to be selective; we accepted 9 of 23 papers for an acceptance rate of 39%. This continues to ensure the high-quality program of a widely recognized regional conference.

We hope you find the conference informative and engaging, meet new colleagues, and get new ideas to contribute to computing education in Northeastern Region. If you are interested in volunteering for our conference, we encourage you to attend the CCSCNE Business Meeting on Saturday afternoon. We also look forward to seeing you next year at Ramapo College of New Jersey.

Alice Fischer
University of New Haven
Mark Hoffman
Quinnipiac University
Conference Co-chairs

2019 CCSC Northeastern Conference Committee

Alice Fischer, Conference ..... University of New Haven
Mark Hoffman, Conference ..... Quinnipiac University
Ed Harcourt, Program ..... St. Lawrence University
Ali Erkan, Papers ..... Ithaca College
Yana Kortsarts, Papers ..... Widener University
Susan Imberman, Panels ..... The City University of New York
Joan DeBello, Lightning Talks ..... St. John's University
Bonnie MacKellar, Tutorials and Workshops ..... St. John's University
Ting Liu, Tutorials and Workshops ..... Siena College
Daniel Rogers, Faculty Posters ..... The College at Brockport
Ingrid Russell, Speakers ..... University of Hartford
Mike Gousie, Speakers ..... Wheaton College (Massachusetts)
Karl Wurst, Student Unconference ..... Worcester State University
Jacob Aguillard, Student Unconference ..... Worcester State University
Darren Lim, Encore ..... Siena College
Sandeep Mitra, Undergraduate Posters ..... The College at Brockport
Liberty Page, Undergraduate Posters ..... University of New Haven
Jim Teresco, Undergraduate Posters ..... Siena College
Aparna Mahadev, Undergraduate Posters ..... Worcester State University
Mark Hoffman, Registration ..... Quinnipiac University
Stefan Christov, Registration ..... Quinnipiac University
Frank Ford, Programming Contest ..... Providence College
Del Hart, Programming Contest ..... SUNY Plattsburgh
Benjamin Fine, Programming Contest ..... Ramapo College
Christopher Martinez, Programming Contest ..... University of New Haven
Tim Chadwick, Career Fair co-Coordinator .... University of New Hampshire
Kevin McCullen, Vendors ..... SUNY Plattsburgh
David Benedetto, K-12 Coordinator ..... New Hampshire Department of Education

Regional Board — 2019 CCSC Northeastern Region

Lawrence D'Antonio, Board Representative .. Ramapo College of New Jersey
Mihaela Sabin, Editor ..... University of New Hampshire at Manchester
Mark Hoffman, Registrar ..... Quinnipiac University
Adrian Ionescu, Treasurer ..... Wagner College
Stoney Jackson, Webmaster ..... Western New England University

Reviewers — 2019 CCSC Northeastern Conference

Chris Alvin ..... Furman University, Greenville, South Carolina
Dan DiTursi ..... Siena College, Loudonville, New York
Alfreda Dudley ..... Towson University, Towson, Maryland
Cenk Erdil ..... Sacred Heart University, Fairfield, Connecticut
Michael Filippov ..... Rivier University, Nashua, New Hampshire
Benjamin Fine ..... Ramapo College of New Jersey, Mahwah, New Jersey
Timothy Fossum .... Rochester Institute of Technology, Rochester, New York
Seth Freeman ..... Capital Community College, Hartford, Connecticut
Sally Hamouda ..... Rhode Island College, Providence, Rhode Island
William Harrison ..... St John Fisher College, Rochester, New York
Sarah Huibregtse ... Rochester Institute of Technology, Rochester, New York
Jeremiah Johnson ... University of New Hampshire at Manches, Manchester, New Hapshire
Zach Kissel ..... Merrimack College, North Andover, Massachusetts
Bradley Kjell Central Connecticut State University, New Britain, Connecticut
Lisa Lacher ..... University of Houston - Clear Lake, Houston, Texas
David Levine ..... Saint Bonaventure University, Allegany, New York
Qian Liu ..... Rhode Island College, Providence, Rhode Island
Matija Lokar ..... University of Ljubljana, Ljubljana, Slovernia
Joan Lucas ..... College at Brockport, SUNY, Brockport, New York
Robert McCloskey ..... University of Scranton, Scranton, Pennsylvania
Muath Obaidat ..... City Univeristy of New York, New York, New York
Suhaib Obeidat ..... Bloomfield College, Bloomfield, New Jersey
Pat Ormond ..... Utah Valley University, Orem, Utah
Sofya Poger ..... Felician University, Rutherford, New Jersey
Stefan Robila ..... Montclair State University, Montclair, New Jersey
Thomas Rogers ..... Millersville University, Millersville, Pennsylvania
Nick Rosasco ..... Valparaiso Univeristy, Valparaise, Indiana
Richard Scorce ..... St. Johns University, Queens, New York
Joo Tan ..... Kutztown University, Kutztown, Pennsylvania
Giovanni Vincenti ..... University of Baltimore, Baltimore, Maryland
Yueming Yang ..... Baldwin Wallace University, Brea, Ohio

How Kiva Robots Disrupted Warehousing*

Keynote

Pete Wurman, Vice-President of Engineering at Cogitai

Kiva Systems introduced swarms of agile robots into an industry dominated by stationary conveyor systems. The path from concept through successful startup and eventual acquisition involved challenges on all fronts. Peter Wurman will explain the business problem that motivated the innovation, Kiva technology and the benefits it brought to customers, and the future of applications of robotics in warehouses.

Dr. Pete Wurman, Vice-President of Engineering at Cogitai Pete Wurman is currently Vice-President of Engineering at Cogitai, an AI startup delivering reinforcement learning as a service. Pete is best known for his work as a technical co-founder of Kiva Systems, the Boston-based company that pioneered the use of mobile robotics in warehousing. In May of 2012, Kiva was acquired by Amazon, and has subsequently deployed more than 150,000 robots to Amazon distribution centers. Prior to joining Kiva, Pete was an Associate Professor of Computer Science at North Carolina State University. Pete earned his Ph.D. in Computer Science from the University of Michigan, and his B.S. in Mechanical Engineering from M.I.T.

Transform the Era of Health with Blockchain*

Keynote

Jia Chen, IBM Healthcare Solutions

Today's healthcare system faces several systemic challenges, including complex/inefficient processes, lack of interoperability, data silos, fraud and lack of transparency. Blockchain technology has the potential to bring industry wide transformation to the healthcare ecosystem by reducing costs and frictions, bringing more trust and transparency to multiparty transactions, and even unlocking new sources of revenue for various constituents. We'll discuss examples of leveraging blockchain technology to enhance the fluidity of healthcare information among key stakeholders, leveraging smart contract to reduce administrative costs for value based payment models, and the formation of an open network to drive digital transformation in the industry.

Dr. Jia Chen is an Offering Leader of Blockchain solutions for Healthcare and Life Sciences at IBM's Innovation and Solution Incubation team. She serves on the IBM Academy of Technology Leadership team. She previously led technical strategy at IBM Watson Health Innovation, with a focus on data and AI. Prior to that, Dr. Chen was the global leader of Watson Experience Centers at IBM, responsible for Watson AI client experiences across all Watson group. She held leadership positions for Innovation and client engagement at IBM Corporate Headquarters as well as emerging markets. She was formerly the Director of Health Solutions for Smarter Cities at IBM, and the Director of Technical Sales & Innovation for IBM's Growth Market Units. She led the identification, structuring and execution of first of a kind technology and business initiatives that provide innovative and sustainable differentiation for IBM's clients. Dr. Chen received her Ph.D. in Physics from Yale University. She was named as one of the top 35 technology innovators under the age of 35 worldwide by MIT's Technology Review in 2005, the Best Researcher of the Year by Small Times magazine in 2006 and one of the top 26 tech women innovators at IBM in 2015. She serves on the Yale Graduate School Alumni Association Board.

Teaching Neural Networks in the Deep Learning Era *

Jeremiah W. Johnson
Applied Sciences & Engineering
University of New Hampshire
Manchester, NH 03101
jeremiah.johnson@unh.edu

Abstract

This paper describes the design and evaluation of the first iteration of a standalone course in neural networks aimed at upper level undergraduates and first-year graduate students. The development of this course was motivated by recent state-of-the-art results on challenging tasks in computer vision and natural language processing that have been obtained using deep neural networks, and the subsequent widespread adoption of these models for various applications in industry. The course design emphasizes theoretical understanding and development of applications following existing best practices. Throughout, many unsettled aspects of the underlying mathematical theory of deep neural networks are highlighted, and students are prepared to adapt as current trends and techniques evolve.

1 Introduction

Neural networks are a type of classical machine learning models often covered as part of an introductory machine learning course. These models are one of the earlier ideas in artificial intelligence, dating to the work of McCulloch and Pitts in 1943 [15]. Since then, the popularity of neural networks has waxed and waned, as experimental results and development of the underlying theory have alternately sparked excitement or despair among researchers and practitioners.

The past few years have been a period of notable success for neural networks, as their use has led to remarkable results on challenging tasks in computer vision and natural language processing [9, 12, 20, 7]. This has coincided with and been partially driven by the development of graphic processing units (GPUs) that enable massively parallel computations ideally suited to neural network optimization algorithms, as well as the development of high-quality open-source libraries for research, experimentation, and development with neural networks [3, 1, 17]. The combination of impressive results, widespread availability of relatively inexpensive GPUs, and high-quality software has led to pervasive adoption of these models in industry.

This paper describes the development and implementation of an elective course in neural networks targeted at upper-level undergraduates and first-year graduate students. The focus of the course is narrow: attention is confined to the class of models trainable using backpropagation inside a gradient descent process. The primary goal of the course is to provide students with the background necessary to understand and reproduce results from current research and develop their own neural network models for applications that interest them going forward.

2 Context

2.1 University

In the past few years, several new programs have been launched at the University of New Hampshire's Manchester college, including degrees in Analytics & Data Science in 2015 and a Master of Science in IT degree in 2017. These new programs, along with the existing undergraduate computer science programs, are all housed in the same department, and students majoring in any one of the programs will often take courses from the others. The course described here sits at the intersection of these programs, and is targeted toward both students pursuing degrees in data science and students pursuing degrees in computer science, at both the Bachelors and Masters level. This context informed the development of the course as a primarily application-driven one that would prepare students to use modern neural network models and machine learning best practices. The course does not serve as a prerequisite to more advanced material in machine learning, artificial intelligence, data science, or computer science more generally, and it is assumed that students may have very limited experience with machine learning. Although not a prerequisite for more advanced material, the course may well serve as an introduction to topics that could form the basis for a senior-year Capstone course or a Master's thesis. Although the students expected to enroll in the course might come from dif-

ferent programs, the degree requirements for the target audience insure that everyone in the course would have had several years programming experience in at least two different programming languages, mathematics through calculus, and likely some linear algebra.

Training neural networks is computationally intensive. Most of the neural network models developed in the past several years have been trained on GPUs, which dramatically accelerate training times. At the time of this writing, these cards are for all intents and purposes required to train deep neural networks in a reasonable amount of time, so student access to such cards was deemed a necessity. The author and a colleague received a GPU Education Center Grant from NVIDIA Corp. in 2015 that provided three GPUs for student use. In addition, early in 2018 Google Inc. debuted Google Colab, a cloud-based Jupyter notebook environment for scientific computing. This environment provides free GPU computing resources to users. Students made heavy use of Colab throughout the course.

2.2 Neural Networks

In recent years there has been an explosion of research into neural networks and their applications. A quick keyword search of the Thompson-Reuters Web of Science database shows that approximately 43% of the publications on the topic of neural networks from 1900-present were published in the last six years. Most of the eye-catching results have been obtained using variants of classical feedforward or recurrent neural network architectures, trained using variants of stochastic gradient descent. These architectures are not new; they have been used with mixed results for decades. What has changed in the past few years is the development of training techniques that are successful at training much deeper and more complex versions of these models, along with the development of hardware better suited for this application.

Much recent research has focused on using neural networks to solve applied problems in disciplines such as healthcare or finance. Rather less recent work has been dedicated to resolving the thorny unsettled theoretical issues surrounding neural networks, of which there are quite a few. These range from long-standing issues of model interpretability and bias to recently noted issues such as susceptibility to adversarial attacks [13, 23]. In addition, as new techniques are developed, often new questions are raised and left unanswered even as the techniques are widely adopted. An illuminating example of this is the case of the Adam optimizer, which was introduced in 2014 and remains widely used despite known issues of convergence, generalization, and implementation [10, 14, 18]. A well-designed curriculum for a neural networks course at this point in time must emphasize that current practices are in flux and are likely to change rapidly, and must prepare students to adapt to the changes.

3 Course Design

One of the first curricular design decisions taken when preparing this course was to limit the focus of the course to neural networks trainable via backpropagation using a gradient descent process. This is only a subset of all the neural networks one might consider. However, limiting the scope of the course to this subset has many benefits. Many of the most exciting recent advances have been made with these neural networks, and the mathematical prerequisites to understanding backpropagation and gradient descent are relatively low.

The course described here was taught over a fifteen week spring semester in a hybrid format consisting of readings, online video lectures, and face-to-face meetings. Approximately two-thirds of the course took place online. The course broke down into three loosely defined modules with some overlap: preparatory material, followed by convolutional neural networks and applications in computer vision, and finally recurrent neural networks and applications in natural language processing. The final two weeks of the course offered a survey of recent advances in generative modeling.

3.1 Preparatory Material

The preparatory material in the first module of the course consisted of a review of linear algebra, vector calculus, and fundamentals of machine learning. The linear algebra and calculus review was fast-paced and non-comprehensive, intended to insure only that the students had the mathematical tools necessary to understand backpropagation and gradient descent. Topics included basic operations with matrices and vectors and vector calculus up to the Chain Rule. Attention then turned to fundamentals of machine learning, including model metrics and assessment, overfitting and underfitting, and cross-validation. This portion of the course was approximately four weeks long. At the end of the fourth week, students were given some starter code and expected to complete a vectorized implementation of gradient descent for a one-layer perceptron.

3.2 Convolutional Neural Networks

In the fifth week, dense feedforward neural networks were used to solve an image classification task and a sentiment analysis task. For approximately the next five weeks, attention turned to convolutional neural networks (CNNs). Computer vision has been revolutionized by CNNs, starting with AlexNet in 2012, whose top-5 error rate on the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) improved upon that of the non-neural network based runner-up by more than 10.8% [12]. Only three years later, CNNs brought the top-5 error rate on ILSVCR below the estimated human top-5 error rate [7].

Weeks 5 - 10 formed the heart of the course. During this module the machine learning fundamentals covered in the first module were brought out of the realm of theory and put to use. After an introduction to convolution and pooling, students solved the classical object classification challenge presented by the MNIST handwritten digit dataset, and then considered CNN-based approaches to object recognition more generally, up to and including the introduction of residual neural networks in 2015 [7]. Following this, students read the research paper [24] and then reproduced to the extent possible the model described therein.

Following the object recognition task, the course took a whirlwind tour of several other computer vision challenges, including semantic segmentation and image style transfer [5].

This module also provided a framework in which several additional topics relevant but not specific to CNNs could be introduced, such as optimization algorithms and tuning. Momentum and Nesterov momentum were introduced and discussed [22] as well as adaptive optimization algorithms. Hyperparameter optimization was discussed, as well as overfitting and the current arsenal of techniques to combat it, such as regularization and dropout [21].

Batch normalization, another widely used but controversial technique to accelerate training, was also covered in this module. This is another example of a commonplace method for which the theory is far from settled, but as a component of many recent successful models, it merited inclusion in the course.

3.3 Recurrent Neural Networks

At the ten-week mark, the application focus shifted from computer vision to natural language processing and sequence modeling. This section focused on recurrent neural networks (RNNs) and especially long short-term memory networks (LSTMs), a class of RNNs that utilizes gating mechanisms to mitigate the so-called vanishing gradient problem [2, 8]. The gated recurrent unit (GRU), a modern variant of the LSTM, and bidirectional versions of all of the preceding models were discussed [4, 19].

In addition to RNNs, LSTMs, and GRUs, the natural language focus of this section of the course provided an opportunity to introduce word embedding models such as the celebrated word2vec model [16]. This in turn enabled the use of pretrained word embeddings with RNNs for NLP tasks.

The primary application considered in this section of the course was sentiment analysis, though the techniques introduced are applicable to most sequential data and other applications, such as time series, were briefly discussed.

3.4 Generative Modeling

The last two weeks of the course surveyed two recently developed generative models: variational autoencoders (VAEs) and generative adversarial networks (GANs) [11, 6]. The development of GANs in particular has spurred much novel and exciting research in the past few years. Although these models are fundamentally different than those considered in the rest of the course, each utilizes a gradient-based training process that is not much different than that used in the preceding 13 weeks. That said, VAEs in particular require a higher level of mathematical sophistication than anything else in the course, which combined with the short timeframe limited coverage to only a very brief overview of an area of great interest to the research community.

4 Student Work and Assessment

Students were assessed based on their performance on quizzes, homework, and projects. No exams were given. Quizzes were designed as brief checks to confirm that students were keeping up with the online content and assigned readings in the course, and were automatically graded via course management software. Homework assignments consisted of readings or programming problems with a singular focus, such as a particular model architecture or hyperparameter optimization technique. By the second half of the class a typical homework assignment would involve reading a recent research paper and attempting to implement the technique or architecture described therein, perhaps with some instructor-provided code provided to get started. All programming was done in Python using the libraries Keras and TensorFlow [3, 1].

The first project in the course was introduced in the fifth week, immediately after the introduction of the basic convolutional neural network architecture. This project took the form of a class competition to develop the best model for classifying a dataset of grayscale glyphs of fonts of the letters A - J. The competition was hosted on the data science competition website Kaggle.com, which provided automatic cross-validation and a leaderboard updated in real time for students to track their ranking. At the conclusion of the competition, students then submitted to the instructor a Jupyter notebook containing all of the code necessary to generate their model along with documentation and explanation.

The second project in the course was assigned in the tenth week, shortly after the introduction of recurrent neural network architectures. The project was to classify comments from a database of comments made to Wikipedia edits into six categories that characterized the 'toxic' nature of the comment. This is a multilabel classification problem; a given comment might fall into none,

one, some, or all of the categories. Significantly less support was provided for this project in terms of validation and assessment, and a slight additional challenge was posed by using the mean AUC (Area Under the receiver operating characteristic Curve) as the assessment metric. The AUC is not differentiable and thus cannot be optimized for directly; moreover, the mean AUC is a complex metric that requires better understanding from the student to correctly diagnose model failings and improve model performance.

The final project in the course was open-ended; students were asked to develop a suitable model for an application of their choosing. The project and model were required to be relevant, and students were tasked with selecting appropriate metrics and properly cross-validating their model.

5 Discussion & Conclusions

Based on observation, student evaluations, student survey results, and personal communication, the course described in this paper was quite successful. Over a fifteen-week semester, students were brought very close to the current state-of-the-art in neural networks, despite the fact that some started out with only limited experience with machine learning. This can be attributed to several factors, the most significant of which was the choice to sharply restrict the focus of the course to the subset of neural networks trainable via a gradient descent process. This enabled a much deeper dive into recent work than could have otherwise been taken. The second important factor that influenced the success of the course was the recent development of high quality, high-level, open-source software for developing and training neural networks. The course as described here could not have existed without such software: instead of developing applications of CNNs and studying CNN architectures, we would have had to focus on the technical challenges of implementing convolution, no trivial feat. No doubt the experience of implementing the basic algorithms is valuable; but for the purposes of this course coverage was limited to a survey of implementations and issues surrounding them; students were not required to code them up. The goals of the course dictated the approach taken here; were the course intended to serve as a prerequisite to more advanced coursework or research, this would merit reconsideration.

One of the goals of this course was to insure that students are ready to adapt when the techniques that they learn in the course evolve and change. To achieve this, students were quickly steered away from textbooks and towards original sources as soon as they achieved a basic level of competency with a given topic. This gets the students in the habit of reading and attempting to reproduce current research, a necessary skill for any practitioner. The field at this point in time is unique in that much of the literature is within the reach

of students at this level, and much relevant content hasn't made it in to the textbooks. Students confirmed their satisfaction with this approach, and often requested source materials right away if none were initially provided for a given topic. The success of this approach could be seen in the student's adoption and use on projects and homework assignments of various techniques not taught in class.

The in-class competition was the most successful of the three projects, and is highly recommended to faculty instructing similar courses. The clear objective and competitive nature of the project motivated the students, and the hosting site Kaggle provided a seamless experience, which helped at this relatively early point in the course. Moreover, the difficulty of the project described was ideal, which was an important additional motivating factor: students could make progress on the problem, but it took nontrivial effort to do so. The toxic comments project was perhaps the least satisfactory precisely because the problem proved quite hard for the students to make progress on, and that led to some frustration as students flailed away at it without seeing any improvement. The students would likely have benefited from additional exposure to and experience with metrics such as AUC prior to this project. The open-ended project that concluded the course led to numerous interesting applications ranging from object detection to speech recognition. An initial pre-approval process was used to filter student projects that might be too easy or too challenging, but projects on both ends of the spectrum did make it through. This presented something of a challenge from an assessment standpoint, as the chosen problems ranged dramatically in how well they allowed the student to demonstrate their knowledge of the material.

The content of this course was well suited to an online hybrid offering. In addition, there exists a wealth of high-quality publicly available and liberally licensed content online that an instructor can use to supplement and enhance their own material. By scheduling the weekly class meeting on Thursday evening, the students were given time to go through the online course content before coming to class, and this allowed the face-to-face sessions to serve more as lab than a lecture, which facilitated collaboration and dialogue.

Overall, the course describe here was well-received by students and a pleasure to teach, despite the somewhat unique challenges of covering this material in an era of rapid change. It is anticipated that the course will run again in the next academic year and regularly thereafter with only minor modifications to assignments and presentation.

References

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[5] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423, 2016.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Jeremiah Johnson. Neural style representations and the large-scale classification of artistic style. In *Proceedings of the 2017 Future Technologies Conference (FTC)*, pages 283–285, 11 2017.

[10] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[11] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, 2012.

[13] Zachary C. Lipton. The mythos of model interpretability. *Queue*, 16(3):30:31–30:57, June 2018.

[14] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[15] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[18] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[19] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[22] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[23] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[24] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Student Generation of an Optimal Decision Procedure Using Guess Who?*

Chris Alvin
Department of Computer Science
Furman University
Greenville, SC 29613
chris.alvin@furman.edu

Abstract

We consider the children's game *Guess Who?* as a medium to introduce students in an introductory computer science course to the idea of expected value and consequently a method to construct a model for optimal play. Our pedagogical purpose is to advocate for applied statistics and abstraction with algorithm development in a first computing course. We found that students in an introductory setting can generate an optimal abstract algorithm for playing *Guess Who?* corresponding to a path in an induced decision tree.

Introduction

As machine learning and the corresponding algorithms become pervasive in computer science, we continue to see a rise in the popularity of some of the simpler supervised learning techniques. For example, it is a reasonable 'capstone' project in a non-linear data structures course to require students to implement an algorithm such as ID3 [3] for decision tree induction. In this paper, we describe a sequence of classroom activities to (1) introduce students to expectation and expected value in a binary context, (2) abstract an algorithm from sample data and observations, and (3) induce the notion of a decision procedure comparable to a path in a decision tree.

Background: Guess Who? and Decision Tree Models

The game *Guess Who?* [1] is a question and answer game for children ages six and up. The game instructions give a brief summary.

> *Your object is to guess the mystery person on your opponent's card by asking one question per turn, and eliminating any faces that don't fit the mystery person's description. Guess your opponent's mystery person before your opponent guesses yours and you win!*

For clarity, we will refer to these mystery persons as characters and their descriptive characteristics using machine learning vernacular: features. In 'classic' *Guess Who?*, there are 24 named characters. Each character is presented to the players as a portrait with visually observable features. Sample characters can be reviewed at the Hasbro website[1].

Gameplay is turn-based in which one player refines his/her set of characters down to a single person by asking questions about the characters with Boolean responses. For simplicity, players are restricted to eliciting a single fact by posing simple questions consisting of unary expressions (i.e. questions without logical connectives). For example, it is acceptable to ask, "Does the mystery character have a beard?" In contrast, it is not acceptable to ask "Does the mystery character have a beard or is the mystery character wearing a hat?" Players then keep track of their progress in the game by updating the gameboard of character faces.

Hair Style
Has Partition
Wavy Hair
Bald
Long Hair

Hair Color
Red
White
Brown
Blond
Black

Eye Color
Blue
Red
Brown

Facial Hair
Has Facial Hair
Moustache
Beard

Facial Attribute
Large Nose
Rosy Cheeks

Other
Glasses
Earrings
Female
Wearing Hat

Figure 1: Guess Who? Character Features

Guess Who? Character Features. Each character in the game can be described as evidencing (or not) a particular facial feature or attribute. In our analyses we considered the features defined in Figure 1. A fair game of *Guess Who?* implicitly requires that the players agree upon a clear set of features. For example, if a player asks "Does the mystery character look sad?" , the

[1] https://shop.hasbro.com/en-us/product/guess-who-classic-game:7DEC61D9-5056-9047-F55F-FC686C17D23F

response is subjective. This imprecision is evident in some of the features listed in Figure 1. For example, analysis of character noses might result in an observer defining nose as a trinary feature: small, medium, large. This may lead to player confusion and disagreement.

Decision Procedures. At every step in gameplay, a player must choose a feature for which formulate a question. That is, which feature is best to use at each node in the tree? Since each feature results in a Boolean response for *Guess Who?*, we will employ a simple expected value computation which calculates the expected number of characters remaining after asking a question based on feature X:

$$E(X) = |X| \cdot P(X) + |\neg X| \cdot P(\neg X) \tag{1}$$

where X refers to the number of characters evidencing feature X and P(X) refers to the probability of feature X. For example, at the beginning of a game with all 24 characters under consideration, we note 4 characters have beards. Thus, $E(X = beard) = 4 * 4/24 + 20 * 20/24 = 17.3$ For inducing a decision procedure, we will thus greedily choose features based on the smallest expected value of all features of the remaining characters. We note mathematically that the smallest expected value arises from feature(s) where entropy is maximized; that is, when a feature is evident in 50% of the set of characters.

Sequentially computing all expected values would result in a binary decision tree model. A decision tree [2] is a model for approximating a discrete-valued function f where the root and each internal node corresponds to a test (decision) of a feature. Each descending branch from a node corresponds to a possible value of the feature. Each leaf of the tree then corresponds to a classification in *range(f)*. In the case of *Guess Who?*, our analyses will define an exact binary tree with 24 leaves, one leaf for each character. As is typical in algorithms such as ID3 [3] and C4.5 [4], we engage in a top-down, greedy procedure.

Project Goals

Our goal is to demonstrate that students in an introductory course in computing can develop an abstract, optimal strategy for playing *Guess Who?* That is, students can take advantage of their innate problem solving abilities and write an algorithm to play *Guess Who?* in the most efficient and effective way possible (in a general sense). In our gameplay and analyses, we assume that a player may only guess the name of the character when one character remains. A subgoal is that students in introduction to computing will identify an algorithm that corresponds to an induced binary decision tree.

Background: Target Course and Constituent Students

The target introductory computer science course is not a traditional introduction to programming course. The course is a general education, survey course for majors and non-majors in computing with the subtitle of "Games and Artificial Intelligence" (*Games and AI*). That is, students self-select the course topic over two other themed sections being offered in the same semester. An introduction to programming is included in all sections of the course, but emphasis is placed on breadth and in the case of *Games and AI*, the focus is on algorithms and algorithm development related to games, gaming, and some essential notions in artificial intelligence (logic, reasoning, probability, and decision making). The class consists of three 50-minute class meeting times and a dedicated 2-hour lab per week. As a department, the pedagogical philosophy revolves around student engagement. *Games and AI* has been designed to engage students in exploring and analyzing concepts beyond the lab time, often resulting in students participating in lab-related activities during the 50-minute class periods.

Class Year	Major	Prior Programming Experience
First Year (8) Sophomore (9) Junior (3) Senior (1)	Undecided (16) Urban Studies (1) Communications (1) Political Science (1) Biology (1) Music (1)	None (16) AP Computer Science A (2) Other Computer Science Experience (3)

Figure 2: Descriptors of Constituent Students in Games and AI

At the beginning of the course, a brief survey was administered to gather some background information on the students. As is shown in Figure 2, the 21 students in *Games and AI* are a diverse group in terms of age, background, and potential interest. Several students expressed in writing that they were taking the course to fulfill a general education requirement in Mathematical Reasoning. Several of the students whom are Undecided will eventually opt to major in Information Technology (IT) or Computer Science (CS). In fact, 2 of the 16 undecided students have since declared as CS majors with more intending to follow. Most students have never been exposed to programming; however, five students have already advertised some experience in programming.

Methodology

As an experiential, lab-based course, students are introduced to algorithm development and analysis from the outset. The course begins by considering the game of "Higher or Lower" where after each integer guess, the expert responds with "Lower" or "Higher" based on the guess with respect to the goal value. As a means of discussion, we consider a general algorithm for how to optimally play this game; students are unaware that they are implicitly developing a binary search technique. For the guessing game, it is emphasized that the player must implicitly track the current minimum and maximum values constituting an interval containing the goal value we call a *goal interval*. Students then engage in a sequence of lab exercises meant to deepen their analysis of the guessing game. Using several techniques, students infer that the width of the goal interval decreases toward a single value: the goal. More importantly, they infer that the most efficient method to guessing the goal value is using a binary search technique. This observation is formalized using a logarithmic analysis, both numerically and graphically. In summary, prior to engaging in an analysis of *Guess Who?*, students already understand the importance of searching a space by dividing that space in half and disregarding the irrelevant subspace. The 21 students in the course were paired randomly (2 groups of 9 and 1 group of 3) and together worked through a structured analysis of Guess Who. The structure is summarized as follows:

1. Each pair reads the game instructions. The lab clarifies that students should ask simple questions (questions without logical connectives). For the purposes of our analyses, we define the notion of casual and competitive game. A casual game requires students to eliminate all other characters before guessing the mystery person while a competitive game allows a player to guess the mystery person at any time.
2. Each pair plays six games with differing specifications: vary the first player as well as strategies by using questions asking about random features or competitive play using a perceived strategy. These games ask the students to implicitly analyze the game for optimal strategies.
3. For the features in Figure 1, students record feature values for each character in a feature matrix.
4. Students are then introduced to the idea of expected value using the example with beards described previously. Students then compute the expected value for each feature at the start of a new game.
5. Using their expected value computations, students choose an optimal feature for which to pose their first question.
6. As a means of gauging understanding, students are asked why 'beard' is not an ideal feature. And as a more meaningful follow-up, students are

asked to describe a scenario where beard is the best feature for which to base a question.
7. Last, students are asked to propose a rule for eliminating as many other characters as possible each turn.

Although not a requirement, each pair was asked to test their strategy against another pair. Students then submitted the lab for grading. The lab was not returned before the main component of this study: algorithm development. The algorithm development activity was broken into two phases: partner development and class development.

1. Phase 1 (20 minutes allotted). Working with a partner (possibly the same partner as before), develop an optimal strategy for playing *Guess Who?*. That is, write an algorithm to play *Guess Who?* in the most efficient and effective way possible (in a general sense). You may assume that your play may only guess the name of the character when one character remains.
2. Phase 2 (20 minutes allotted). Working with all participants in the room, construct and communicate an optimal algorithmic strategy for playing *Guess Who?*.

Results

In this section, we first report observations from students during game analysis and second report results of the algorithmic development activity. Due to absences, there were 18 participants resulting in 9 pairs for the algorithm development activity. *Feature Interpretation Issues.* The first task in the lab was for students to familiarize themselves with the game (prior to considering the features in Figure 1). Observationally, several groups engaged in intermittent discussions about interpretation of character features. For example, one pair disagreed about the age of a character (being young versus old). This pair determined that they needed to implicitly agree on a feature set as well the corresponding feature vectors for each character in order to play a non-contentious game. In anticipation of individual interpretations, student analysis focused on the feature set in Figure 1. Even with a common feature set, each pairs' feature matrix was unique among all student pairs. There was often disagreement about features with adjectives: long hair, wavy hair, large nose. We agree that some characters have ambiguous feature values, but these small issues do not have an impact on algorithm development. *Pair-Based Algorithm Development.* Observationally, student conversation focused on the feature set in Figure 1: this demonstrates a lack of abstraction of the problem space; that is, most student conversations implied that their algorithms would not apply to a

game of *Guess Who?* with 24 new characters and a distinct feature set (e.g. a Star Wars edition of the game). Discussion also focused on clarifications about expected value and why it is generally a good idea to cut the space in half as being optimal. We recorded five unrelated, innocuous conversations during the 20-minute timeframe.

Table 1: Analysis of Pair-Based Algorithms

Contain decision-making based explicitly on the defined feature set.	5
Contain no reference to the defined feature set.	3
Contain a reference to the feature set only as an example.	1
Total	9

Each of the nine pair-based algorithms generally espoused optimal play of *Guess Who?* by removing as close to half of the characters at each step. However, the concept of expected value was only used to communicate this split in two of the nine algorithms; the other seven referred to splitting in half. Table 1 summarizes the results of pair-based algorithm development with respect to abstraction. Even for those students who have taken AP Computer Science, it is clear that many students are more comfortable communicating a decision procedure by referring to concrete features. Of the three pairs who developed a more abstract algorithm, two of the pairs contained one student with prior programming experience. This means one pair without prior programming experience developed an abstract algorithm. Even more interesting is that three students having programming experience constructed a more concrete algorithm.

1) Calculate the expected value of each trait, then see which trait has the lowest value
2) Guess the trait with the lowest expected value
3) Eliminate characters based on your partner's answer
4) After every guess and character elimination, repeat expected value calculation until only one character remains on the board
5) Guess the name of the one remaining character

Figure 3: Unaltered Student-Generated Algorithm for Optimal Play of Guess Who?

Class-Wide Algorithm Development. Coming together into one group, initial suggestions tended toward specifics to the game (large nose). After some discussion, it was clear that those pairs who developed an abstract solution

were able to convince their peers that algorithms should be more general. After a few refinements, the final algorithm developed by the entire class is shown, verbatim, in Figure 3. This algorithm echoes some of the abstract, pair-based algorithms with greater refinement due to peer review. The algorithm in Figure 3 reflects the instructions given to the students: develop an optimal algorithm for playing *Guess Who?*. We observe that this algorithm describes a path in a decision tree. Thus, it provides a natural segue for introducing a tree data structures using a binary decision tree—a follow-up activity in the course.

Discussion and Consequent Activities

This was an activity in which students reported strong, positive sentiment. No survey was issued to confirm these results; however, reaction was positive. While we believe that developing an abstract algorithm for optimal gameplay is a meaningful computing activity, it serves as a quality activity to introduce trees, binary trees, and decision trees. For introductory computing students, trees offer a simple, implicit structure and a clear Boolean response at each node. Constructing a decision tree can be cumbersome, but it provides a foundation for deeper analyses. For example, students might consider the notion of a path in the tree with respect to developing a new, unique character. Also, since greedy decision tree construction does not guarantee a unique tree, students can compare and contrast their trees. One modification to the game is to allow a player to use logical connectives. Thus, a player can identify a character with a single question by constructing an expression that equates to a Horn clause [5] (and thus a path in a tree).

Pedagogical Benefits

While the overarching goal of the introductory computing course is to motivate and generate interest in computer science, we believe there are other pedagogical benefits. As data is consistently pervasive, students should be introduced to some of the tools that analysts, mathematicians, and computer scientists use to extract meaning. While expected value is a simple statistical notion, it can serve as a powerful mechanism for decision-making. We also believe this activity is initial evidence that algorithm development need not be relegated to upper-level courses. Students implicitly construct algorithms via source code development although few students pause to write an algorithm in pseudo-code or natural language unless it is required. This activity forces their hand to explicitly communicate an algorithm verbally and in writing while also allowing them to test it via gameplay. Further, students in introductory programming

courses are exposed to coding simple algorithms, more complex algorithms are not often developed without knowledge of the underlying data structures. As our activity does not require code, it can be viewed as a powerful motivational and synergistic tool for algorithms and data structures. Our analysis also provides a basis for discussion of other games. That is, a decision tree structure is not always appropriate to model many more complex games. In those cases, we (students and computer scientists) must work to develop other techniques and frameworks for such analyses.

Conclusions And Future Work

By introducing the notion of expected value in the context of *Guess Who?*, we have demonstrated that students can generate an optimal, abstract algorithm for playing the game. This provides a foundation for introducing the notion of a binary tree data structure and decision trees in an introductory computing course. We intend to engage students in these activities and collect corresponding data. On top of our content-based goals, we believe the use of *Guess Who?* to be an effective tool for eliciting interest in computer science and recruiting new majors while also providing a meaningful experience for non-majors.

References

[1] I. Hasbro. Guess Who? (Game). `https://shop.hasbro.com/en-us/product/guess-who-classic-game:7DEC61D9-5056-9047-F55F-FC686C17D23F`.

[2] T. Mitchell. *Machine Learning.* McGraw-Hill, 1997.

[3] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[4] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

Applying Social Media Analysis to Real World Business Problems: A Course Project*

Di (Richard) Shang
Department of Technology, Innovation and Computer Science
Long Island University
Brooklyn, NY 11201
Di.Shang@liu.edu

Abstract

This paper discusses the benefits of introducing a group term project to a graduate level computer science course of social media analytics. The course collaborated with a third-party platform –Riipen.com to connect student teams with external organizations. These projects gave students experience in developing applications to solve real world business problems by utilizing analytical techniques learned in the course.

1 Introduction

Social media analytics courses typically introduce to students the common techniques to obtain, process, analyze and visualize social media data [4]. Techniques covered in the courses include topic classification, sentiment analysis, network analysis, user/customer profile identification, etc. Meanwhile, in the real world, analytics is not just a technical exercise for the analytical professionals. To be successful, analytics must be also aligned with business strategies as analytics transforms decision making to a more data driven activity [2]. Therefore, while social media analytics courses focus on teaching practical techniques

and applications to computer science students, the ability to interpret the results and make actionable recommendations is also essential to students and thus should be considered in course designs [3, 1].

In an effort to improve students' soft skills - communication, teamwork, and strategic thinking, we implement a group term project in which students are required to serve as student consultants for external organizations through a platform called Riipen.com. The collaboration platform supports project-based experiential learning between students, industry partners, and educators. In addition to gain real-world work experience and build employable skills, students can network with employers and earn employment opportunities. To help guide the analysis, the external organizations provide a list of specific questions whose answers can be used to discover insights into specific business contexts. Students worked together on small teams to investigate the real-world business problems and conduct research. Through the semester, students work on related data sets and apply analytical techniques to seek potential answers to the given business problems.

The real-world business problems challenge students to decide on what analytical techniques to utilize, and students are motivated to experiment with a variety of tools and evaluate their capabilities. Using social media analytical techniques, students are able to conduct analyses and visualization, and they present the results on interactive online dashboards. Working with real business problems, students not only obtain an understanding of the capabilities of social media analysis to business, they also experience and learn the limitations associated with those techniques. The feedback from students were very positive. Students were, in general, more involved in the class and intellectual development, comparing to our previous course design. Students' course evaluations were higher in understanding and solving problems in related field and applying the course material to real word issues or other disciplines. In addition, students had better experience in developing an appreciation for the field in which this course resides and developing an ability to express themselves in writing or orally in this field.

In the following sections, we introduce the course methodology followed by a sample project, and then we present student evaluations and conclude the paper with our conclusions.

2 Course Methodology and Activities

This course is a survey of applications and techniques of social media analytics. Topics include social media and its impacts on business strategies, topic classification, sentiment analysis, network analysis and visualization. Students analyze real-world data using various applications and methods of text min-

ing and network analysis techniques. Besides gaining a good understanding of prominent social media analytics applications and methods, students who successfully complete this course will be able to understand how social media analytics are used to address business problems.

Throughout the semester various social media analysis techniques are introduced, and students apply their knowledge working on weekly assignments. These weekly assignments help reinforce basic analytical skills and prepared them for group term projects. The group term project is a major component of the course and accounts for 40% of the final grade. Groups of four to five students are established at the beginning of semester and required to develop group projects throughout the course. Each group apply methods and techniques learned in the course to analyze one social media data set. Students are required to conduct research on the topic, present topic, trend, sentiment, and network analysis, and discuss the business implications of their findings. At the end of semester, student groups submit written reports of their projects. Students are also required to give formal presentations to report their research, analysis and findings on an interactive web dashboard developed by each team. The report is required to consist of the following components:

1. Research on the specific topic (market, brand, product positioning, competitive landscape, consumer behavior, etc.)
2. Data collection and processing
3. Social media analysis (topic, sentiment, trend, customer profile, network analysis)
4. Visualizations of results and web application development
5. Interpretation of the results from social media analysis
6. Recommendations

In order to gain real-world work experience and build students' soft skills, in Spring 2016 we connected students with external organizations through a collaboration platform - Riipen.com. On the platform instructors post their course projects, and Riipen makes some edits and sources potential organizations who are interested in the projects. Figure 1 shows the screen shot of our term project on Riipen. The tasks of student teams were positioned as "a group of 3 to 5 student-consultants, specialized in data analytics, will analyze a social media data set in a business context that is specific to your organization. The student-consultants will conduct market research, as well as topic, trend, sentiment, and influencer analyses to provide insights to your organization" . Student teams were committed to the following tasks.

1. Analyze data relating to the industry to identify customer trends
2. Identify prospective customers or demographics via social media postings

3. Identify major influencers on social media who could become advocates for organization's brands

The final deliverable of the project to the organizations would be a 20-minute presentation, with an accompanying 10-page report, which outlines how consumer insights can be gained from social media in order to benefit the companies.

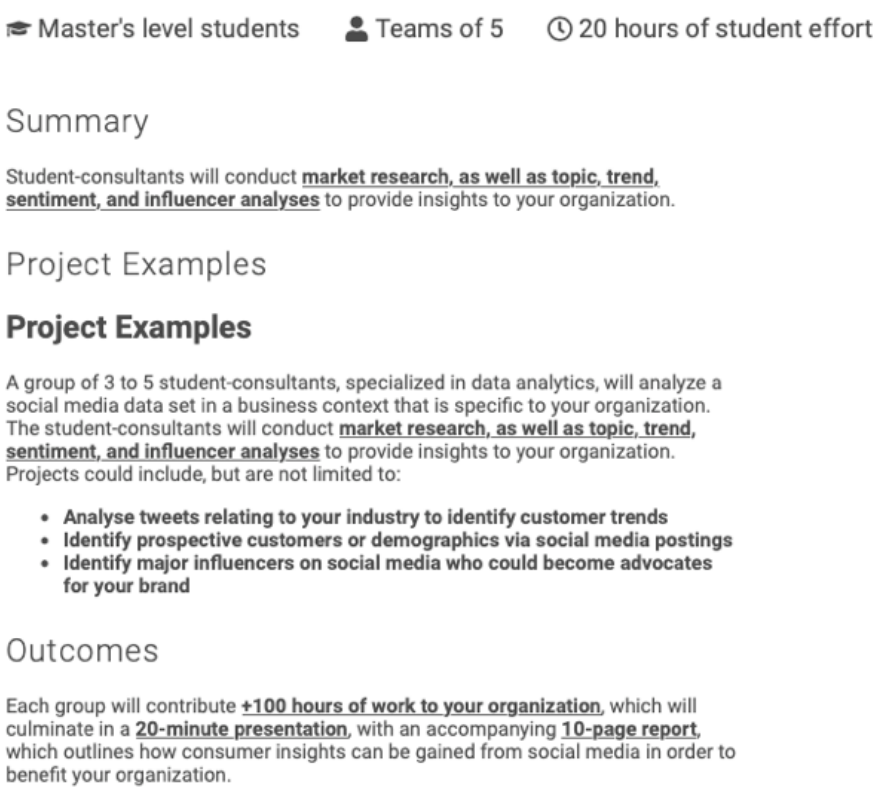


Figure 1: Screen Shot of the Project on Riipen.com

After the course project was posted on Riipen, organizations interested in the projects submitted their requests to the instructors with a brief background of the organization and a list of specific business problems they wanted to solve through social media analytics. The instructors could ask for clarity or necessary changes before their final approval of the requests. Organizations were required to provide necessary data for students to analyze if public data is not

available. They also committed to be available for a 1-hour interview (virtual, phone, or in person) in order to make sure our students fully understand the organization, industry and their specific needs. Through the semester, the organizations were responsive to periodic emails to answer any questions or concerns that students might have as they progressed. At the end of the semester, managers from the organizations were welcome to attend the final presentations (virtually if not local).

3 A Sample Project

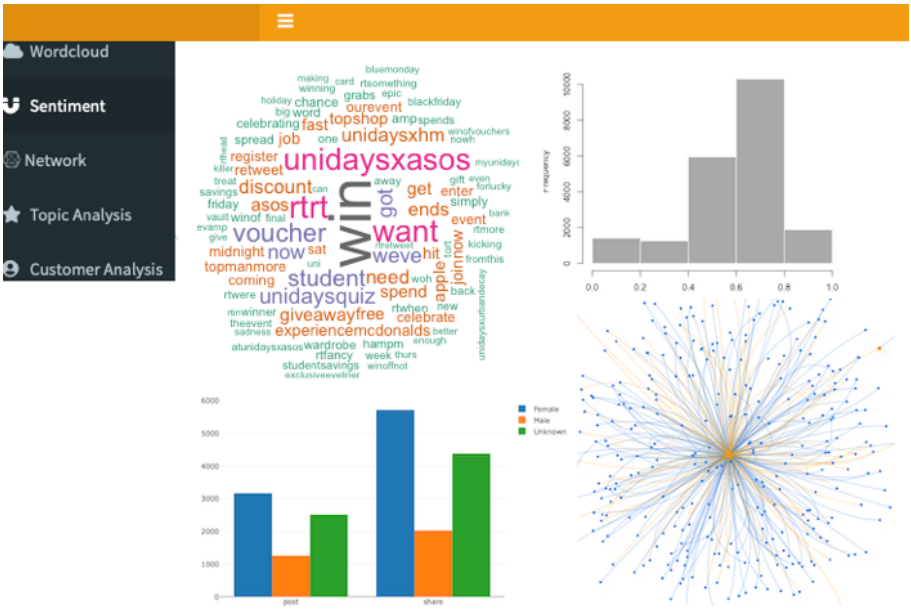


Figure 2: Screen Shot of the Shiny Web Application Developed for the Term Project

One of student teams was connected with a global company selling apparels to students and young professionals. The company has US social handles and promotes brand savings and exclusive member offers across social channels. The company stated that "our current social media presence consists of Instagram, Twitter, Facebook, ···, we'd like to develop a better growth strategy and triple our following and engagement ...A successful project would result in a higher following and higher engagement across all social media channels."

Through the project, the company wanted to get a better understanding of consumer mindsets for its brands and knowing what marketing messages and social media strategies would be to ultimately grow brand awareness and sales both in store as well as on web sites. The company suggested that "we hope to gain valuable insights into how to improve our social media presence and receive an actionable plan we can implement."

The company's questions were challenging, but they motivated students to research the business context and carefully plan the techniques they would apply to data analysis. In addition, to make actionable recommendations they had to examine the business implications of their findings and present only those can derive valuable insights. 20800 tweets related to the company were collected from Twitter API. Students experimented the analytical techniques learn in the course on the data. To get a better understanding of consumer mindsets for the company, students conducted topic classification and sentiment analysis. Network analysis were conducted to investigate the network structure and identify key opinion leaders. Students also generated customer profile of the company based on the demographic information and social media activities collected. At the end of the semester, student developed a Shiny web application with interactive dashboards to demonstrate their findings, as shown in Figure 2. Based on the findings, a number of actionable recommendations were concluded by the student team.

4 Course Evaluations

Students gave very positive feedback of their learning experience of the group term projects with Riipen. Table 1 shows student evaluations of the same course in Spring 2015 and Spring 2016 respectively. Students in Spring 2015 had the similar group term project, but they were not connected with external organizations and did not work on real business problems. Instead, they were provided social media data and worked on hypothetical business problems. In Spring 2016, we experimented with Riipen.com and connected students with external organizations. As shown in the table, having students working on real business problems resulted in higher evaluation scores on 1) involvement and intellectual development; 2) understanding and solving problems in the field; 3) applying the course material to real word issues or other disciplines; 4) developing an appreciation for the field in which the course resides; 5) developing an ability to express themselves in writing or orally in the field. Since students enrolled in the two semesters were from different groups, there were many uncontrolled factors that potentially affected the evaluation results. Therefore, we did not run any statistical testing to claim significant difference.

Table 1: Student Evaluation Scores (out of 7)

Evaluation Item	Median Score (Spring 2015)	Median Score (Spring 2016)
Your involvement in course	6.6	6.9
General intellectual development	6.5	6.8
Understanding and solving problems in this field	6.2	6.8
Applying the course material to real word issues or other disciplines	6.1	6.8
Developing an appreciation for the field in which this course resides	6.5	6.8
Developing an ability to express yourself in writing or orally in this field	6.3	6.8
The intellectual challenge presented	6.3	6.9
The amount of effort to succeed in this course	6.2	6.9

5 Conclusion

Group term projects are good course components to help students master analytical skills and apply knowledge to practical business problems. In addition, students improve their communication through writing reports and enhance teamwork skills by collaborating on the projects. In this paper we discuss the benefits of a course project design which connects student teams with external organizations to utilize analytical techniques to solve real world business problems.

While the project design did help student gain real-world work experience and build employable skills, we also recognize a number of limitations associated with the design. First, while most organizations have social media presences, many of them don't have quality social media data for students to analyze. As data is one of the most important inputs to the analysis process, instructors need to carefully screen the organizations and evaluate their suitability to the project. Second, students need to conduct in-depth research to understand the specific business contexts the organizations reside in, which can be time consuming and challenging for the time limit of one semester. As shown in the student evaluations, the amount of effort required to succeed in the course became higher and thus created extra stress to the students. Last, the business problems of some organizations are beyond the scope of the course. In certain cases, results from social media analysis may not provide valuable

insights into the business, and students are not able to deliver meaningful and actionable recommendations to the organizations to solve their specific business problems.

References

[1] Forbes. What are the top five skills data scientists need? https://www.forbes.com/sites/quora/2017/06/15/what-are-the-top-five-skills-data-scientists-need.

[2] R. Sharda, D. Delen, and E. Turban. *Business Intelligence, Analytics, and Data Science: A Managerial Perspective*. Pearson, 2017.

[3] C. Wilson. It's more than science: 5 soft skills needed to become a data scientist. http://blog.syncsort.com/2015/04/big-data/its-more-than-science-5-soft-skills-needed-to-become-a-data-scientist.

[4] D. Zeng, H. Chen, R. Lusch, and S.H. Li. Social media analytics and intelligence. *IEEE Intelligent Systems*, 25(6):13–16, 2010.

Demystifying Blockchain by Teaching it in Computer Science*

Adventures in Essence, Accidents, and Data Structures

Alan G. Labouseur, Matthew Johnson, Thomas Magnusson
School of Computer Science and Mathematics
Marist College
Poughkeepsie, NY 12601
{Alan.Labouseur,Matthew.Johnson,Thomas.Magnusson1}@Marist.edu

Abstract

This paper demystifies the advanced computer science topic of blockchain by placing it in the context of course and content development. In presenting suggestions for using blockchain as a tool to teach core computer science concepts, the authors reflect on student-centered, research-based projects spent understanding blockchain and developing an elementary implementation. Their experiences led to several teachable moments applicable to many topics across CS curricula including software design, algorithms and data structures, and distributed computing. The authors discuss many definitions of blockchain filtered through the philosophical lens of essence and accidents, give a precise definition of "essential" blockchain, and provide insight to understanding blockchain by presenting several of its structures and their implementation in the context of those curricular topics.

1 Introduction

As a buzzword, "blockchain" generates gratuitous hyperbole. Regardless of whether or not it really is "the next big thing", and in spite of the fact that it is generally mystifying and resists explanation, blockchain presents a useful gateway into teaching core concepts in computer science. In this paper,

we demystify blockchain by discussing its essence and accidents and making suggestions for using blockchain as a tool to teach core concepts in computer science.

1.1 Background

Having enjoyed (and endured) several student-centered, research-based experiences spent trying to understand blockchain and develop an elementary implementation of it, we encountered several teachable moments. We found that we could provide intuition to aid students' understanding by relating blockchain-specific teachable moments to various areas in traditional computer science curricula. These curricular areas include Software Design (in terms of defining the problem and discovering its essence and accidents), Algorithms and Data Structures (including hashing, linked lists, and trees), Distributed Computing (with issues like cooperating peers and peer discovery), and others (zero-knowledge protocols, graph theory, and more).

1.2 Contributions and Outline

Using the advanced topic of blockchain to teach core concepts in computer science is not well covered in existing literature. This paper – a step towards addressing that deficiency and demystifying blockchain in the process – makes the following contributions:

- In Section 2 we discuss some difficulties inherent in reasoning about blockchain and pose a critical question to motivate its exploration in academic computer science.
- In Section 3 we provide a precise definition of essential blockchain and suggest its use as a teaching tool.
- In Section 4 we dig into blockchain's details by exploring its fundamental nature through a discussion of the historic "essence and accidents" software design approach in this contemporary context. We also show examples of essential blockchain structures suitable for use in teaching several computer science topics.

Finally, Section 5 provides a brief conclusion, links to our blockchain source code, and comments on our future work.

2 Blockchain Difficulties

In reviewing the literature, we find blockchain difficult to define, difficult to characterize, and difficult to classify. It's frustrating. Yet it is within these

difficulties that we also find teaching opportunities.

2.1 Difficult to Define

Block-chain has too many definitions. It has been defined as: "a public ledger" [17], "a distributed ledger" [16], "a peer-to-peer distributed ledger technology" [10], "a ledger replication system", an "incorruptible digital ledger of economic transactions" [15], "a linked list that is built with hash pointers" [14], a "terrible [...] database" [7], a "state transition system with a consensus system" [5], and a "distributed database of records or public ledger of all transactions or digital events" [6].

How is one to make sense of all these definitions? How can our students dig through all that noise and arrive at meaningful insight? These are motivating questions.

2.2 Difficult to Characterize

Blockchain's characterizations are as varied at its definitions. Blockchain is "an undeniably ingenious invention" unable to be controlled by a single organization or fail at a single point [2]. This "disruptive technology... opens the door for developing a democratic open and scalable digital economy" as well as presents "tremendous opportunities". The Bitcoin blockchain "solved fundamental problems in a highly sophisticated, original, and practically viable way" [16].

What does all this mean? Can our students dig through these implications and still arrive at meaningful insight? These are also motivating questions.

2.3 Difficult to Classify

Just as its definitions are varied and its characterizations wide-ranging, there are many classifications of many kinds of blockchain. These include "cryptocurrency, private blockchains, permissioned ledgers, distributed tech, or decentralized tech" [1]. The common term "Distributed Ledger Technology" (DLT) further stirs the cauldron of bubbling blockchain brands. Corda, a "distributed ledger" for financial transactions is a DLT [4], but its whitepaper explicitly states that, "there is no block chain" instead calling the structures "notaries" [8]. The Hyperledger Sawtooth documentation, however, claims that an "enterprise distributed ledger" is the same as a blockchain [9].

Who are we to believe? Is there meaningful insight at all? These questions add still more metaphorical fuel to the equally metaphorical (and pedagogical) fire.

All of these difficulties led us to ask a motivating question: "What is blockchain?" But then we distilled it to a more important question:

What is *essential* blockchain?

3 Essential Blockchain

In the context of blockchain history from Bitcoin to Ethereum to Hyperledger, we build our definition of *essential blockchain* by first defining its structural parts.

Transaction – a container for arbitrary data.

Block – a container for one or more transactions.

Blockchain – an ordered, append-only container for one or more blocks where the i^{th} block b_i depends on the prior block b_{i-1} to confirm b_i's permanent stasis where $i \geq 1$. (We will revisit this idea in Section 4.2.1.)

Given the above definitions, and after examining many Bitcoin, Ethereum, and Hyperledger use cases, we find that blockchain is an overloaded term because it's more than a data structure, it's also a consensus network of peer instances of that data structure. Now we have its essence:

Essential Blockchain – a peer-to-peer network of Blockchain instances cooperating for consensus.

Let's have a look at some differences between blockchain as a data structure and blockchain as a consensus network.

3.1 Blockchain as a Data Structure

Blockchain is a data structure in that it is an *abstract structure* created to hold information. Satoshi Nakamoto [13], the anonymous inventor(s) of Bitcoin, the first instance of a blockchain, created blocks to contain monetary exchange transactions between entities. Blocks in a general blockchain are not limited to monetary exchanges. The Ethereum blockchain [5], in addition to holding payment transactions, also contains "smart contracts" that enforce various rules about various actions on the blockchain. The Hyperledger Fabric [10] blockchain stores many kinds of data. As one example, the Everledger project [11] maintains a Hyperledger blockchain for the Kimberly process of diamond mining, seeking to insure that none of those diamonds are mined unfairly.

The simplest blockchain data structure is the Transaction (Definition 3), an example of which is shown in the detail cutaway in Figure 1. A Block (Definition 3) is more complex, and is shown in the main body of Figure 1. Transactions and Blocks contain both essential and accidental attributes. (More on this in Section 4.1.2.)

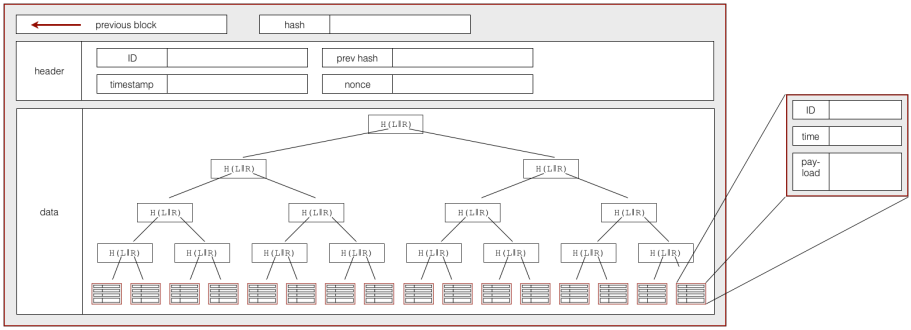


Figure 1: Block Data Structure with Transaction cutaway detail

3.2 Blockchain as a Consensus Network

Blockchain is a consensus network in that peers in a blockchain network cooperate to agree on which blocks of transactions are valid and which are not. To achieve this, Bitcoin uses a *proof of work*[1] mechanism [13] to ensure that Bitcoin peers who have performed more computational work have more influence on consensus decisions. Ethereum, likewise, adopted proof of work consensus, but aims to transition to *proof of stake*[2], in which those with more currency have more influence over consensus [5]. Hyperledger aims to have "modular, plug-and-play consensus" [10], meaning it is left to the developers of specific Hyperledger instances to choose the consensus method best for their needs.

Regardless of the consensus method, a blockchain cannot be called a blockchain unless it can also be called a peer-to-peer consensus network.

The duality of blockchain as both a data structure and a consensus network invites its use in many areas across computer science curricula. The next section presents a detailed look at a few of those areas.

[1] *Proof of work* systems are based on members of the consensus network performing computationally expensive calculations that are easy to verify to insure the validity of blocks. This is mining. From Nakamoto's Bitcoin paper[13]: "The proof-of-work involves scanning for a value that when hashed . . . begins with a number of zero bits. The average work required is exponential in the number of zero bits required and can be verified by executing a single hash."

[2] *Proof of stake* systems are based on certain members of the consensus network (called *validators*) pledging a portion of their assets (their stake) to vouch for the integrity of new blocks. The more they stake, the more blocks they validate, the more assets they have to lose if their blocks are found to be invalid.

4 Blockchain in Computer Science Curricula

Our faculty and student researchers developed a rudimentary blockchain implementation in Java[3], initially as an in-memory data structure, adding file-based persistence later, and eventually building cooperating peers. In doing so, we discovered that the duality of blockchain as both a data structure and a consensus network makes it a useful teaching tool in many computer science areas, including the study of software design.

4.1 Software Design

In his famous essay, "No Silver Bullet" [3], Frederick P. Brooks Jr. addresses some of the difficulties inherent in software development. He points out that software development bridges the chaotic world of "arbitrary complexity, forced without rhyme or reason by many human institutions and systems" with the abstract, yet precise domain software affords. Thus, complexity is software development's proverbial "middle name", and managing that complexity is a primary software development challenge.

Motivated by the desire to manage complexity while developing a custom blockchain implementation, we presented our students a fundamental question: What problem we are trying to solve?

4.1.1 Defining the Problem

Analyzing the landscape of commonly used blockchain "solutions" is daunting and difficult. As noted in Section 2.1, there are many definitions of blockchain. This makes generalization a challenge. The difficulty in characterizing and classifying blockchain systems only makes matters worse. Having found that blockchain is a fundamentally distributed system (via Definition 3 and Section 3.2), we assert that applications of blockchain to use cases that are not fundamentally distributed are wrong. In other words, if we're using blockchain, we should be trying to solve problems where multiple decentralized parties need concurrency, collaboration, cooperation, and fault tolerance. If that is not the case then we should not use blockchain and would be better served by an existing, well-established approach like relational, graph, or even NoSQL databases. We reached and support this conclusion through an exploration of blockchain's essence and accidents.

[3]The source code is freely available and we invite you to use it. See Section 5 for details.

4.1.2 Essence and Accidents

Brooks takes inspiration from the mother of all sciences, philosophy, to approach complexity. He defines two terms to discuss complexity:

accidents

essence Difficulties inherent in the nature of software.

accidents Difficulties that attend its production but are not inherent.

These ideas are not easy to self-appropriate. Instead of conjuring essence and accidents from the ground up, it is often easier to take something familiar and strip away accidents until, like a miner extracting precious metals from muddy rock, essence emerges.

By way of illustration, consider an instance of a coffee cup, in this case from Starbucks: it has a green logo showing the famous mermaid; it has a lid; it is made from a cardboard derivative; and it's filled (one hopes) with wonderfully-fragrant, rich coffee. But our Starbucks cup need not have its logo, nor a lid; those features are *accidents* of the cup. It need not be made from cardboard, nor does it need to be filled with coffee, as those features are also *accidents*. Removing those accidental features (or properties), we are left with a cylindrical object with the capacity to contain and ability to pour liquids. This is the *essence* of a cup... its "cupness".

We take the same approach with blockchain. By embracing the difficulties in defining, characterizing, and classifying blockchain over its short history from Bitcoin to Ethereum to Hyperledger, we can identify its properties and categorize each of them as *essential* or *accidental* in nature. Table 1 shows the results of this process when applied to Transaction and Block structures in the context of Definitions 3, 3, and 3.

We note an interesting meta-question about the Block structure: Is the use of a Merkle tree (discussed in the next section) an accident or is it essential to a block? It's certainly a solution to an essential problem of blockchain, but it is not the only solution. So is it essential or accidental? (That sounds like a good essay question for our students.)

4.2 Algorithms and Data Structures

Before we can implement blockchain, we must first become familiar with its structures and some algorithms that work with them. Understanding these structures and algorithms, along with their common implementations, helps build fundamental computer science skills necessary to comprehend emerging technologies such as blockchain. We begin with hashing.

Structure	Property	Nature
transaction	id	accidental
	timestamp	essential-ishα
	payload	essential
block	id	accidental
	hash	essential
	timestamp	accidental-ishα
	previous block's hash	essential
	nonceβ	accidental
	Merkle tree	essential?

α The essential or accidental nature of timestamps is unclear. On the one hand, a transaction or a block clearly comes into existence at some particular point in time. On the other hand, it may not be essential to record that.

β A nonce is a value that, when combined with the previous block's hash, will result in a new hash that satisfies the proof of work.

Table 1: Blockchain Essence and Accidents

4.2.1 Hashing

Primarily a mathematical concept, hashing requires a hash function, H, that maps many possible inputs to a smaller number of outputs. There are many kinds of hash functions. For blockchain, we are particularly interested in *cryptographic* hash functions. A *cryptographic hash function* is one that fulfills these properties [14]:

Collision-resistant It is infeasible to find two values, x and y, such that $x \neq y$, yet $H(x) = H(y)$.

Hiding If a secret value r is randomly chosen from a set of values with equal probability, then given $H(r||x)$ it is infeasible to find x. ($||$ represents concatenation.)

The primary purpose of a cryptographic hash function in blockchain is to create a "fingerprint" of a given piece of data without revealing any information about it. This "fingerprint" output from a hash function is commonly called a *digest* or a *hash*.

Most (if not all) blockchains use hashing to achieve the "chain" part of their nature with *hash pointers*, which we describe in the next section. Hashing also provides an efficient method for "confirming permanent stasis" (as noted in

Definition 3) because comparing hash values is a fast, easy, and anonymous[4] way to detect whether or not content has been altered since its initial hash.

4.2.2 Linked Lists

A useful and concrete definition of blockchain as a data structure states that, "A block chain is a linked list that is built with hash pointers." [14]. A *hash pointer* is "a pointer to where data is stored together with a cryptographic hash of the value of that data at some fixed point in time" [14].

Since linked lists are a fundamental topic in virtually all introductory or intermediate computer science courses, enhancing them with hash pointers to support blockchain applications makes for a nice modernization. Combining a cryptographic primitive such as a hash with a basic data structure like a linked list enables students to learn computer science fundamentals and emerging technologies simultaneously. Such teachable moments build understanding of and excitement for computer science. The excitement continues (one hopes) into discussions of $O(n)$ operations like list (or blockchain) traversals and linear searches. Once we've found a target block, we'll need to do a tree traversal to find transactions.

4.2.3 Trees

Instead of storing all transactions in a simple list within a block, Satoshi Nakamoto used a version of a balanced binary tree called a Merkle tree [12, 13]. A Merkle tree is:

binary Nodes have at most two children.

ordered Nodes maintain some ordering appropriate for the data they carry, e.g., lexicographic or numeric.

balanced For every node, the heights of its subtrees differ by at most 1.

leftmost All nodes are placed as far left as possible.

cryptographically hashed The root and all internal nodes contain the hash of the content of their children.

This structure allows all transactions within a block to be validated by a single hash stored at the root of the Merkle tree and called, appropriately enough, the *Merkle root*. The node at #1 in Figure 2 is its Merkle root. Note that it contains a hash of the content of its (left and right) children. They

[4] Anonymous in the "zero knowledge" sense that we can confirm whether or not some value has changed without knowing the actual value itself.

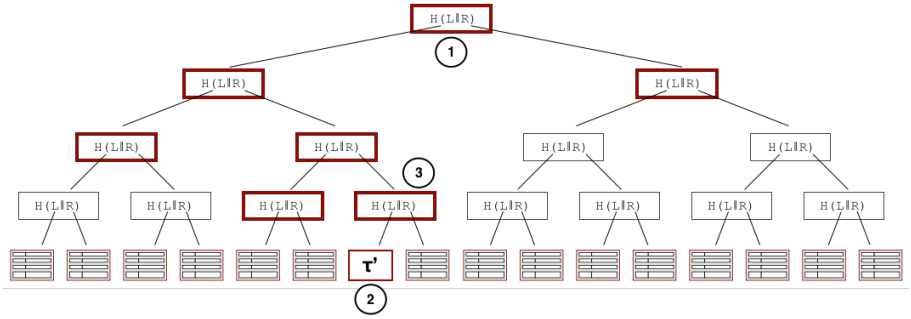


Figure 2: Hash Validation of Transactions in a Merkle Tree

in turn contain a hash of the content of their children, and so on all the way down to the penultimate nodes, which contain a hash of the content of (their children) the leaf nodes, which are the actual transactions. (See Figure 1 for a closer look at a Transaction.) In this manner, if any transaction τ in a block (i.e., any leaf node in a Merkle tree) is altered (e.g., #2 in Figure 2), that node's parent (e.g., #3 in Figure 2) will change as well. This change will propagate up the tree, level by level, until it reaches the root. The root therefore reflects all changes in the tree so it is the only datum a block needs to validate to confirm the permanent stasis of the transactions contained therein.

We constructed a Java class called `MerkleTree` that satisfies the above properties and accepts any data and any hashing algorithm. This exercise might be appropriate in a classroom setting as a bridge from simple binary trees to more complex data structures. It may also be a good way to teach the importance of choosing an appropriate abstract structure for the task at hand and determining which properties it must satisfy **before** implementing it in code, as bugs are easier to fix while planning than while programming.

Constructing a Merkle tree from an arbitrarily-sized collection of transactions is fairly interesting. The details of such an algorithm are out of the scope of this paper, but would be very appropriate in an upper-level algorithms course. As Niklaus Wirth said, algorithms plus data structures equals programs. Tying algorithms to data structures in this manner reinforces the link between the two and their importance in programming fundamentals.

4.3 Distributed Computing

The essentially distributed nature of blockchain (Definition 3) serves as a natural example by which to teach topics of networking, routing, and consensus algorithms. Our preliminary work used multiple blockchains on one computer

running many JVM instances. This makes for a nice simulation of a distributed system, as each socket requires an IP address and a port. It's easy to see how, for example, 127.0.0.1:2007 exchanging messages with 127.0.0.1:2112 is similar enough to 172.217.10.78:42 exchanging messages with 137.254.120.50:71 as not to matter. Once we had that working, we moved with ease to multiple blockchains on multiple cooperating peers (both physical and virtual).

4.3.1 Cooperating Peers

We devised a rudimentary REpresentational State Transfer (REST) Application Programming Interface (API) over Hypertext Transfer Protocol (HTTP) with a handful of endpoints that cooperating peers use to communicate. These REST endpoints allow peers to request (1) the entire blockchain, (2) a specific block in the blockchain, or (3) a specific transaction in the blockchain by using HTTP verbs with the following paths:

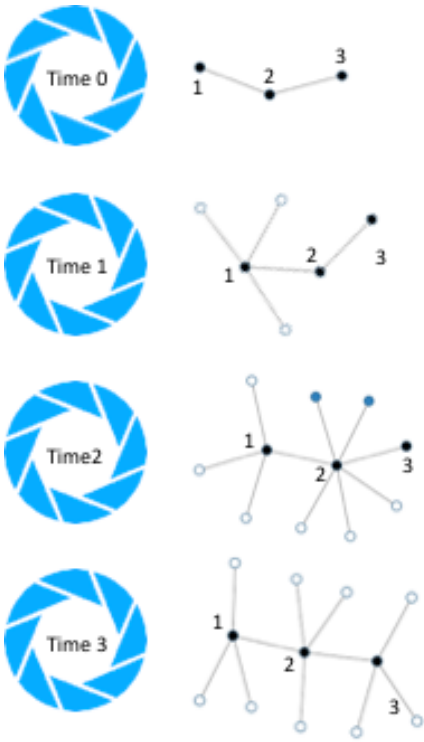
1. `GET /blockchain`
2. `GET /blockchain/{block id}`
3. `GET /blockchain/transaction/{transaction id}`

Peers can POST new transactions with the intention that they be accepted as soon as possible. But POSTed transactions do not immediately become part of the blockchain. The peer looking to accept transactions into the blockchain puts them into its *transaction buffer*, a container for pending transactions. A peer mines its buffer when another requests it via the `POST /mine` endpoint. This triggers the *proof of work* mining process. When complete, the previously buffered transactions officially constitute the next block of the blockchain.

4.3.2 Peer Discovery

Peer discovery is accessed through the REST API endpoint `GET /friends`. It provides the IP addresses of known peers on the blockchain network. (A friend is a peer with whom another peer has communication over the network through the REST API.) When starting a peer for the first time, the user must list the IP addresses participating in the blockchain network. This constitutes the initial friend list for the newly-started peer. The network effect of requesting the friends list of its friends allows peers to discover each other.

While discovering its peers, each node builds a graph of the blockchain network by denoting itself as a *from* vertex and its friends as one-hop *to* vertices. This process is repeated for all of the *to* vertices, but this time with them as the *from* vertex. The result of the whole process is a snapshot of the entire



Vertices with the largest change in degree over consecutive graph snapshot pairs:

Pair	vertexID	change
0->1	1	+3
1->2	2	+5
2->3	3	+3
0->1	2	0
0->1	3	0
1->2	1	0
1->2	3	0
.		
.		
.		
2->3	2	-2

Figure 3: Blockchain network evolution captured through graph snapshots with output from a query about changing influence

network as a graph. The network will change over time (they all do), providing a opportunity to visualize and analyze blockchain evolution with a graph analytics, an example of which is shown in Figure 3.

There is more work to be done, including an implementation of consensus algorithms as required by Definition 3 and supporting smart contracts. These are first up in our future work.

5 Conclusions and Future Work

In studying new technologies like blockchain and creating their own implementation of it, students engage in a holistic learning process bridging theory and practice that promises to be both challenging and rewarding, not to men-

tion demystifying in terms of the new technology itself. We encourage institutions to promote this kind of learning and to teach core concepts through emerging technologies like blockchain. To that end, the source code for our blockchain implementation is available on GitHub at `https://github.com/Marist-Innovation-Lab/blockchain`. All are welcome to take it, use it, and build on it.

We are pursuing two paths in our future work: implementation and research. On the implementation path, we will be programming consensus algorithms and adding support for smart contracts. On the research path, having achieved a network of cooperating blockchain peers and the ability to capture snapshots of their evolution over time, we will be simulating common network attacks (man in the middle, denial of service) and uncommon attacks (Sybil subversion in peer-to-peer networks) so we can test theories of blockchain's security characteristics. Both promise to be fun.

Acknowledgements

The authors wish to thank Daniel Gisolfi and the other students (and faculty and staff) of the Marist/IBM Joint Study and the NSF-funded *SecureCloud* research grant (#1541384). We appreciate their technical and emotional support. Thanks go out as well to the reviewers for their thoughtful comments. Writing papers like this, just as with writing software, relies on critical feedback from external stakeholders.

References

[1] Berkeley Blockchain. What is blockchain?, 10 2016. `https://drive.google.com/file/d/0ByBe1QJVC-EJRUJqVWcyY2VNdlU/view`. Accessed on 2018-11-18.

[2] BlockGeeks. What is blockchain technology? a step-by-step guide for beginners, 2017. `https://blockgeeks.com/guides/what-is-blockchain-technology/`. Accessed on 2018-11-18.

[3] Frederick P. Brooks, Jr. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, April 1987.

[4] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: An introduction, 2017. `https://docs.corda.net/_static/corda-introductory-whitepaper.pdf`. Accessed on 2018-11-18.

[5] Vitalik Buterin et al. Ethereum white paper: A next-generation smart contract and decentralized application platform, 2013. `https://github.com/ethereum/wiki/wiki/White-Paper`. Accessed on 2018-11-18.

[6] Michael Crosby, Pradan Pattanayak, Sanjeev Verma, and Vignesh Kalyanaraman. Blockchain technology: Beyond bitcoin. *Sutardja Center for Entrepreneurship & Technology*, 2:6–10, 2016. `http://scet.berkeley.edu/wp-content/uploads/BlockchainPaper.pdf`. Accessed on 2018-11-18.

[7] Trent McConaghy et al. Bigchaindb 2.0: The blockchain database, 2016. `https://www.bigchaindb.com/whitepaper/bigchaindb-whitepaper.pdf`. Accessed 2017-8-21.

[8] Mike Hearn. Corda: A distributed ledger, 2016. `https://docs.corda.net/_static/corda-technical-whitepaper.pdf`. Accessed on 2018-11-18.

[9] Hyperledger. Sawtooth introduction, 2016. `https://intelledger.github.io/introduction.html`. Accessed 2018-11-18.

[10] Hyperledger. Hyperledger whitepaper. Google Drive, 2017. No specific authors or publish date were given. `https://docs.google.com/document/d/1Z4M_qwILLRehPbVRUsJ3OF8Iir-gqS-ZYe7W-LE9gnE/`. Accessed on 2018-11-18.

[11] Everledger Ltd. Everledger industry applications, 2016. `https://www.everledger.io/industry-applications`. Accessed on 2018-11-18.

[12] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford University, Stanford, CA, USA, 1979. AAI8001972. `http://www.merkle.com/papers/Thesis1979.pdf`. Accessed on 2018-11-18.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `bitcoin.org/bitcoin.pdf`. Accessed on 2018-11-18.

[14] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction.* Princeton University Press, Princeton, NJ, USA, 2016.

[15] Don Tapscott and Alex Tapscott. *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World.* Brilliance Audio, Brilliance Audio, 2016.

[16] F Tschorsch and B Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials*, 18(3):2084–2123, 2016. `https://eprint.iacr.org/2015/464.pdf`. Accessed on 2018-11-18.

[17] Zibin Zheng, Shaoan Xie, Hongning Dai, Xiangping Chen, and Huaimin Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *Big Data (BigData Congress), 2017 IEEE International Congress on*, pages 557–564, ., 2017. IEEE.

Puzzling Through Discrete Mathematics*

Edmund A. Lamagna
Department of Computer Science and Statistics
University of Rhode Island
Kingston, RI 02881
eal@cs.uri.edu

Abstract

An applied, yet mathematically rigorous, course on combinatorial problem solving is discussed. Each class begins with a set of puzzles that introduce and begin to stimulate thinking about the topic for the day. Students work on the puzzles in small groups for about one-third of the class period. When puzzles were first introduced, it was thought that less material could be covered but that this would be outweighed by an increase in student interest and participation, and that the course would be more fun. Unexpectedly, all of the original material is still covered since students are now better prepared and motivated for the more traditional presentation that follows "puzzle time." The key to this approach is selecting relevant, intriguing puzzles for each topic.

1 Introduction

The author teaches an applied, yet mathematically rigorous, junior level course on combinatorial problem solving. Algorithmic thinking is emphasized throughout, and the course provides a solid foundation for a follow-on course on the design and analysis of algorithms. Major topics include sets, logic, probability, proofs by induction and contradiction, the pigeonhole principle, arrangements, selections, distributions, binomial identities, inclusion-exclusion, recurrence relations and recursion, and graphs and trees.

Several years ago, the author decided to integrate active learning into the course. Each class now begins with a set of puzzles (typically four) that introduce and begin to stimulate thinking about the topic for the day. Students work on the puzzles in small groups of 2-4 individuals for about one-third of the class period. "Lone wolves" are allowed work by themselves as long as they spend the time on the puzzles. When puzzles were introduced, it was thought that less material could be covered but that this would be outweighed by an increase in student interest and participation, and that the course would be more fun. Unexpectedly, all the original material can still be covered since students are now better prepared and motivated for the more traditional presentation of the new material that follows "puzzle time."

The course carries four credits and meets twice a week for 1.75 hours. The format of a typical class is: puzzles (30 minutes), homework solutions (15 minutes), new material (60 minutes). The topics in the course are highly cumulative, so it is important for students to keep up with the material. Homework problems are assigned every class period, with about one-third considered easy, one-third of medium difficulty, and one-third challenging. The assignments are collected at the start of the next class and are graded. The instructor goes over several homework problems to illustrate key points, and also takes requests from students for problems they would like to see. The text, Alan Tucker's *Applied Combinatorics* [4], was chosen for its rich selection of problems and examples, and the fact that the sections of the book are short enough to encourage students to read them. Typically, one section of the book is covered per class period.

The key to the author's approach is selecting relevant, intriguing puzzles for each topic. A sample of problems that have been successfully utilized are presented in the remainder of this paper. These cover a variety of topics throughout the semester. The presentation here includes a discussion of the important pedagogical ideas behind the puzzles and their selection.

2 Mathematical Preliminaries

The first unit of the course provides a broad overview of set theory and mathematical logic, probability, proofs by induction and contradiction, and the pigeonhole principle. Most computer science students have prior exposure to set theory, probability, induction, and (to some extent) logic. The unit attempts to "level the playing field" as much as possible for those who have not seen one or more of these topics, and also provides a brief review of fundamental mathematical concepts that are used throughout the course.

Our opening puzzle is an enticing one that is presented in conjunction with mathematical logic on the first day of class. At first glance, it seems impossible

to solve. Raymond Smullyan popularized problems of this nature, setting them on an island all of whose inhabitants are either knights (Truth Tellers, who always answer questions correctly) or knaves (Liars, who always give incorrect answers to questions) [3]. It is important to set the tone for the semester at the first class meeting and this puzzle admirably fits the bill.

Example 1. A Fork in the Road. You've been out for a long walk on the island. Unfortunately you're alone, hopelessly lost, very tired, and it's getting dark. You come to a fork in the road. Fortunately you know from the signpost lying on the ground that one fork leads back home. Unfortunately the other fork leads to a pit of venomous snakes. Fortunately there's a native who will answer one yes/no question for you. Unfortunately you don't know whether the native is a Truth Teller or a Liar. What question should you ask to determine with certainty the way home?

Solution. This puzzle has two unknowns: whether the native is telling the truth and which road leads home. The four possibilities are depicted in the truth table shown. We're seeking a question where a "yes" answer tells us that we should take, say, the left fork, and a "no" answer tells us to take the right fork. Observe that when formulating the question, "yes" means "yes" and "no" means "no" if the native is a Truth Teller. On the other hand, if the native is a Liar, a question whose correct answer is "yes" will elicit a reply of "no," and vice versa. So we want the correct answer to the question to be as shown in the third column of the truth table.

Is native TruthTeller?	Left fork leads home?	Correct answer
yes	yes	yes
yes	no	no
no	yes	no
no	no	yes

Solutions to this problem in the literature are generally given in terms of a conditional, or "if" operation. The author's goal is for students to be able to form queries from an arbitrary truth table in a systematic way using *disjunctive normal form* (DNF). A DNF question that will get us home safely is, "Are you telling the truth and does the left fork lead home or are you lying and does the right fork lead home?" □

Example 2. Not So Magic Squares. Can you fill a 6×6 matrix with 1s, 0s and -1s so that the rows, columns, and two main diagonals all have different sums? Find a matrix or prove none exists. Generalize to an $n \times n$ matrix.

Solution. The solution to this puzzle involves a simple yet non-trivial application of the Pigeonhole Principle. The greatest sum of six numbers that can be formed from 1s, 0s, and -1s is 6; the smallest is -6. So there are 13 possible sums that can appear in a row, column or diagonal. But there are 6 rows, 6 columns and 2 main diagonals — 14 sums in all. So at least two sums must be

the same! For the $n \times n$ case, $2n+1$ sums can be formed and $n+n+2 = 2n+2$ sums appear in the matrix so, again, at least one must be repeated. □

Students love magic tricks — especially ones that they can learn and show to impress family and friends. The following card trick is based on the principle of mathematical induction.

Example 3. Perfect Shuffle. The instructor performs a magic trick where he cuts a standard deck of playing cards and has a student merge the two halves together with a single riffle shuffle. After shuffling, the instructor takes pairs of cards from the top of the deck. Amazingly, every pair consists of one red and one black card! How were the cards prearranged to perform this astounding feat? Why does the trick work?

Solution. The "set up" is to prearrange the deck so that black and red cards alternate. When the instructor cuts the deck, it is done so the bottom cards of each half are of opposite color. It is easy to show by induction that, for every pair of cards that drop during the shuffle, one is black and one is red. Consider the three cases: (1) both cards drop from the left pack, (2) both cards drop from the right, and (3) one card drops from each side. Furthermore, after each pair of cards drop, the packs still have the invariant properties that the bottom cards are of opposite color and the cards in each pack alternate in color. □

This is known as a "Gilbreath shuffle" and has many interesting variations [2]. It is left as an exercise to figure out how to perform the trick so every four cards include one of each suit, or every thirteen cards include one of each rank. An even more impressive variation is to shuffle together two decks (one with red backs, the other blue), count off 52 cards, and discover that both the first and last 52 cards (with scrambled colored backs) are perfect decks with one each of the 52 cards in a standard deck!

3 Enumeration

Mastermind is a two-player game of deductive reasoning and logical thinking. One of the players, called the *codebreaker*, tries to guess a secret code selected by the other player, called the *codemaker*. In the standard version of the game, the code consists of a row of 4 colored pegs selected with possible repetition from 6 colors, so $6^4 = 1296$ codes are possible.

The codebreaker cracks the code through a series of guesses, each consisting of a row of four colored pegs. The codemaker scores each guess with *key pegs* revealing how close the guess is to the actual secret code. A black key peg is scored for each peg in the guess that is the correct color *and* in the correct position. A white key peg is scored for each peg that is the correct color but in the wrong position.

The game of Mastermind poses interesting logical thinking challenges and raises a number of combinatorial counting problems. In class, the students work to solve a series of progressively more difficult Mastermind puzzles, but the author's favorite problem is to have the students analyze a simplified version of the game. This exercise begins to stimulate the kind of thinking developed in the course unit on enumeration.

Example 4. Simplified Mastermind. A simplified version of Mastermind has just three positions and pegs of only three colors. There are three possible opening strategies:

- playing three pegs of one color (e.g., <red, red, red> or 111),
- playing two of one color and one of another (e.g., <red, red, blue> or 112),
- playing three different colors (e.g., <red, blue, green> or 123).

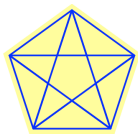
	1 1 1	1 1 2	1 2 3
—	8	1	0
O	0	4	0
●	12	6	3
O O	0	3	6
● O	0	4	6
● ●	6	6	6
O O O	0	0	2
● O O	0	2	3
● ● ●	1	1	1
WC left	12	6	6
Avg left	9.07	4.41	4.85

Fill in the table to show the number of secret codes remaining for every possible reply the codemaker can give to each opening. Which strategy is the best in the worst case? Which is best on average?

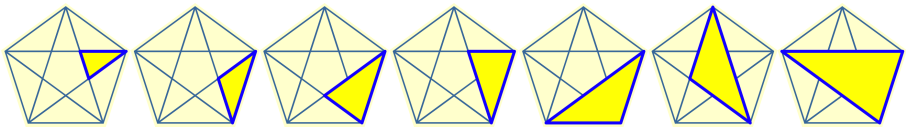
Solution. The second and third strategies can leave the codebreaker with 6 codes in the worst case. The second strategy is best with regard to the average number of codes left. Students are always interested when the instructor presents a similar analysis of openings for the standard game of Mastermind. □

The addition and multiplication principles lie at the heart of the unit on enumeration. In order to be able to apply these principles successfully, a student needs to break a problem into "disjoint cases" that cover all possibilities. The next puzzle is an excellent exercise in this kind of reasoning.

Example 5. Counting Triangles. How many triangles are there in the pentagon shown?



Solution. There are seven types of triangles, as illustrated below.



Each group consists of 5 triangles, one for each rotation of the

pentagon by 72 degrees. So overall, there are $7 \cdot 5 = 35$ triangles.□

Example 6. Rubik's Cube. How many ways can a Rubik's cube be scrambled?

Solution. There are 8 corner cubes and each can be placed in any of the 8 corners. A corner cube can have one of 3 orientations. Similarly there are 12 edge cubes and each can be placed in one of the 12 edge cubicles. An edge cube has one of 2 orientations. So a good student answer, and one the instructor would be very happy to see, is $(8!\,38)(12!\,2^{12})$. To get closer requires special knowledge of the mechanics of the cube.

The actual number of permutations is $(8!\,38)(12!\,2^{12})/(3 \cdot 2 \cdot 2)$. While a corner cube can have one of three orientations, once 7 of the corners are oriented, the orientation of the last is determined. If you remove the stickers from one corner and paste them back incorrectly, the cube cannot be solved! As with the corner cubes, once 11 of the edges are oriented, the orientation of the last is fixed. The other factor of 2 in the denominator is due to the fact that only "even permutations" can be achieved since each twist of the cube moves an even number of corners and edges. □

4 Recurrence Relations

The unit on recurrence relations and recursion is an important and useful one for computer science students. Most of them will take a required course on the design and analysis of algorithms in the following semester. An important skill they will need is the ability to set up and solve recurrences. A variety of divide-and-conquer and reduce-and-conquer recursive algorithms are presented and analyzed in this unit. Students learn how to solve divide-and-conquer recurrences via back-substitution (telescoping), and both homogeneous and inhomogeneous linear recurrences with constant coefficients. From the author's experience, discrete mathematics courses taught by computer science departments differ from those taught by mathematics departments in their greater emphasis on recursion and recurrence relations.

The next puzzle is the first in the unit on recurrences. It is a good starting point since, throughout their academic careers, students have encountered similar problems asking for the next number in a sequence. Recurrence relations provide a formal model for solving such problems. With an appropriate recurrence, students can generate not only the next number in the sequence but all subsequent terms as well. Moreover, having a recurrence may lead to a general formula for the numbers in the sequence. In class, we use the convention that sequences start with a 0-th element: $a_0, a_1, a_2, a_3, \ldots$

Example 7. On the Underground. A few years ago, an advertisement on the London Underground (subway) system read:

If you can determine the next number in each of the following lists before you arrive at your stop, come in and we'll offer you a job!

For each of the sequences shown, give both a plausible next entry and a recurrence relation to calculate the n-th term an from previous terms and (possibly) a function of a_n.

a) 1, 2, 4, 8, 16, 32, ___
b) 0, 1, 3, 6, 10, 15, ___
c) 1, 1, 2, 3, 5, 8, ___
d) 1, 2, 5, 12, 29, 70, ___
e) 1, 2, 3, 6, 11, 20, ___
f) 0, 1, 3, 8, 21, 55, ___

Solution. a) Students can immediately see that the next number in the sequence is 64 and computer science students, in particular, see that each term is a power of two. But how can we express this as a recurrence? Since each term doubles the previous one, $a_n = 2a_{n-1}$.

b) Here students encounter a recurrence with an inhomogeneous part. a_1 is one more than a_0, a_2 is two more than a_1, a_3 is three more than a_2, etc. So $a_n = a_{n-1} + n$. The sequence gives what are known as the "triangular numbers" since each term can be depicted as an equilateral triangle of dots.

c) These are the famous Fibonacci numbers (shifted so the 0-th term is 1 instead of 0). Each number in the sequence is the sum of the previous two, so $a_n = a_{n-1} + a_{n-2}$.

d) Each term in this sequence is twice the previous term plus the term two back: $a_n = 2a_{n-1} + a_{n-2}$.

e) These numbers form a so-called "tribonacci" sequence. Each term is the sum of the previous three, $a_n = a_{n-1} + a_{n-2} + a_{n-3}$.

f) These are the terms of even index among the standard Fibonacci numbers: $F_0, F_2, F_4, ...$ The sequence can be generated by the recurrence $a_n = 3a_{n-1} - a_{n-2}$. □

The Tower of Hanoi is a popular puzzle familiar to most computer science students. It consists of a board with three poles and n disks of different sizes. The disks are initially placed on the left pole in order of size, with the smallest at the top and the largest on the bottom. The object is to move all n disks from the pole on the left to the pole on the right using the middle pole for intermediate storage. The disks must be moved one at a time, and a bigger disk can never be placed on a smaller disk.

In class, the students are given a copy of the game and are asked to develop an algorithm for the standard puzzle, as well as a recurrence for the number of moves. Since the standard version of the puzzle is well known, we focus attention on a variant that students are also asked to solve. The same board can be used as a manipulative to assist in solving both versions.

Example 8. Restricted Tower of Hanoi. This version is similar to the

standard one except that any move must either place a disk on, or move a disk from, the middle pole. Develop a procedure to solve this version of the Tower of Hanoi. Analyze your algorithm to obtain a recurrence relation, including initial conditions, for the number of moves, then solve.

Solution. The following recursive procedure solves the puzzle by moving n disks from pole X to pole Y through the middle pole:

If $n = 1$, move disk 1 from X to the middle pole, then from there to Y.

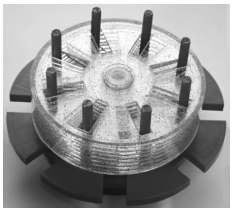
Otherwise,

1. Move $n-1$ disks from X to Y through the middle pole.
2. Move disk n from X to the middle pole.
3. Move $n-1$ disks from Y to X through the middle pole.
4. Move disk n from the middle pole to Y.
5. Move $n-1$ disks from X to Y through the middle pole.

The procedure requires three invocations with $n-1$ disks at steps 1, 3, and 5. So the recurrence relation giving the total number of moves is $a_n = 3a_{n-1}+2$. The base (stopping) case gives the initial condition, $a_1 = 2$. This is a simple linear recurrence with constant coefficients that students learn how to solve in the course: $a_n = 3^n - 1$. □

The classic example of recursion is the Tower of Hanoi, but the author's personal favorite is the next puzzle. As with the Tower of Hanoi, an inexpensive manipulative, manufactured by Mag-Nif and called "The Brain," is available for students to explore the solution.

Example 9. The Brain. This puzzle consists of n numbered switches, or levers. (n = 8 for the version of the puzzle manufactured by Mag-Nif.) The switches are all initially in the on position. The rules for manipulating the switches are:



1. the first switch can always be turned on/off,
2. the $k-th$ switch can be turned on/off only when the $(k-1)$-st switch is on and all preceding switches are off.

How many moves are required to turn all n switches off?

Solution. This is a challenging problem, so students are asked to begin by solving the puzzle for a small nontrivial case, $n = 5$ switches. In order to get at the fifth switch, the first three switches must be turned off with the fourth remaining on. So from the starting configuration, the switches must be toggled through a state where the first three are all off.

11111 => 00011− > 00010 => 11110

After the fifth switch is turned off, the next goal is to get the fourth off. But in order to get the fourth off, the third must be on. Similarly, the second must be on to get to the third, and the first must be on to get to the second. So we must progress through a state where the first four switches are all on. This leaves the problem of solving the puzzle for $n = 4$ switches —– recursion in action!

Here is the complete sequence of moves to solve the puzzle for $n = 5$.

0) 1 1 1 1 1	6) 0 0 0 1 0	12) 1 0 1 1 0	18) 0 1 0 0 0
1) 0 1 1 1 1	7) 1 0 0 1 0	13) 0 0 1 1 0	19) 1 1 0 0 0
2) 0 1 0 1 1	8) 1 1 0 1 0	14) 0 0 1 0 0	20) 1 0 0 0 0
3) 1 1 0 1 1	9) 0 1 0 1 0	15) 1 0 1 0 0	21) 0 0 0 0 0
4) 1 0 0 1 1	10) 0 1 1 1 0	16) 1 1 1 0 0	
5) 0 0 0 1 1	11) 1 1 1 1 0	17) 0 1 1 0 0	

Turn off procedure. Generalizing the above reasoning, in order to turn off n switches starting with them all on, we have to: (1) turn off the first $n - 2$ switches, (2) toggle the *n-th* switch off, (3) turn the first $n - 2$ switches back on starting with them all off, and (4) turn off the first $n - 1$ switches starting with them all on.

11...11 => 00...011− > 000...010 => 11...10 (now solve for $n - 1$ switches)

Turn on procedure. In order to accomplish step 3, another procedure is needed to turn on n switches starting with them all off. By analogous reasoning, the steps are to: (1) turn on the first $n - 1$ switches, (2) turn off the first $n - 2$ switches starting with them all on, (3) toggle the *n-th* switch, and (4) turn on the first $n - 2$ switches starting with them all off.

00...00 => 11...10 => 00...010− > 00...011 (now solve for $n - 2$ switches)

Let a_n be the number of moves to turn off n switches starting with them all on, and b_n be the number of moves to turn on n switches starting with them all off. Analyzing the procedures,

$$a_n = a_{n-2} + 1 + b_{n-2} + a_{n-1} \quad \text{and} \quad b_n = b_{n-1} + a_{n-2} + 1 + b_{n-2}.$$

Observe, however, that the number of moves to turn off n switches is the same as the number of moves to turn on n switches. This is true since the sequence of moves to solve one problem is the reverse of the sequence to solve the other. (See the solution for $n = 5$ above.) So we obtain a single linear recurrence with constant coefficients, $a_n = a_{n-1} + 2a_{n-2} + 1$ with initial conditions $a_0 = 0$ and $a_1 = 1$. The solution is $a_n = 2/3\,2^n - 1/6\,(-1)^n - 1/2 = [2/3\,2^n]$.□

Solving the puzzle involves two *mutually recursive* procedures: one to turn n switches off starting with them all on, and the other turn n switches on

starting with them all off. The notion of mutual recursion is seldom broached in undergraduate computer science classes, and that is one reason why the author likes this puzzle so much. It is a very difficult problem to attack without mutual recursion.

5 Graphs

The final unit of the course deals with graphs. Topics include basic properties and applications of graphs, isomorphism, planarity, Euler and Hamilton circuits, vertex and edge coloring, and trees. As with recurrences, students will revisit trees, graphs, and graph algorithms in the follow-on course on the design and analysis of algorithms.

The next puzzle is assigned on the day Euler's formula is presented and proven in class. The puzzle motivates the topic and paves the way for the discussion that follows.

Example 10. Platonic Solids. The Platonic solids are three-dimensional shapes where (1) each face is the same regular polygon and (2) the same number of polygons meet at each vertex. There are only five such solids:

tetrahedron cube octahedron dodecahedron icosahedron

Complete the table by counting the number of vertices and edges in each of the Platonic solids.

	tetrahedron	cube	octahedron	dodecahedron	icosahedron
faces, F	4	6	8	12	20
vertices, V	4	8	6	20	12
edges, E	6	12	12	30	30

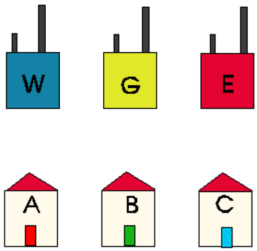
Can you find a relationship between the number of faces F, the number of vertices V, and the number of edges E?

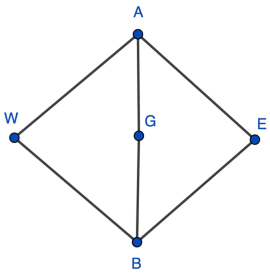
Solution. The relationship is Euler's identity, $F = E - V + 2$. The formula holds for all convex polyhedra and not just the Platonic solids. For planar graphs, the number of faces F is replaced by the number of regions R in a planar depiction of the graph. By appropriately positioning a point in space,

a polyhedron can be projected onto a plane to produce a planar graph where regions in the plane correspond to faces of the polyhedron (see [1], for example). This transformation is discussed during the lecture portion of the class. □

Two simple graphs play a key role in graph planarity. K_5 is the complete graph on five vertices, with an edge between every pair of vertices. $K_{3,3}$ is a bipartite graph with three vertices on each side, and an edge between every vertex on one side and all three on the other. Kuratowski's theorem states that any non-planar graph must contain either a K_5 or a $K_{3,3}$ "configuration" (similar to a subgraph; see Tucker [4] for details). The next puzzle establishes the non-planarity of $K_{3,3}$; a similar puzzle (not presented here) shows that K_5 is non-planar.

Example 11. Three Utilities. Three houses have to be connected to three utility companies: water, gas, and electric. Each house must be connected to all three utilities. Can you do this without the connections crossing? You must work "in the plane," and you are not allowed to route cables or pipes through any buildings.



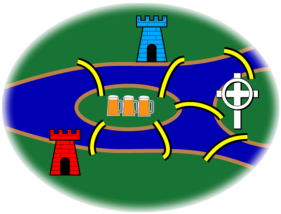


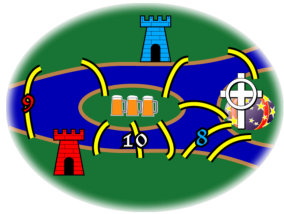
Solution. On the right, we show a planar depiction of $K_{3,2}$, with one of the houses missing. From Example 10, every planar depiction of this graph contains three regions (including the outside region). If we try to place the third house any of the regions, an edge crossing results.

Our final puzzle adds a cute story line to the Seven Bridges of Königsberg problem, the classic example for discussing Euler circuits and trails. The author finds that students are able to discover, on their own, when these properties exist in a graph by working through the puzzle. Some nontrivial thinking is involved, but this is precisely why the author introduced puzzles into the course.

Example 12. Bridges over Troubled Waters. In a faraway kingdom lived two feuding brothers. On the northern bank of the river that ran through the kingdom was the castle of the Blue Prince; the southern bank was the home of the Red Prince. Between the castles was an island on which there was a very

popular inn. Just to the east of the island, there was a fork in the river where the Bishop's cathedral was located.





a) There was a challenge among the townspeople to walk over each of the seven bridges exactly once. Many claimed to have done so late at night after quaffing ale at the inn, but no one had been able to duplicate the feat during the day. Can you explain why?

b) The Blue Prince devised a plan to build an eighth bridge so he could start at his castle, walk over each of the bridges exactly once, and end at the inn to brag of his feat. Where did he build the eighth bridge?

c) The Red Prince, infuriated by his brother's actions, decided to build a ninth bridge so he could begin at his castle, walk over each of the bridges exactly once, and end at the inn to rub dirt in his brother's face. In doing so, his brother was no longer able to achieve the feat from the blue castle. Where did the Red Prince build the ninth bridge?

d) The Bishop watched this bridge building with dismay since it upset the peace of the kingdom. He decided to build a tenth bridge that allowed all of the inhabitants, no matter where they lived, to walk the bridges and return to their own homes. Where did the bishop build the tenth bridge?

Solution. An *Euler circuit* is a path that begins and ends at the same vertex and traverses each of the edges in a graph (or multigraph) exactly once. An *Euler trail* is a path, with different starting and ending points, that traverses each edge exactly once.

The path sought in part a) cannot exist because the degree of every vertex (landmass) is odd. In order to have an Euler circuit, the degree of every vertex must be even because each time the path passes through a given vertex it must enter and leave along a different edge. A graph has an Euler circuit if and only if the degree of every vertex is even, and the graph, of course, must also be connected.

An Euler trail exists in a connected graph if and only if exactly two vertices have odd degree. These are the starting and ending points of the trail. In part b), the Blue Prince wants the vertices for his castle and the inn to remain of odd degree, and the other two vertices to be of even degree. So he builds the eighth bridge between the southern bank and the fork with the cathedral.

For part c), the Red Prince wants the vertex with his castle to be of odd degree and the one with his brother's castle to be of even degree. So he builds the eighth bridge between his vertex and his brother's. Finally, to restore peace in the kingdom, the Bishop wants all vertices to be of even degree. So the tenth, and final, bridge should be constructed between the Red Prince's vertex and the inn's. □

6 Conclusion

The introduction of puzzles has undoubtedly improved the author's discrete math class. The same amount of material is covered as the puzzles have replaced many of the examples that were previously used. Students enjoy solving the puzzles in groups, and they are definitely better prepared and ask better questions during the more formal presentation of the material.

The first time the author taught the class with puzzles, the groups wanted to stay together throughout the semester, and a friendly rivalry developed between the groups. Several students have mentioned that, although the class period is longer than most, the time seems "to fly by."

In addition to the author, four other instructors have taught the course using the same set of puzzles. Most had equally favorable results and positive comments about the approach.

References

[1] Benjamin, A., Chartrand, G., Zhang, P. *The Fascinating World of Graph Theory.* Princeton University Press, Princeton, NJ, 2015.

[2] Diaconis, P., Graham, R., Magical Mathematics. *The Mathematical Ideas that Animate Great Magic Tricks.* Princeton University Press, Princeton, NJ, 2011.

[3] Smullyan, R.M. *What is the Name of this Book?: The Riddle of Dracula and Other Logic Puzzles.* Dover, Mineola, NY, 2011.

[4] Tucker, A. *Applied Combinatorics.* Wiley, New York, NY, 2012.

Top-10 Suggestions from a Decade of Managing Undergraduate Software Teams*

Weiqi Feng and Mark D. LeBlanc
Computer Science Department
Wheaton College
Norton, Massachusetts
{feng_weiqi, mleblanc}@wheatoncollege.edu

Abstract

Sustaining a multi-year research project with undergraduates is a labor of love that leverages the very best of computer science teaching and research. We present a decade of software development during which we led an interdisciplinary research group focused on the implementation and use of a web-based app for scholars and students who wish to explore their digitized texts. In our experience, scholars, e.g., those from the Humanities, who might like to perform computational analysis in their areas of expertise and/or wish to teach their students how to do so become discouraged too early in the game. Our research model combines interdisciplinary teaching and recruitment during the school year with simultaneous scholarly activity and software development sprints each summer in a blend of graduate school and start-up-like student experiences. Led by faculty in Computer Science and English, 63 undergraduate researchers have participated. Forty-nine of these students (40% of those women) have contributed to the software, many assuming leadership roles over six software releases. Both students and faculty offer lessons learned and our "top 10" suggestions for sustaining a large software effort across multiple student cohorts.

1 Introduction

We report on a decade-long, interdisciplinary research group focused on the development and use of a web-based app that offers effective practices, workflows, and tools for scholars across the academy who wish to explore their digitized texts. This multi-year, iterative work combines interdisciplinary teaching during the school year, external and internal grant funding, and software development sprints each summer. In the sections to follow, we share our motivation for building an app that helps scholar-clients view their texts via a computational lens, we offer from both student and faculty perspectives our "top 10" suggestions for help managing a long-term project with undergraduates, and we share what worked and what didn't when faculty lead a large software effort across multiple student cohorts.

1.1 Computing Everywhere

Computer science has emerged as a pivotal discipline in interdisciplinary research across the academy and industry, from data science to machine learning, from web development to genomics where, for instance, the "stuff of life" is data [15]. At a rapid pace, computational thinking is emerging beyond STEM-centric fields and is increasingly playing a role in other fields. Our focus here is on the Humanities [8, 18], where for example Optical Character Recognition (OCR) efforts from Google *et al.* are digitizing texts and entire corpora in a plethora of languages, including rare and ancient texts such as the entire corpus of surviving literature in Old English [6]. As the "Digital Humanities" (DH) gains access to a wide array of digitized corpora and matures to a discipline that creatively defines new methods for computationally performing close and distant readings [19], a growing gap has emerged between those who apply sophisticated programming, e.g., Stylo In R [7], and those who are new to the game and need an introduction to the field. Typical of the community spirit in DH, significant efforts are underway to bridge this gap, including web-based tools for entry-level exploration including Voyant Tools [22] and domain-specific introductions to programming, e.g., text mining in R [12] and the Programming Historian [3]. Our group seeks to lower the barriers required for computer-assisted text analysis by developing tools that are easy to use, encourage effective practices, and are applicable to a wide range of scholars in various disciplines and written languages. The *Lexos* app [17] offers tools within a workflow that highlights the computational methods employed and the choices made, thereby facilitating the replication of experimental results [14]. The suite of tools includes pre-processing, segmentation, tokenization, statistical analyses, and visualizations.

1.2 Lexos

We present a decade of work as an interdisciplinary Lexomics Research Group (English, Computer Science, Statistics faculty, visiting scholars, and to date, 63 undergraduate researchers, 49 who contributed to the software) to build computational, web-based tools to meet the needs of both entry-level and sophisticated scholars who seek to computationally explore literature [9]. Figure 1 reveals our last six summer code sprints (coding hours) since we first moved to a GitHub repository in 2013, each managed in a scrum-like methodology.

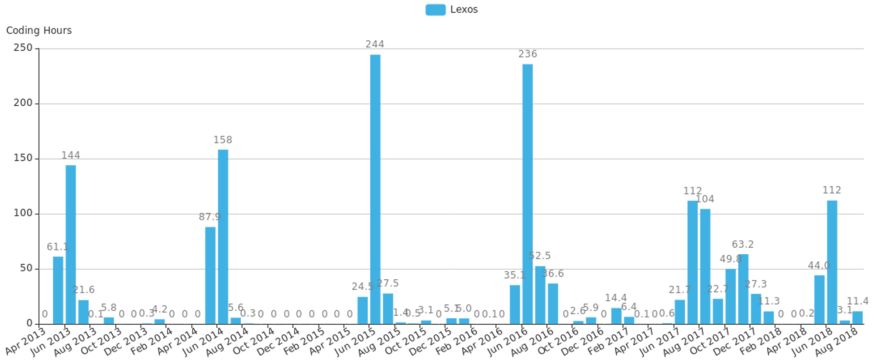


Figure 1: Total coding hours over six summer sprints [17].

Our group's model of simultaneous scholarly activity and software development each summer ensures that we continuously adapt our *Lexos* application to the needs of our core users (mostly scholars and students in Humanities fields) in an interdisciplinary blend of graduate school and start-up [2]. Beginning with humble Perl scripts a decade ago, *Lexos* [16] is presently a browser-based suite of tools that provides an easy entry to computational text analysis for scholars and their students (see Table 1 for functionalities provided in *Lexos*).

In short, the *Lexos* back-end (server-side) is implemented in Python 3.6 (as distributed in Anaconda 5.2) using the Flask microframework, based on Werkzeug and Jinja2. The front-end leverages Flask blueprints and uses jQuery and the Bootstrap 3 framework with a few functions derived from jQuery UI and DataTables. Visualizations are generated with d3.js and, increasingly, the Plotly Python library. Much of the statistical processing is implemented with the Python sckikit-learn module.

Examples of using *Lexos* with various written languages are discussed more fully elsewhere, including Old English [5], Old Norse [1], Classical Chinese [21], and modern English (e.g., a contested Edgar Allan Poe story [14]).

Table 1: A summary of tools and methods in *Lexos* v3.2.0 [13]

***Lexos* tool**	**Functionality**
pre-processing	UTF-8 encoding and options for handling punctuation, case, digits, HTML and XML tags, stop words, lemmatization, and pre-defined entities
segmenting	cutting texts by character, lines, tokens, or user-defined milestones
tokenization	bag-of-words by token or character N-grams, culling by frequency or by token presence in a percentage of documents
statistical analysis	corpus-wide filesize anomaly detection, cosine similarity comparisons, clustering by hierarchical agglomerative, k-means, and bootstrap consensus tree (beta), "top words" (prominent term detection by text, corpus, or class), and content/sentiment analysis (beta)
visualizations	Cluster analysis dendrograms (hierarchical and bootstrap consensus), k-means clusters of PCA reductions as Voronoi cells, 2-D, or 3-D Scatter plots, word and bubble clouds, and "rolling window" plots showing term frequency over the course of a document or corpus

2 Ten Suggestions for Managing Undergraduate Software Teams

From both the student and faculty perspectives, we offer ten suggestions for managing a long-term software project with undergraduates. First, we state the obvious point that most of our undergraduate developers have not worked on a software project of considerable size. For students, the most important aspect is that a long-term project is more "realistic" . The work is not homework. For most, it is the first of what will be many experiences in their career where their contribution is not "one and done" , a homework "submitted" . We offer two such experiences. First, 607 issues were opened on our github repository in the past six years. Most of the issues were bug-solving or project performance related, the remainder being project management and future plans related. Of those 607 issues, our student development team resolved 507 of them. Second, finding, using, and maintaining libraries (both on the front and backends) is a constant need and challenge. In some cases, it is because the libraries stopped

being supported or we found libraries that are faster and the change helped us to create more efficient code. Our ten suggestions are set in this context of a large software project for clients who help design and apply our app to their scholarship, experiences that are vital for undergraduate developers and their budding careers.

2.1 Recruit actively and iteratively

We employ and encourage a multi-faceted strategy for finding and recruiting a cohort of undergraduates each year. Finding the right mix of students for a new team is critical and requires a full-year recruitment mindset (not unlike that of a coach who recruits athletes). The following list summarizes our recruitment activities.

(a) **Finding funding**: External grant funding is not essential but very helpful. We find that even our unfunded grant applications provide long-range plans, specifically next steps that are vital for maintaining momentum. The Lexomics Group benefited from three rounds of National Endowment for the Humanities (NEH) funding and it helped sustain our decade-long effort. Internally, Wheaton funds one student position per faculty member per summer ($3K student, $3K faculty). In addition, we scour the list of students with financial awards and leverage our institution's new guaranteed access to internship funding, Wheaton typically funding another 2-8 students per summer ($3K each). We consider compensation for students a must. Our experience dictates that volunteer-only groups do not generate the serious attention to the mission that is required if the outcomes are highly sought (such as a new software release).

(b) **Recruit in our classrooms**: Two faculty team members in Lexomics teach at the same institution (English and Computer Science) and regularly teach pairs of connected courses. ("Connected courses" are a graduation requirement. Students must complete two pairs of connected, interdisciplinary courses, e.g., "Computing for Poets"is connected to "J.R.R. Tolkien", "Anglo-Saxon Literature" , or "Digital Maps") [15]. Thus, we regularly work closely with and recruit students during the semester, especially those who show a passion for research. We have been especially successful with finding students from the humanities during our connected teaching. Perhaps most importantly, our students see by example that faculty research does not stop during the academic year and that undergraduate scholarship is worthy of compensation.

(c) **Finding student leaders**: The success of a large software project hinges on leadership, in particular on the software skills of very talented students, typically one or two per cohort. These students seek opportunities for leadership *and* want to implement architectural changes in the systems design. Deliberate care is made throughout the year to learn of student plans

and desires in order to recruit the right leaders. Recruitment efforts for such multi-year projects should plan to retain at least one student who returns to the group for a second summer. For example, international students are not eligible to apply for many national research programs (e.g., NSF's REU) but value the summer research experiences.

(d) **Finding students who can work independently**: We try to provide an enjoyable research experience for all students, offering options for areas of interest with flexible timelines. Over the last decade, we map student's level of experience onto a range of tasks. We hope all our students enjoy text mining and desire to learn more about it. Even a novice developer can learn the software and quickly become an expert tester, often suggesting new features. Students with more experience are often interested in learning the structures and systems of the entire repository of code. As we recruit, we pay attention to the team's balance of experience. Given a team of 5-10 developers, a best case is when at least half of the students can work on implementation while others can take a lead on testing (see below).

2.2 Proceed the initial sprint with a bootcamp

We find a 3-5 day commitment is required in order to acclimate new developers to their development environment and the *Lexos* software repository.

(a) Our software development team, like our public servers, use **Linux**. Most development software, tools, apps, drivers etc. are supported on popular Linux distributions and package management on Linux is straightforward, causing few compatibility issues. Alternatively, on Windows, many Python packages like Scikit-Bio require Visual Studio code libraries. We train students to become increasingly familiar with the shell (e.g., bash).

(b) IDE: We use **PyCharm** [11] as our Integrated Development Environment (IDE). The professional version is free to students, it supports the Flask microframework very well, and PyCharm provides excellent git integration. For students who has never learned about git, we believe PyCharm provides a most "breezy way" for them to learn to develop with others in a code repository. This is especially true when a conflict occurs: PyCharm helps the developer view any conflicts and merge them as needed. Importantly, sharing PyCharm configurations is easy, for example, we make sure everyone is using the same code style settings to ensure consistency [20, 24, 25].

(c) Learning **github MarkDown** and **git** practice: Over the years, we find that our new team members struggle with Git and we automatically factor in time for practice, for example, we introduce a "toy repo" . Additionally, we want students to learn GitHub MarkDown: encouraging them to share issues, communicate with each other, and review other's code through the github website. So, early in the bootcamp each summer, we let students work on one

markdown file together. Each of them receives a specific task to explain one markdown command, e.g., create a list. When they are done, we let them open a pull request. Then students review each other's pull requests. Finally, we let them merge the pull requests together. After each pull request is merged to the master, they have to update the next pull request before the merge can happen. The chances that they have to deal with conflicts are great because they started to work on an empty file together. So, after dealing with the conflicts a number of times, everyone gains a basic idea of when conflicts happen and how to resolve them. This way, they are learning both git and markdown at the same time.

2.3 Use continuous integration (CI) tools

We highly recommend continuous integration tools as a best practice. CI detects integration errors at an early point and provides software leads with a full view of the project code. CI tools build after running a suite of automated tests for every commit. Thus, we encourage students to commit regularly after each small step is done. Instead of fixing one large problem at the end, most issues can be easily identified and addressed with each integration. A direct benefit is that CI helps maintain the code quality to ensure that team members produce well documented and formatted code, a code standard shared by all, thereby reducing potential friction between team members. *Lexos* applies Travis CI (Linux) [23] and Appveyor (Windows) [10], and includes checks for code style, documentation, and unit tests. In order to help students understand the idea of CI tools and also to familiarize them with the *Lexos* software, we ask students to improve function documentation and in-line code comments during the first one or two days, providing them another chance to read through CI outputs and gain even more practice with git skills (see 2.8 below).

2.4 Use a good, modularized structure

After six years of development as a web-based app, *Lexos* is comprised of 20K lines of Python on the server-side backend (recently refactored to Python v3.6 [26]) and 9K lines of Javascript and 12K lines of HTML/CSS on the frontend. With this code base, continually training new cohorts of students remains a challenge. Based on our experience, an average team member spends about a month prior to contributing to the repo. In response, our recent refactoring has attempted to make the *Lexos* software structure as simple as possible. Over the last two summers, we moved to a hierarchical structure, where each leaf is a fully functional tool. In particular, we applied a model–view–controller (MVC) pattern. Each tool in *Lexos* has a receiver that accepts and parses options from the user and passes values on to a model for calculation or graph rendering. Eventually, a view file renders the result. Based on our experience,

we find that most students can explain the idea of the *Lexos* structure within a week of working with the repo.

2.5 Apply unit testing

Writing unit tests is beneficial, helping developers capture bugs early and locally. Whereas unit tests are carried out by our developers, bugs are likely spotted and fixed before integration. Since the bugs are found early, unit testing helps reduce the cost of bug fixes. As seasoned developers will note, one of the hardest steps in debugging is finding a bug's location. Unit testing helps developers focus on the function(s) causing the bug. So instead of going through an entire file or even multiple files, developers only need to deal with one function. As noted earlier, *Lexos* has over time had to change some libraries. Having unit testing in place makes the upgrading steps much easier, thereby facilitating safe refactoring.

We always want developers, especially those early in their career, to "think before they start to type." Writing unit tests before actually coding is a good programming discipline. Unit testing exposes the edge cases; as shown in the software engineering literature, considering those cases leads to safer code. Additionally, unit testing provides a unique documentation-perspective of the entire system. For example, if a developer finds the documentation of a particular function too abstract, studying the corresponding unit tests offers a new view. By looking at "inputs" and the correct output, novice developers are more apt to understand the functions faster.

2.6 Conduct peer reviews

Peer review ensures consistent design and implementation, a fresh set of eyes to identify bugs and simple coding errors before integration. Each summer, the *Lexos* team appoints a student software lead. All pull requests opened on GitHub must get approval from the leader before being merged. (Currently, our group uses the git branch system together with GitHub and PullApprove [4] to ensure that all the code in the main (master) branch is reviewed by specific developers). Though we do not ask all students to make suggestions to pull requests, we do ask them to read through each and leave comments if they cannot understand what the code does. Sharing comments and reviews with peers both in writing and orally is a valuable skill. Projects benefit from team players who can share how comprehension was hindered by unclear variable namings, nested loops, and/or a lack of in-line documentation. We submit that peer review is an essential steward of writing readable software. Of course, peer review is also a great way to have students learn from each other and discuss their own techniques. We find our team members are willing to learn how to

share their own knowledge through peer review and our software product is better for it.

2.7 Meet in daily stand-ups

Holding daily meetings, in our case, a "stand-up" each weekday morning (10 a.m.) is deceptively simple and ridiculously important. Daily scrums and weekly sprints require each team member to answer two questions: (i) Where are you? (ii) What is blocking you? For most undergraduates, peer review (of code, see above) and daily stand-ups are personally challenging. It can be challenging to publicly share our work; being subject to even positive criticism can be difficult. But our experience shows a wonderful willingness to participate, students marveling at how much can be learned from peers in the group (not just their professor). In addition to the positive learning experiences, from a software engineering management perspective, daily stand-ups enforce "no place to hide" . Knowing the status of each team member leads to wiser assignments in subsequent sprints.

2.8 Lead toward success: assign easy tasks early

To start new team members off, we ask them to carefully review the warning messages on our front-end. For students who do not have any experience with front-end web development, we ask them to simply improve some of the comments or update some of our testing suites. We believe this step is crucial because it helps students build their confidence, for example, when using git. Early on, most students are not confident with directly changing the code base, often concerned that they are using git in an incorrect way. Our experience shows that students like to review and perhaps upgrade warning messages and associated documentation; completing a small set of items being a confidence boost. In this way, our developers are actively committing and mastering the mechanics of the development environment, not to mention a new opportunity for then to be exposed to the CI reports.

2.9 Improve one tool

Due to the time gap between consecutive summers, we often return to *Lexos* development and find some functions in our code base which are now deprecated. We assign each student one specific tool and ask them to "adopt the code." Action items include reading the documentation for the libraries used and replacing deprecated functions. We argue that this type of learning plays an important role for programmers. No curriculum can prepare them for the range of issues, problems, and challenges they will face. Here, we train them

how to study by themselves, for example, how to figure out how to find the most reliable online documentation and importantly, how to extract information from questions asked in online communities. This is where self-learning starts. During the last ten years of development, many students have designed, suggested, and implemented amazing ideas for the tools as they were fixing them. For example, a re-designed User Interface (UI), new visualizations, and optimized algorithms.

2.10 Add new functionalities

This is where some exciting interdisciplinary group work happens. Our "power" users, humanities scholars, many who are probing ancient texts, often run a local install of *Lexos* using large numbers of texts. These users are consistently requesting new functionalities for the tools. Each year we follow a goal of inserting at least one new tool into *Lexos*. We ask the entire team to work on this together. This often appears in the second half of the summer sprint. By that time, students have already figured out their favorite parts and team strengths are more evident. Each new tool contains three essential elements: the backend (computational), the frontend (UI), and a testing suite. The effort affords new opportunities for student leadership on smaller tasks that are contributing toward a larger, common goal.

We close this section of suggestions with a note on the critical importance of *interdisciplinary* teams. Our Lexomics Research Group is a vibrant blend of scholars of many languages, genres, and time periods who work alongside software developers. Our clients are our colleagues, many who are experts in ancient texts *and* write algorithms on the whiteboard, often participating in the design of new tools. Spending time together in the same lab is vital. For a decade, undergraduate developers have been listening to and designing new tools with their professors and colleagues in the humanities. Unlike classroom assignments, the energy of building *for* others, *with* others, is a most welcome and refreshing benefit of managing a long-term, undergraduate software team.

3 Pitfalls and Cautions

We share a set of pitfalls and cautions to faculty who might choose to lead a large software effort across multiple student cohorts.

3.1 Managing a long-term software project is not for all faculty

Faculty must acknowledge that each year a new cohort arrives and training must take place each summer anew. For untenured faculty with very real publication pressures, the time commitment is an important consideration and in fact, managing a team may not be the wisest use of research time depending on the faculty member and institutional research expectations. We recommend open discussions with colleagues to help faculty make wise choices. All faculty leaders in the Lexomics Group have tenure and their research with undergraduates is an acknowledged part of their active scholarship.

3.2 The Problem of Scale

There is a limit of scale when offering research experiences to undergraduates. No faculty research group can offer experiences to all students and that may discourage some faculty (we recommend that each faculty member advise at most 3-8 undergraduate developers per summer). That said, our research students laud the experience as a pivotal experience prior to their graduate and/or industry work and the faculty involved consider it a vital part of their mission to prepare the next generations of researchers. The truth is that this work does not scale well to all students.

3.3 Peer review is hard

Tensions can mount when committed code is returned for failing a review, again and again. This is especially true when the failed review is from a peer. We might argue that *this* is part of the lesson; however, we felt we should acknowledge our mixed success with navigating this issue.

3.4 Interdisciplinary software development is harder

The interdisciplinary team was overall a boon for the project's ability to establish a user base beyond its home institution, evolve over time to address an increasing range of scholarly questions, and to bring students into contact with disciplines outside their majors. For the computer science students, who are often given problems and data sets from the STEM disciplines, developing software to answer questions about the often messier "cultural" data frequently studied in the Humanities provides a valuable experience, as does designing for a user base willing to explore, but not accustomed to computational thinking. But working in multidisciplinary teams is nuanced and our setup has not always worked. For instance, the software developers often incorporated

options that were built into the scikit-learn module for data science, such as multiple distance metrics for hierarchical clustering, without regard to whether these options were useful or meaningful for users. Despite our sharing physical space and time, more directed communication strategies between the computer science and Humanities students during the summer code sprints would have helped address some issues. A related concern was the lack of front-end development and data visualization skills in the computer science cohorts, which made it more challenging for them to address user interface and user experience issues.

4 Meeting the real need for professional development

Computer science is helping drive the entrepreneurial spirit appearing across the academy. And so, our young discipline assumes new leadership responsibilities on our campuses. We present a multi-year software project that affords cohorts of undergraduates a very real pulse of the world of software development as they build "for others", specifically, a growing number of literary scholars who seek computational methods for close and distant readings of their texts. For computer science students in particular, the experience is invaluable. But we do a disservice to the faculty if we fail to acknowledge how leading such an effort provides "built in" professional development. Unlike many disciplines, computer science is in a constant state of change and faculty face unique challenges as they balance teaching load, service, research, and a need for professional development. We submit that building apps in multi-year projects for passionate clients can help faculty "learn for life", including more than one lesson from our talented students.

References

[1] Berger, R. and Drout, M.D.C. A reconsideration of the relationship between víga-glúms saga and reykdla saga: New evidence from lexomic analysis. *Viking and Medieval Scandinavia*, 11:1–32, 2015.

[2] Boese, E.S., LeBlanc, M.D., and Quinn, B.A. . EngageCSEdu: Making interdisciplinary connections to engage students. *ACM Inroads*, 8(2):33–36, 2017.

[3] Crymble, Adam, Fred Gibbs, Allison Hegel, Caleb McDaniel, Ian Milligan, Evan Taparata, Amanda Visconti, and Jeri Wieringa. *The Programming Historian. 2nd ed.* http://programminghistorian.org/., 2016.

[4] Dropseed LLC. Pullapprove. `docs.pullapprove.com/`.

[5] Drout, M.D.C. and Smith, L. *A Pebble Smoothed by Tradition: Lines 607-61 of Beowulf as a Formulaic Set-piece*. Oral Tradition, 2018.

[6] Drout, M.D.C., Kahn, M., LeBlanc, M.D., Nelson, C. . Of dendrogrammatology: Lexomic methods for analyzing the relationships among old english poems. *Journal of English and Germanic Philology*, 110(3):301–336, 2011.

[7] Eder, M., Kestemont, M. and Rybicki, J. . Stylometry with R: A package for computational text analysis. *R Journal*, 16(1):107–121, 2016.

[8] M.K. Gold. *Debates in the Digital Humanities*. University of Minnesota Press, 2012.

[9] Lexomics Research Group. Lexomics research group. `http://lexomics.wheatoncollege.edu`.

[10] Appveyor Inc. Continuous integration and deployment service for Windows developers - AppVeyor. `https://www.appveyor.com/`.

[11] JetBrains. Pycharm. `https://www.jetbrains.com/pycharm/`.

[12] Jockers, M. . *Text Analysis with R for Students of Literature*. Springer, New York, 2014.

[13] Kleinman, S., LeBlanc, M.D., Drout, M., and Feng, W. Lexos v3.2.0. `https://github.com/WheatonCS/Lexos/doi:10.5281/zenodo.1403869`.

[14] M.D. LeBlanc. Toward reproducibility in DH experiments: A case study in search of Edgar Allan Poe's first published work. digital humanities. 2017, Montreal, Canada, August 2017. `https://dh2017.adho.org/abstracts/027/027.pdf`.

[15] LeBlanc, M.D. and Drout, M.D.C. . DNA and 普通話 (mandarin): Bringing introductory programming to the life sciences and digital humanities. *Procedia Computer Science –International Conference On Computational Science*, 51:1937–1946, 2015.

[16] Lexos. Lexos development version. `https://github.com/WheatonCS/Lexos/tree/master`.

[17] Lexos. Lexos tools. `http://lexos.wheatoncollege.edu`.

[18] Alan Liu. The meaning of the digital humanities. *PMLA*, 128(2):409–423, 2013.

[19] F. Moretti. *Distant Reading*. Verso Books, 2013.

[20] Lexomics Research Group (n.d.). Python coding style guide. `https://github.com/WheatonCS/Lexos/wiki/Python-Coding-Style-Guide`.

[21] Nichols, R., Slingerland, E., Nielbo, K., Bergeton, U., Logan, C., Kleinman, S. Modeling the contested relationship between analects, mencius, and xunzi: Preliminary evidence from a machine-learning approach. *The Journal of Asian Studies*, 77(1):19–57, 2018.

[22] S. Sinclair and G. Rockwell. Voyant tools. `http://voyant-tools.org/`.

[23] GmbH. Travis CI Travis CI. Test and deploy with confidence. `https://travis-ci.com/`.

[24] Guido van Rossum, Jukka Lehtosalo, and Lukasz Langa. PEP 484 – Type Hints. `https://www.python.org/dev/peps/pep-0484/`.

[25] Guido van Rossum, Barry Warsaw, and Nick Coghlan. PEP 8 – Style Guide for Python Code. `https://www.python.org/dev/peps/pep-0008`.

[26] Zhang, C., Feng, W., Steffens, E., de Landaluce, A., Kleinman, S., and LeBlanc, M.D. Lexos 2017: Building reliable software in Python. *The Journal of Computing Sciences in Colleges*, 33(6):124–134, 2018.

Factors Influencing Women Entering the Software Development Field through Coding Bootcamps vs. Computer Science Bachelor's Degrees*

Sherry Seibel, Nanette Veilleux
Simmons University
300 Fenway, Boston, MA 02115
{sherry.siebel, veilleux}@simmons.edu

Abstract

The gender disparity in technology related fields is well known and well documented. Only 18% of computer science undergraduates and 26% of computer science professionals are women. Despite numerous interventions in the past decade, women are still underrepresented in the undergraduate pipeline. However, in 2016 35% of post-baccalaureate "coding bootcamp" participants were women, suggesting these women may have different characteristics, attitudes, and mindsets than women and girls who sought a traditional college path. Examining data from 18 interviews, the results show that the bootcamp women entered computer science as adults after gaining a better understanding of what the industry would require, having familial and peer support, and some positive experiences in introductory classes and workshops. These women report being initially deterred from participating in a computer science major due to a lack of understanding of the discipline, a low mathematical self-efficacy, the belief they wouldn't be able to do what was considered challenging work, and/or, indirectly, anticipating being uncomfortable when the major was comprised mostly of men. In contrast, the women who studied computer science in college were exposed to coding earlier and had a better understanding of what the major would entail. They also believed they were good at math and were encouraged to pursue a computer science major.

1 Introduction

The shortage of women in computer science (CS) programs is well known and well documented. Since 2013, only 18% of undergraduates in the computer sciences were women [11], despite women earning 57% of all bachelor's degrees [15]. Through the early 80s, women in CS appeared close to reaching gender parity. The number of female CS majors was at a steady increase with women earning 37% of undergraduate CS degrees [4]. In 1984 a cultural, gendered shift in computing began permeating college campuses, which resulted in a sharp decline of female CS graduates by 1987 [17] despite the increase of women in other STEM fields such as psychology and biological sciences [14]. One hypothesis for this shift may be the advent of personal computers available for public consumption, with companies marketing them to boys as videogame platforms. As a result, most personal computers were owned or accessible to males, and popular culture perpetuated these views [17].

The gender gap still persists in traditional educational models despite the growing number of students in computer science majors. However, the number of women entering the workforce is beginning to shift due to many factors, including the emergence of coding bootcamps. This study examines the attractiveness of these bootcamps for women who presumably choose not to pursue the discipline as traditional college students.

Coding bootcamps are overwhelmingly for-profit schools that provide full-time, immersive web development education. While many bootcamps offer online options, this paper will focus on in-person matriculation. Of the 95 operating bootcamps in the United States, the average duration and tuition in 2016 was 12.9 weeks and $11,451 respectively [11]. In 2016, the average age of the 15,077 coding bootcamp graduates was 30, and 35% of those graduates were women [7, 11]. In comparison, there were 64,405 university graduates in the computer sciences (including Information Technology and other related majors) in 2016 [11] but only 11,592 (18%) were women [12]. If one looks only at CS majors, 2,392 of 15,633 (15.3%) graduates in 2015 were women [13]. Coding bootcamps produced an estimated 5,276 women graduates in 2016, slightly more than double the 2015 women CS graduates, and slightly less than half of all women majoring in any form of the computer sciences. While 2017 saw 4,696 (29%) women bootcamp graduates, statistics released by Course Report in December 2018 project 6,833 (37%) women bootcamp graduates for 2018 [6, 8].

Coding bootcamps were not created with the goal of achieving gender parity in computing [9]. The concept was informally conceived in November 2011 when Shareef Bishay, future founder of Dev Bootcamp, who initially solicited six individuals to spend 8 weeks learning Ruby on Rails full-time through the Hacker News Forum [2].

Since then, the industry has ballooned and the volume of women consequently committed to learning programming through bootcamps is unprecedented. While the gender ratio of undergraduate CS enrollment stagnates, coding bootcamp gender diversity is steadily expanding. The central question of this present study is: why do women, presumably only slightly older and already college educated, seek to enter a discipline that they avoided when entering college? Related questions concern whether the bootcamp women see themselves differently or see the discipline differently. In short, this study seeks to understand the social motivators behind the larger percentage of women seeking a career in coding through nontraditional channels with respect to the participation through traditional college CS preparation. The results from this study could be used to identify steps towards achieving gender equality in undergraduate CS programs, or, failing that, in coding professions.

2 Methods and Materials

2.1 Study Population

This qualitative study was comprised of 18 women: 9 were CS graduates and 9 were coding bootcamp graduates. Four coding bootcamp graduates attended a program in Massachusetts, two attended in New York, and three attended in Illinois. Three CS graduates went to universities in California, three went to universities in Massachusetts, and three went to universities in Pennsylvania. All of the CS graduates were employed when participating in this study.

Six coding bootcamp graduates attended a Ruby on Rails program and all were employed doing web development at the when participating in this study. The remaining three coding bootcamp graduates attended a JavaScript program. One was employed doing web development and two were not; however, the two unemployed graduates finished their coding bootcamp two months prior to their participation in this study, whereas the other coding bootcamp graduates concluded their programs more than six months prior to their participation in this study. The ages of the coding bootcamp graduates during the time of participation in this study had a mean age of 29.67 years old (25-40, median: 28), and CS graduates a mean age of 25.7 (21-30, median 25).

2.2 Study Method

Interviews were conducted using a semi-structured instrument to gather qualitative data, and all interviewees were given codes to preserve anonymity. Two interviews were held at an academic institution, two at an interviewee's home, one at the interviewer's home, one at an interviewee's workspace, and 12 online

through Google Hangouts. Two of the interviews were conducted with other people present with the interviewees' consent. In both cases the other people present were family members. The interviews took place between September 2017 and August 2018.

Participants were recruited virtually, verbal announcements, and via word of mouth. Emails were sent to recent Simmons University CS alumni, a call for participation was posted on Facebook, and the study was advertised in four Slack channels: Women Who Code Boston, Boston Ruby Women, RailsBridge Boston, and the Philadelphia chapter of Girl Develop It. The emails to recent Simmons University alumni and the Facebook post were not successful in recruitment. The Slack channels yielded one CS graduate and two bootcamp graduates. Two CS graduates and four bootcamp graduates were recruited following an announcement at a RailsBridge Boston event. The rest of participant recruitment was via word of mouth. Participants were asked at the end of the interview if they knew others who might be interested in joining the study. Participants who did provided contact information or connect the individual to the researchers either by Facebook, email, or Slack, allowing for a personal introduction. This approach facilitated the enrollment of six CS graduates and three coding bootcamp graduates.

Before each interview commenced, participants signed a consent form and the recording process was explained in detail. The women were shown the recording mechanism, told the average length of the interviews, informed they could skip questions or stop the interview at any time, and encouraged to ask any questions they might have.

Participants were asked 85 questions about their backgrounds, peer interactions, opinions on math, social motivators and inhibitors, self-esteem, and industry perceptions. Eight of those questions were structured to gather demographic data at the end of the interview session. The interviews were recorded using the password protected VoiceRecorder iPhone app, and ranged from 25-91 minutes depending on amount of detail the participant provided.

2.3 Data Analysis

Results from the interviews were transcribed and coded by one of the authors using a Conversational Analysis Codebook. Codebook themes included a) Developmental factors including computer-related background such as childhood access to computers and coding instruction, b) Previous assumptions and attitudes on software development and the skills necessary in this profession and c) Previous assumptions and attitudes on the subject's ability to fit this profession, either by the ability to gain the necessary skills (self-efficacy) or ability to match, demographically, the existing workforce.

3 Results

3.1 Developmental Factors

In these results, the two target subject populations will be referred to as Computer Science or CS graduates (those who graduated from a traditional college CS major or similar background) and bootcamp graduates (those who finished a post-baccalaureate bootcamp training program).

Except for one CS graduate, all of the participants had access to a computer as a child, shared with other members of the family. Four computer science graduates and one bootcamp graduate felt they were the primary user for the machine. Eight of the nine CS graduates had been exposed to coding before arriving at college, and all of those eight had formal coding classes before college. In contrast, five of the Bootcamp graduates wrote some code prior to college but only one had taken a formal CS class.

As anticipated, the bootcamp subjects were older when they chose to pursue a software development career. The ages at which the coding bootcamp participants decided to change careers and enroll in a bootcamp were between 22 and 38 years old with a mean age of 26.89. The CS graduates decided on their majors at ages 12 - 20 with a mean age of 17, consistent with typical undergraduate major choice. The average age of the nine bootcamp graduates interviewed for this study were slightly younger than the 2017 and 2018 national average of 29, and the 2016 national average of 30.41[9, 6].

3.2 Attitude towards Computing Profession and Required Skills

A main contributor to bootcamp graduates' delay in studying CS (e.g. in college) appears partially based on belief the major and field are math intensive, coupled with low mathematical self-efficacy. Only three of the nine bootcamp graduates considered themselves to be good at math. One subject reported:

> *So, personally I don't feel like I'm that good at math. And I also I don't like math. And actually, since you bring up math, I do think that was also maybe part of why I never thought about coding because I did have this idea in my head that you needed to be good at math to be a software engineer and in practice, you really don't.*
> - Bootcamp graduate

In contrast, while CS graduates also initially believed that software development heavily required mathematics skills, seven of the nine computer science graduates believed they were good at math. Interestingly, none of the graduates from either the computer science or bootcamp programs are currently

using post-algebra mathematics in their current jobs, and most no longer believe a strong math background is necessary. All emphasized logical reasoning as the main requirement for being a competent coder.

> *The difference in background, etc. between CS majors and boot campers who now do the same job, is pretty interesting. It kind of shows that CS programs could and should do better. Calc is such bs.* - Computer Science Graduate

Only one computer science graduate currently maintains that coders do need strong mathematical backgrounds to be effective in her job, despite reporting that she herself didn't use complex math in her work. However, the subject pool did not include women working in artificial intelligence, robotics, or machine learning specialties that typically would require advanced math and often advanced degrees.

More generally, subjects reported initially having vague and incorrect assumptions about professional skills and aptitudes required for a software developer. Only one of the bootcamp graduates, an engineering major, reported that she previously knew what a computer science curriculum might entail. Only two of the computer science graduates stated that they understood the major requirements; however, their university required they specifically apply and be admitted to the CS program before starting freshman year, so they were more likely to engage in investigating more closely. One bootcamp subject discusses her retrospective on the nature of the work that a software developer engages in:

> *I thought I was going to do marketing or something creative and I never thought of software engineering was actually a really creative field and then now I know, actually it's extremely creative. I just don't think a lot of women or just a lot of people ... have a clear idea of what software engineering entails. ... I didn't realize until like later on when I ended up working in tech ... and realized "oh wow, it's actually quite creative and you're building something every day." Yeah, and crafting and something while problem solving.* - Bootcamp Graduate

Most of the participants rated themselves as being more comfortable around other women. One study of middle school girls showed that when placed in an environment comprised entirely of other girls, they were more likely to participate in classroom discussion, specifically mathematics, than when placed in coeducational classrooms. They cited an increased comfort around female peers as the facilitator of this behavior, as well as instilling self-concept as mathematicians and increasing their ability to learn math [21]. The trend

continues into post-secondary classrooms, shown by another paper where undergraduate women were more comfortable and excelled more in introductory classrooms comprised of mostly other women [5]. Based on the comfort around other women interviewees from this study expressed, the preference for majority women learning environments seems persistent across age groups. Given the well-founded belief that most of their workplace colleagues will be men may have an influence on their decision as well.

3.3 Attitudes towards one's abilities

As stated above, the bootcamp graduates reported having low math skills self-efficacy more often than the CS graduates. Most reported not knowing, or having not had the chance to fully explore, their coding abilities until they entered either a traditional CS major or a bootcamp. Most subjects expressed "wishing they had coded earlier" in both populations: eight bootcamp graduates and six of the CS graduates wish they started coding earlier.

In terms of assessment of their abilities, eight of the bootcamp graduates were surprised by the ease in which they learned coding. Two bootcamp graduates likened writing code to writing an essay, and every participant in the study conceded that most of coding is logic-based, not what they had thought of as math-based. Three CS graduates were also surprised by their strong coding aptitude, one because of a negative experience in a high school CS class, one because she had been exposed to coding at an earlier age than her classmates and as a result was learning faster, and the third because of her perceptions of the media's portrayal of coding.

> *. . . my [Discrete Math] TA was fantastic, and she was a computer science major. And up until that point it really hadn't even occurred to me that I could do computer science. Um but watching her be like, amazingly good at explaining the material that she definitely understood I was like "oh this is a thing that women can do." It hadn't occurred to me until I saw her. It sounds so ridiculous now.*
> - Computer Science Graduate, initially a Math major

This surprise reveals the lack of understanding both of the needed skills, as described above, and of the subject's assumptions about whether she could acquire such skills, hinting at holding a fixed mindset with respect to these skills. When some of the bootcamp subjects were actually able to experience coding in the right environment, they adjusted their assumptions. Five of the bootcamp graduates attended workshops prior to enrolling in a bootcamp, and four cited them as catalysts for making their career change to coding. Specifically, they commented on the supportive environment and helpful instructors.

In terms of fitting into the profession, all of the CS graduates and seven of the bootcamp graduates mentioned encountering "know-it-alls" in either their CS major or bootcamp, and all mentioned encountering them once they entered the workforce as professional coders. The effects of "know-it-alls"on the participants' psyches ranged from negligible to severe. One interviewee defined "know-it-alls" who created toxic environments as "knowledge parading" by confidently exclaiming that a concept was easy or using complicated, although sometimes inaccurate, jargon. The participants reported the belief that this tactic was used as a means to belittle others and reported participating less in classes as a result. While all the women overcame this hurdle, they report the fear of appearing to have inferior knowledge as a factor still at play in their professional lives and impacts self-confidence.

> *I was successful in the class 100% because the class was set up so "smartasses" who took computer science in high school couldn't be better because they solved the problem before.* - Computer science graduate

The importance of a healthy peer group was further highlighted by all coding bootcamp graduates and seven CS graduates attributing some of their success to peer support while learning how to code. Three bootcamp graduates and three CS graduates strongly emphasized that they would not have been motivated to either sign up and/or stay in their program without other women peers supporting them. Six bootcamp graduates mentioned at least one woman as a catalyst for changing careers, whether it was a friend, mother, character in a book, or desiring to be a good role model to young daughters and girls.

Seven CS graduates were encouraged to pursue the major by their friends, family and/or significant other after expressing a desire to code, and two received mixed feedback. In contrast, three bootcamp graduates were encouraged to attend a bootcamp by family, friends, and/or significant other after initially expressing a desire to do so. When asked to rate their self-confidence, comfort in voicing their opinions, and comfort making mistakes, the women who responded in both cohorts were on average more at ease in front of women than men.

4 Discussion

4.1 Developmental Factors

In the past, a hypothesis for the 1980's downturn explaining the mechanism for women not entering Computer Science was the lack of childhood access to

computers [17]. Only one of 18 participants did not have access to a machine, indicating other factors are now at play.

No other development factors seemed to stand out as influential in this small subject pool except for a slight difference in prior coding experience. However, even those girls who had experienced coding activities before high school (8/9 CS majors and 5/9 bootcamp graduates) did not seem to extrapolate more accurate information about their abilities or the requirements of the field.

The last developmentally related difference between the CS graduates and the bootcamp graduates was, of course, age and the expected increase in maturity. It would be difficult to control for this factor since bootcamps are post-baccalaureate programs. The difference does suggest, however, that the factors that dissuaded these women from entering the software development field when younger have been offset. Interestingly, this doesn't seem to evidence itself with more complete understanding of their abilities (hence, "surprised at the ease of coding") or the field.

4.2 Attitude towards Computing Profession and Required Skills

There was a general misunderstanding by both bootcamp and CS graduates regarding what a CS major actually entails. Only three participants understood the structure and details of a CS major before enrolling in college. This could be attributed to how broad CS employment opportunities are, encompassing areas from developing artificial intelligence and machine learning algorithms to informatics and networking management. When categorizing degrees, NCES lists 26 sub disciplines under Computer and information sciences and support services [13]. This broadness may make a CS major difficult to describe and therefore hard to envision for young students and, as such, could potentially deter enrollment for those who don't naively envision themselves in the field. Furthermore, the subjects from both cohorts reported not fully understanding the skills required by a software development workplace. Most ascribed mathematical talent as a higher priority skill than their current work requires.

4.3 Attitudes towards one's abilities

Low mathematical self-efficacy was related to the subjects' disinclination to formally study CS in college. While belief in one's mathematical competency may be based on actual aptitude, research shows that there is a significant cultural confidence gap independent of actual ability in mathematical performance [18]. It has been shown that women and men average equal performance in math. Perceived gender differences in mathematical ability are largely due to engrained, inaccurate social perceptions, evidenced by a 2017 Google memo

arguing biological factors are the reason for fewer women in CS, as well as a 2005 talk given by former Harvard University President Lawrence Summers [20].

However, despite the lack of mathematics in the bootcamp curriculum, 72% of 1000 interviewed companies believed their bootcamp hires were just as competent as their traditional CS hires, with 12% believing the bootcamp graduates were more competent than the CS graduates [19]. Given these two pieces of evidence, it is perhaps not surprising that many of the study participants adjusted their beliefs in the professional requirements (i.e. software development doesn't need math) rather than their belief in their own ability with respect to mathematic facility.

An important step in CS enrollment and recruitment then becomes the ability to communicate more clearly the work that software developers do, across the spectrum of applications. While this might still not encourage women to enter the qualitative heavy applications, there would still be the benefit of recruiting women into the profession in some capacity.

From this study, one also finds that students who are confronted with hostile environments in the forms of "know-it-alls" in their classes experience decreased confidence and therefore may leave the field. It has been recommended that educators intervene with this potentially toxic behavior [3] [16].

Supportive peer groups have been linked to the retention of CS students, as has the importance of a passionate educator who encourages a safe space to ask questions, attend supplemental lessons, and facilitates positive peer interactions [3]. For example, Harvey Mudd College in California was able to reach 55% woman enrollment in CS by incorporating group projects to facilitate peer bonding, placing students with similar programming skill levels in appropriate introductory courses, instilling asking for help as imperative to classroom success, and diverting students who dominate conversations [1].

4.4 Prior Coding Experience

An additional, albeit weaker, predictor of CS participation in college was prior, formal exposure to code. Five of the CS graduates took formal CS courses before college, and all participants had done some coding before college. This is consistent with data released from Accenture and Girls Who Code, showing that early engagement in CS leads to more CS majors, and the best time to introduce females to coding is before high school. However, introducing women to CS in college or as adults [10] also has some effect.

The women bootcamp graduates who took courses after college before making a career change reported strongly positive experiences regarding the culture of the courses and comfort with the subject material. Coding workshops before bootcamp enrollment changed self-perceptions on abilities and changed their

understanding of the tech industry. These factors coupled together became the catalyst for change. Women in this study entered computer science as adults after gaining a better understanding of what the industry would require, familial and peer support, and positive experiences in introductory workshops. They were deterred from participating in a CS major due to low mathematical self-efficacy, the belief they wouldn't be able to do such challenging work, not understanding the major, and, indirectly, being more comfortable around women when the major was comprised mostly of men. The women who studied CS in college were exposed to formal CS classes earlier, believed they were good at math, and were encouraged to pursue a CS major.

5 Conclusions

From this study, one can appreciate the success of BRAID colleges in increasing the number of women CS majors. The use of diverse routes into the major showcases the broad set of skills needed by a broad set of software development application areas. Furthermore, a set of entry paths allow those without prior experience to catch up free of intimidating "knowledge-parading" colleagues. A second, and certainly far more controversial suggestion arising from this study would require the willingness to diversify the CS curriculum to fit the needs of different CS application areas rather than trying to provide the broadest education for all students. This would be challenging for traditional CS professors who feel strongly that no part of a traditional curriculum should be omitted, even those aspects that were more useful initially than are perhaps today. A more appealing higher education CS reform might be to reach out to career-changing women with second bachelors or Masters' degrees (e.g. like Northeastern University's ALIGN program and others). These more experienced and slightly older women might engage in the decision to enter a software development field by a willingness to let go of assumptions about themselves and the field. Finally, by encouraging positive learning environments through healthy peer interactions and enthusiastic professors, CS becomes less intimidating.

Finally, to address the lack of specific understanding what courses or skills are most useful in a Computer Science major, colleges might increase awareness by hosting short, introductory, ungraded workshops for their women students to try coding.

References

[1] Barker, L. J., Cohoon, J. M. Key practices for retaining undergraduates in

computing. National Center for Women Information Technology, 2009.

[2] S. Bishary. Tell HN: I want to teach you web development. in 8 weeks. for free (sort of). hacker news web forum, 11 november 2011. https://news.ycombinator.com/item?id=3267133.

[3] Indeed Blog. What do employers really think about coding bootcamps? 2 may 2017. http://blog.indeed.com/2017/05/02/what-employers-think-about-coding-bootcamp/.

[4] ComputerScience.org. Women in computer science. www.computerscience.org/resources/women-in-computer-science/. Accessed 19 November 2018.

[5] Dasgupta, N., Scircle, M. M., Hunsinger, M. . Female peers in small work groups enhance womens motivation, verbal participation, and career aspirations in engineering. *Proceedings of the National Academy of Sciences*, 112(16):4988–4993.

[6] L. Eggleston. Coding bootcamp alumni outcomes & demographics report. course report, 19 dec. 2018. www.coursereport.com/reports/coding-bootcamp-job-placement-2018. Accessed 20 January 2019.

[7] L. Eggleston. Coding bootcamp market size study. course report, 19 july 2017. www.coursereport.com/reports/2017-coding-bootcamp-market-size-research.

[8] L. Eggleston. Coding bootcamp market size study. course report, 21 aug 2018. www.coursereport.com/reports/2018-coding-bootcamp-market-size-research. Accessed 20 January 2019.

[9] L. Eggleston. Coding bootcamp outcomes & demographics report. course report, 19 dec. 2017. www.coursereport.com/reports/coding-bootcamp-job-placement-2017.

[10] C. M. Ely. An analysis of discomfort, risktaking, sociability, and motivation in the l2 classroom. *Language Learning*, 36(1):1–25, 1986.

[11] National Center for Education Statistics. Bachelor's degrees conferred by postsecondary institutions, by field of study: Selected years, 1970-71 through 2015-16. revenues and expenditures for public elementary and secondary education: School year 2001-2002, e.d. tab. nces.ed.gov/programs/digest/d17/tables/dt17_322.10.asp.

[12] National Center for Education Statistics. Bachelor's degrees conferred to females by postsecondary institutions, by race/ethnicity and field of study: 2014-15 and 2015-16. revenues and expenditures for public elementary and secondary education: School year 2001-2002, e.d. tab. nces.ed.gov/programs/digest/d17/tables/dt17_322.50.asp. Accessed 19 November 2018.

[13] National Center for Education Statistics. Table 318.30. bachelor's, master's, and doctor's degrees conferred by postsecondary institutions, by sex of student and discipline division: 2014-15. https://nces.ed.gov/programs/digest/d16/tables/dt16_318.30.asp?current=yes. Accessed 19 November 2018.

[14] National Science Foundation. National science foundation - where discoveries begin. edited by Mark K Fiegener, NSF. www.nsf.gov/statistics/2015/nsf15326/.

[15] National Science Foundation. Women, minorities, and persons with disabilities in science and engineering. https://www.nsf.gov/statistics/2017/nsf17310/digest/fod-women/. Accessed 19 November 2018.

[16] Josephine A. Gasiewski. From gatekeeping to engagement: A multicontextual, mixed method study of student academic engagement in introductory stem courses. *Research in Higher Education*, 53(2):229–261, 2011.

[17] S. Henn. When women stopped coding. NPR. Morning Edition www.npr.org/sections/money/2014/10/21/357629765/when-women-stopped-coding Accessed 19 November 2018.

[18] Hyde, J. S., Mertz, J. E. Gender, culture, and mathematics performance. *Proceedings of the National Academy of Sciences*, 106(20):8801–8807, 2009.

[19] H. W. Marsh. Self-concept, social comparison, and ability grouping: A reply to Kulik and Kulik. *American Educational Research Journal*, 21(4):799, 1984.

[20] L. H. Summers. Remarks at NBER conference on diversifying the science engineering workforce (2005). ADVANCE Library Collection. Paper 273.

[21] K. Weisul. How Harvey Mudd College achieved gender parity in its computer science, physics, and engineering programs. Inc.com, Inc., 31 May 2017, www.inc.com/kimberly-weisul/how-harvey-mudd-college-achieved-gender-parity-computer-science-engineering-physics.html.

Course Redesign to Improve Retention: Finding the Optimal Mix of Instructional Approaches*†

Sotirios Kentros, Manish Wadhwa, Lakshmidevi Sreeramareddy, Komalpreet Kaur, Marc Ebenfield, Allan Shwedel
Computer Science Department
Salem State University
Salem, MA 01970
{skentros, mwadhwa, lsreeramareddy, kkaur, mebenfield, ashwedel}
@salemstate.edu

Abstract

Recognizing the key role of Introduction to Programming in the Computer Science curriculum we wanted to improve the course in ways that will enhance the success of students in the course and the curriculum. To this end, we launched a three year department wide effort in collaboration with the Center for Teaching Innovation. After evaluating a collection of different academic interventions, we adopted Supplemental Instruction and added four new laboratory assignments, based on Finch robots, that have a peer-learning component, inspired from Team-Based Learning. We are currently on the third year of our project. We have been evaluating the impact of our interventions in the course retention and the main learning objectives of the course. Our preliminary results show reduced D/F/W rates and improved overall student achievement in the course.

†This work was supported in part by the grant Active Engagement of Students in Whole Course Redesign. The grant was received from the Davis Educational Foundation established by Stanton and Elisabeth Davis after Mr. Davis's retirement as chairman of Shaw's Supermarkets Inc.

Introduction

In an effort to improve retention on our Introduction to Programming course, in summer 2016 we started a collaboration with the Center for Teaching Innovation. CSC 110 - Software Design and Programming I is the first programming course in the Computer Science curriculum in our department. CSC 110 is offered as a lecture and lab course, meeting six teaching hours per week (three for lecture and three for lab). The same instructor offers the lecture and lab session, which are closely coupled in the material they cover during the semester. During lecture new topics are introduced, then during lab the students practice these topics by completing lab assignments which typically include programming tasks of varying complexity. The work load of the course is heavy, typically including 10-14 lab assignments, 4-6 homework assignments, a few quizzes, a midterm and a final.

Recognizing the key role of CSC 110 in the Computer Science curriculum we want to improve the course in ways the will enhance the success of students in the course and the curriculum. The goal of our collaboration with the Center for Teaching Innovation was to explore different interventions on our CSC 110 course in order to increase retention in the class and the program, while at the same time ensuring and improving the extent to which the key objectives of the course are achieved. To be more specific, we aimed at reducing D/F/W("D", fail and withdraw) rates, ensuring at the same time that students that complete the course have further developed their critical thinking and problem solving skills, have developed an understanding of the basic principles of object-oriented design and can successfully apply them to solve programming problems using Java.

Towards this goal in Fall 2016 we examined different interventions in the three sections of CSC 110 we offered that semester. At the end of the semester we surveyed our students in order to understand what seemed to work better for them. Furthermore the faculty that offered those sections gave their feedback on their experiences. Through this process we settled on using Supplemental Instruction [10] and implementing four lab assignments with Finch robots [7] in a team-based format [12]. This tripartite redesign of the course includes key aspects of active learning that have been shown to increase retention of information by students, enhance critical thinking skills and improve student engagement [14]. In contrast to normal lab assignments for CSC 110, Finch labs are team-based activities that revolve around programming Finch, a programmable robot to do activities like moving, sensing the environment, playing music, etc. Introducing the Finch labs has multiple benefits for the students. It helps anchor the basic concepts of object-oriented programming, since interacting with the Finch robot is done by instantiating a Finch object and performing method calls on the object. This helps students understand many

of the object-oriented topics of the course, like object instantiation, method calls, passing arguments to a method, changing the state of an object, etc. Furthermore the labs facilitate peer learning in a natural way, since students team up to work on a physical object. Post-lab reflective activities are used to help students reflect on their teamwork and reinforce the gravity of the team-based component of the lab.

The semesters that followed we continued improving the design of the four Finch labs with a focus on their peer learning aspect. We surveyed students each semester while monitoring the D/F/W rates of the course. Our surveys aimed to understand the satisfaction of the students from the interventions as well as their impact on the learning objectives of the course. Our collaboration entered its third year in the Fall of 2018. During the current academic year our goal is to evaluate our finalized interventions both in terms of D/F/W rates and their effectiveness on promoting the learning objectives of the course. We are using post-lab surveys and end of semester surveys, as well as comparison of retention data to this end.

In the *background* section we are presenting related work on the different interventions we have considered over the past semesters. In the *our intervention* section we present in more detail the interventions we did on our CSC 110 course as well as the finalized interventions as they evolved during our two year collaborative experience. In the *preliminary results* section we present some results of our surveys during the first two years of our intervention as well as D/F/W rate results. We conclude with a discussion of our experience over the past two years.

Background

In our Introduction to Programming course we explored different academic interventions to decrease D/F/W rates and increase retention of students. Specifically we considered a)the introduction of Supplemental Instruction, b) Peer Learning approaches and c) use of the Finch robot. Next we present related work on the interventions we considered.

Supplemental Instruction

Supplemental instruction (SI) is an academic intervention that employs students who have successfully taken a course, to facilitate peer-learning sessions for that course. Supplemental instruction has been primarily used in courses identified as high-risk. The employed students called Peer Leaders or Supplemental Instructors, are students who have already taken the specific course, probably with the same instructor, and performed well in it.

Since Supplemental Instruction was developed at the University of Missouri-Kansas City (UMKC) in 1973 [10], there has been a wide body of research examining the impact of SI in high-risk courses. The U.S. Department of Education validated in the 90s three specific claims about the effectiveness of SI:

- Students participating in SI within the targeted high-risk courses earn higher mean final course grades than students who do not participate in SI. This finding is still true when analyses control for ethnicity and prior academic achievement.
- Despite ethnicity and prior academic achievement, students participating in SI within targeted high-risk courses succeed at a higher rate (withdraw at a lower rate and receive a lower percentage of [fail] final course grades) than those who do not participate in SI.
- Students participating in SI persist at the institution (reenroll and graduate) at higher rates than students who do not participate in SI [9].

Multiple reviews and studies have supported these claims for high-risk courses [1] and more specifically for STEM(Science, Technology, Engineering, and Mathematics) courses [13].

Introduction to Programming (CSC 110) is a high-risk course in the Computer Science curriculum. Student performance in it is often an indicator of whether students will continue in the major. Literature is strongly in favor of introducing SI in such a course and thus in our intervention we examined the addition of SI in our Introduction to Programming.

Peer Learning Approaches

As part of our intervention we explored different peer learning approaches, since these have been shown to increase academic achievement and student retention [14]. In particular we explored lightweight teams and Team-Based Learning. Lightweight teams are class teams that have little or no direct impact on the team members' grade, but aim to provide a significant component of peer teaching and peer learning in the course. As an added bonus they may facilitate the socialization of less socially adept students [6].

Team-Based Learning (TBL) is a collaborative learning and teaching strategy that gives emphasis to the structured process of group learning activities. Typically students are assigned to teams that persist for the duration of the semester and work together to apply learned concepts in order to solve problems collaboratively. TBL is structured around units of instruction, known as "modules". Each module consists of the following three-steps: preparation,

in-class readiness assurance testing, and application-focused exercise. TBL is a mature academic intervention which has been studied extensively [4]. In the context of introductory programming courses, there have been some adoption attempts [2, 5], although people seem to drift towards modified TBL interventions.

Finch Robots

Numerous research studies have shown that using robots to teach introductory programming courses increases student motivation, satisfaction and retention [11, 3, 8]. We chose to use Finch robots [7] in our CSC110 –Software Design and Programming I course. Finch robot's support for object-oriented programming was the primary reason for choosing it. Other positive features of the Finch robots are its simple hardware, ease of use for students and short learning curve for instructors.

Our Intervention

In Fall 2016 we offered 3 sections of CSC 110. Each section was offered by a different faculty member and we tried different interventions in them. In one section we added Supplemental Instruction. In the second section we introduced added Supplemental Instruction and group work. Students were placed in 5 groups of 4-5 people from the very first day of the class. They were required to sit with their group members during class and lab. Groups rotated on their class positions on a weekly bases. The following class activities were performed using these groups: a) odd numbered lab assignments (5 lab assignments, for 50% of regular lab credit), b) lab quizzes, c) peer review of lab reports and d) a post-exam group activity on the mid-term. So in this section we included a more integrated group learning experience. The primary goal of the introduction of groups was to facilitate peer learning and create a peer support structure for students. Although groups worked to a satisfactory degree, we noticed that the constant interchange between group and individual work, was problematic, since students would often prioritize individual work, over group work. The inclusion of SI proved particularly useful and was seen positively by students in both sections.

In the third section we did not add Supplemental Instruction, but we added three Finch-based lab activities. The activities were intertwined with normal lab work. The students had to individually produce solutions for the Finch lab assignments and work in groups to test them in the Finch robots the department had available.

At the end of the Fall 2016 semester we surveyed our students and evaluated their reaction to the different interventions. Based on our experiences and the results of the student surveys we decided to combine approaches and refine implementation of those that showed promise. In the Spring of 2017 we moved the group work out of the lecture and into the lab, offered an additional Finch lab (4 total) and continued the SI. Finch labs facilitate peer learning in a natural way, since students team up to work on a physical object. The team based nature of the activity also helps generate a "gravity" to the assignments, since there is also peer accountability. For the Fall 2017, Spring 2018 and now the Fall 2018 offering, we have settled on this format of the intervention, having SI and four Finch lab group activities of increasing difficulty and complexity. Progressively we have added stricter structure on the group activity elements of the Finch labs, primarily inspired by Team-Based Learning. For example, we have added post-lab reflective activities after each lab, styled after reflective questions in exam wrappers, asking the students to reflect on the topics the lab addressed and their participation and interaction in the group component of the activity.

Next we are describing the four Finch labs in a little more detail. In terms of content covered. As mentioned apart from the Finch labs, students work in 8-10 individual labs. Students use the NetBeans IDE (Integrated Development Environment) to work on the Finch Labs. In the Finch labs, there were two categories of tasks: individual and group. These tasks are clearly marked in the lab document. After completing the individual tasks, students were asked to run the code, correct any errors, discuss the tasks in the group and when they were satisfied, they were asked to show the output to the instructor or peer leader (SI).

The primary purpose of the first Finch Lab is to familiarize students with the process of writing a program for the Finch robot, compiling, executing the program and observing the behavior of Finch. The assumption for this lab is that students are not familiar with the application development environment of the Finch robot. The individual tasks include: downloading the Finch NetBeans software package, opening the Finch project, creating a new package and going through some demo java classes provided with the software package. The group tasks are focused on modifying the specific classes by providing different arguments to method calls, that result in different behavior of the Finch, resulting in changing the color of specific leds and the moving pattern of the robot.

The second Finch lab, focuses on creating classes and working with instance variables in Java using the Finch and Netbeans. In addition students learn how to use the scanner class to take user input. The individual tasks include: reading and understanding some demo java classes provided with the software

package, creating a class with three instance variables and appropriate setter and getter methods, as well as creating a tester client class. The group tasks include: answering questions regarding the java classes the students read and created. For example students need to identify number of objects and instance variables in the sample code. Then they are asked to compare their individual code, convince their peers of the better approach to solving the problem, settle on a final solution, test it and demonstrate it to the peer leader (SI) or the instructor.

The third Finch lab focuses on making the robot move based on conditions that are implemented using selection (if, if..else) and repetition (while, do..while, and for) control structures. The individual tasks include: answering questions on control structures and writing a program to make the robot display different colors, move and react when there is an obstacle. The group tasks include: comparing their individual code, convincing their peers of the better approach to solving the problem, testing it and demonstrating it to the peer leader (SI) or the instructor.

The fourth Finch lab focuses on using one dimensional arrays and random number generation. The students have to use arrays to pass multiple color values for the Finch robot to display. Then they need to go through the values initially in a sequential manner and then following a pseudorandom pattern, using random number generation. The lab has no separation of individual and group tasks. The students need to work on it as a group from the beginning.

As mentioned after each Finch lab students participate in a post-lab activity, reflecting on the topics the lab addressed and their participation and interaction in the group component of the activity. During the Finch labs, we observed a noticeable interactions, discussions and excitement amongst the majority of the students. Students generally liked the idea of having a peer leader (SI) answering the questions, while the group component of the labs helps students to learn from team members. Teams also provided an opportunity to improve communication and interpersonal skills.

Preliminary Results

Next we present some preliminary results from our experience over the past two academic years. As we discussed, the goal of our intervention was to reduce D/F/W("D", fail and withdraw) rates, while at the same time ensuring and improving the extent to which the learning objectives of the course are achieved. In order to evaluate the later, we conducted end of semester surveys to assess the comfort level of students with the various key topics covered in the course. The level of comfort of the students was evaluated on a scale with three levels: very-little, somewhat comfortable and strongly comfortable. A total of

11 key topics were considered for the survey namely: performing simple input and output, primitive types and reference types, top-down refinement, if-else selection, sentinel controlled repetition, counter-controlled repetition, switch statements, class declaration and object creation, constructors, declaring class attributes, and declaring and using static variables and methods.

The table 1 shows survey results for the 11 course topics for the Fall 2016, Spring 2017, and Spring 2018 semesters. The numbers and percentages shown under each semester indicate the students who rated their level of comfort as strong. Unfortunately during the Fall 2017 semester we used a different survey and as a result we cannot directly compare our Fall 2017 findings with the ones from the other semesters. As can be seen from table 1, comfort level increased for 10 of the 11 topics between Fall 2016 and Spring 2018.

Course Topics	F16		Sp17		Sp18	
	#	%	#	%	#	%
Static Variables and Methods	4	25.0%	7	46.7%	9	60.0%
Counter Controlled repetition	7	43.8%	10	66.7%	12	80.0%
Sentinel Controlled repetition	7	43.8%	9	60.0%	11	73.3%
Instance variables	7	43.8%	8	53.3%	10	66.7%
Class and objects	8	50.0%	10	66.7%	11	73.3%
If and If-else selection	10	62.5%	12	80.0%	13	86.7%
Constructors	7	43.8%	8	53.3%	9	60.0%
Simple Input –Output	12	75.0%	11	73.3%	14	93.3%
Switch statements	6	37.5%	10	67.7%	7	46.7%
Top-down refinement	8	50.0%	9	60.0%	9	60.0%
Primitive vs Reference Types	9	56.3%	9	60.0%	8	53.3%

Table 1: Number and Percentage of Students Who Rated Their Level of Comfort as Strong

Semester	Very Little		Somewhat		Strongly		Total#
	#	%	#	%	#	%	
F16	8	5%	83	47%	85	48.3%	176
Sp17	3	2%	59	36%	103	62.4%	165
Sp18	6	4%	46	28%	113	68.5%	165
Total	17	3%	188	37%	301	59.7%	506

Table 2: Degree of Comfort Across the 11 Key Topics Covered in the Course

Table 2 presents the degree of comfort of students across the 11 key topics covered in the course. The values are the number of responses. Fisher's exact

test was used to see if the results obtained are statistically significant or not. The p-value obtained for table 2 was 0.0002, indicating the results obtained are statistically significant.

To determine if the teaching pedagogy is improving the course success or not, final grades obtained by the students were considered. Table 3 presents the course success rate in terms of the final grades. The second column presents the D/F/Ws and the last column presents the grades in the range A through C-. Table 3 compares the grades obtained before our intervention and during our intervention. Table 3 indicates an increase in the final grades in A through C- category, and drop in the D/F/W rates.

	DFWs(%)	A through C-(%)
Pre-Intervention (AY 2015-16)	25.6	74.4
During Intervention (F16 through S18)	18.1	81.1

Table 3: Course Success Rate Based on Final Grades

Conclusions

Over the course of the past two academic years, in collaboration with the Center for Teaching Innovation, we have examined a collection of different academic interventions for our introduction to programming course. Our main goal is to reduce D/F/W rates, while ensuring and improving the extent to which the key learning objectives of the course are achieved. This has been a department wide effort, since to this point four different faculty have been taking part in our project. Through end-of-semester student surveys and inter-faculty discussions we have settled on the interventions to be adopted and during this academic year, we are concluding our effort, evaluating the finalized interventions in our course. Preliminary results indicate that our interventions have positive impact both in reducing D/F/W rates and in improving overall student achievement in the course.

References

[1] Phillip Dawson, Jacques van der Meer, Jane Skalicky, and Kym Cowley. On the effectiveness of supplemental instruction: A systematic review of supplemental instruction and peer-assisted study sessions literature between 2001 and 2010. *Review of Educational Research*, 84(4):609–639, 2014.

[2] Ashraf Elnagar and Mahir Ali. A modified team-based learning methodology for effective delivery of an introductory programming course. In *Proceedings of*

the 13th Annual Conference on Information Technology Education, SIGITE '12, pages 177–182, 2012.

[3] Juan Felipe García Sierra, Francisco J. Rodríguez Lera, Camino Fernández Llamas, and Vicente Matellán Olivera. Inside the maze: Who would find the cheese first, a robot or a mouse?: Teaching IT using robots. In *Proceedings of the First International Conference on Technological Ecosystem for Enhancing Multiculturality*, TEEM '13, pages 297–302, 2013.

[4] P. Haidet, K. Kubitz, and W. T. McCormack. Analysis of the team-based learning literature: TBL comes of age. *Journal of Excellence in College Teaching*, 25(3-4):303–333, 2014.

[5] Michael S. Kirkpatrick. Student perspectives of team-based learning in a CS course: Summary of qualitative findings. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 327–332, 2017.

[6] Celine Latulipe, N. Bruce Long, and Carlos E. Seminario. Structuring flipped classes with lightweight teams and gamification. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 392–397, 2015.

[7] Tom Lauwers and Illah Reza Nourbakhsh. Designing the Finch: Creating a robot aligned to computer science concepts. In *Proceedings of the First Symposium on Educational Applications of AI*, July 2010.

[8] Stefanie A. Markham and K. N. King. Using personal robots in CS1: Experiences, outcomes, and attitudinal influences. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '10, pages 204–208, 2010.

[9] D. C. Martin and David R. Arendale. *Supplemental Instruction: Improving first-year student success in high risk courses.* National Resource Center for The First Year Experience, 2nd edition edition, 1993.

[10] Deanna Martin and Sandra Burmeister. Supplemental instruction: An interview with deanna martin. *Journal of Developmental Education*, 20(1):22–24, 26, 1996.

[11] Monica M. McGill. Learning to program with personal robots: Influences on student motivation. *ACM Transactions on Computing Education*, 12(1):4:1–4:32, March 2012.

[12] Larry K. Michaelsen, Arletta Bauman Knight, and L. Dee Fink. *Team-Based Learning: A Transformative Use of Small Groups in College Teaching.* Geenwood Puclishing, Westport, Connecticut, 2002.

[13] Alan R. Peterfreund, Kenneth A. Rath, Samuel P. Xenos, and Frank Bayliss. The impact of supplemental instruction on students in stem courses: Results from San Francisco State University. *Journal of College Student Retention: Research, Theory & Practice*, 9(4):487–503, 2008.

[14] Michael Prince. Does active learning work? a review of the research. *Journal of Engineering Education*, 93:223–231, 2004.

Introducing Students to Computer Science and Programming using Data Analytics*

Jorge A Silveyra
Department of Mathematics and Computer Science
Muhlenberg College
Allentown, PA 18104
jreyes-silveyra@muhlenberg.edu

Abstract

Enthusiasm for learning computer science continues to grow as more non-computer science students are interested in creating computational solutions to the problems encountered in their respective fields. Many of those students, however, are anxious to take traditional computer science classes given their lack of experience with programming. To dissipate some of those fears, a course that taught students core concepts from introductory CS classes in the context of data analytics was introduced. Some of those concepts included variables, loops, conditionals, and dictionaries. In this paper, a detailed description of the course and the assignments used are delineated.

1 Introduction and Related Work

Computer Science departments are facing a trend of increased interest in their field from students from all majors [2]. This population is not always interested in a major or minor in computer science, but they wish to expand their computational skills in an effort to be more attractive to prospective employers or graduate schools. This trend has created a higher demand to offer courses that serve the interest and needs of this population. To address the increased

enthusiasm for these classes, a large number of institutions are incorporating more accessible courses into their curriculum. Some use specialized software [6], while others cover computer science topics while teaching a specific topic, such as biology [9], robotics [16], or media computation [8]. In this document, a course that concentrates on teaching computer science concepts using problems commonly solved using data analytics is presented.

There are multiple approaches to teaching data science and data analytics to majors and non-majors in introductory courses. Gil [7] proposed a course that was designed to teach data science and big data analytics while covering fundamental concepts in parallel and distributed programming. Dichev et al. [4] presented a flipped program pedagogy that introduced students to data science with no previous experience in the area. The curriculum of Dichev's class intermittently covered topics in statistics and computer science required to prepare students in data science. Finally, Anderson et al. [1] and Johnson[12] both presented courses that teach introductory programming using data analysis problems from real data sets.

The course to be presented here, CS109, is most similar to Anderson's course; however, the two courses differ in approach and goal. Most importantly, CS109 does not cover statistical models or machine learning, and the primary goal of this course is to help non-computer scientists to become comfortable and interested in learning computer science by way of solving data analysis problems. Other significant differences are that the assignments in CS109 do not include any starting code, and most of the tasks have a sequential progression, instead of being entirely different entities. This author does not intend to claim that Anderson et al. or any other approach is incorrect, contrarily, the objective is to present a different method to allow educators to potentially incorporate either program or combinations of to satisfy the needs of their audience and institution.

2 Description of the class and assignments

Currently, at Muhlenberg College, there are multiple introductory computer science classes based on different topics: Multimedia, Game Programming, etc. These courses strive to introduce major and non-major students to computer science and encourage their interest in the field. These classes usually attract a large group of students; however, the majority of the time the audience is self-selecting depending on the topic, i.e., art students take the multimedia class. The Computer Science Department is interested in expanding the breadth of the current courses, hence the need for new classes. This new course included the core introductory computer science concepts presented in the other courses but by way of a variety of data analysis problems. The problems did not require

an understanding or previous knowledge of data analytics techniques and did not involve advanced statistics.

Furthermore, students did not require any previous knowledge in computer science but were expected to gain familiarity and proficiency in the following concepts: variables, logical statements, conditional statements, loops, lists, dictionaries, files, and regular expressions. In this course, Python (Anaconda [11]) was selected given the simplicity of its syntax and its frequent use in the field of data science. Additionally, Anaconda includes most of the libraries/-modules required to complete the assignments and a simple GUI (Spyder). It is important to note that an introductory Python [5] textbook was used in this class instead of a Data Science textbook. The selection of this book was a mindful decision aimed to provide students with supplementary aid with those Python topics present in class. Students were provided with handouts for any topic not covered in the book. A summary of the topics covered in this class per week and the concepts studied are presented in table 1. The order of topics presented in the table has been adjusted based on feedback received from students.

Table 1: Topics covered in class each week

Book Chapter(s)	Topic	Concept(s) studied
1,2	Introduction to Computational Thinking	Algorithms, data science concepts, computational approaches to solving problems
2	Introduction to Python	Python basic syntax, variables, assignments, basic mathematical functions, order of operators
3,5	Functions and Conditionals	Boolean operators, truth tables, if statements, functions with/without arguments and return statements
2	Keyboard input and Scope	Scope and global variables, input from the user, casting
5,7	Lists and Random	List manipulation, basic list methods (length, etc.), basic list algorithms (add all values in a list, etc), random methods (uniform, shuffle, normal, etc)
4	Repetition and CSV files	For loops, while loops, menus, load CSV files in Python
8,handout	Strings and plots	String manipulation, basic string methods, basic string algorithms (find number of characters, etc.), pyplot plots
9	Dictionaries	Keys and values, dictionaries methods, traversal of dictionaries, difference between lists and dictionaries
9,6	Files	Load and write to text files, importance of files, append to a file
Handout	JSON	Load JSON in Python, differences between file formats, APIs, connect to an API using the requests module
Handout	Regular Expressions	Union, concatenation and closure, regular expressions in Python, Python operators for regex (+, ?, etc.)
Handout	Twitter	Connect to Twitter API, store twitter data in dictionaries

This class consisted of a lecture and a lab in which students were frequently

writing code in pairs or teams of 3 (depending on the assignment). The lectures were formatted such that there were active-learning activities, live-coding, and constant coding practice. The labs were always assigned in advance to allow students to become familiarized with the material to implement the material in a session later during which the instructor could provide instant feedback. Additionally, students had access twice a week to a workshop in which senior students helped them with their assignments. Finally, students were given homework (25% of final grade), quizzes (10% of final grade) and exams (30% of final grade), and lab assignments (35% of final grade) to measure their progress. The following subsections describe each of those course components.

2.1 Homework, Quizzes, and Exam

Most of the homework assignments consisted of problems from the textbook. These problems had 6 different types of questions: multiple choice, true or false, find the error, creation, and modification of algorithms, short answer, and programming exercises. The topic of each assignment was based on the content covered in class and did not follow the order of the book (See Table 1). In each assignment, the workload was adjusted based on other assignments (lab, quiz, etc.) pending at a given time.

Additional homework assignments included exercises with the range and zip functions as well as regular expressions and their usage in Python. These assignments included theoretical, practical, and implementation problems. Additionally, one final assignment consisted of asking students to create a challenge that could involve any topic presented during the semester. The problems were submitted to a repository such that all students could see them and answer as many as possible for extra credit. Some examples of the problems submitted are: Given a sentence or phrase, reverse the words but keep the same order (input = "my kitten has four paws", output = "ym nettik sah ruof swap"), given a dictionary, determine whether a particular argument appears as both a value and a key (if it does, remove the key).

During the course, two exams and one final exam were given. Considering that labs were completed in groups, exams intentionally included questions related to every lab assigned during the semester. These questions were used to measure every student's understanding of the central concepts presented in each lab. Usually, those questions were evaluated by presenting sample code from a lab and asking students to explain it using their own words.

2.2 Lab assignments

Lab assignments were created to provide students with the opportunity to fully explore potential applications of the concepts they were learning in class.

The first three labs were a sequence of assignments in which the concepts of variables, functions, lists, and files were the main focus. The fourth lab was an individual assignment that aimed to provide an opportunity to use and understand JSON files, APIs, and more complex data structures. The last two labs were a sequence of assignments that incorporated all concepts covered in the course as well as introduced the students to the Twitter API and usage of regular expressions. A detailed description of all assignments follows.

2.2.1 Lab 1: Functions and Python

The first lab was used to introduce students to Python, the Spyder environment, variables, and functions. Students were required to create multiple functions that calculated basic operations for two or three numerical values, such as addition, average, min, and max. Some of the functions were intentionally overloaded to help students differentiate between functions with/without parameters and void/non-void types. Students had to write a main function in which multiple calls to each function had to be made and, depending on the type of function called, they had to print the information calculated by each one appropriately. The students manually inserted (hard-coding) all input at this point.

2.2.2 Lab 2: Using lists and random values in Python

The second lab was an expansion on the first one, requiring students to update their functions to calculate operations with lists and not just variables. Additionally, students were introduced to the Python random module since the lists had to be populated with random data obtained using the uniform and normal distributions. Updating their functions was not a trivial task for them since, for example, calculating the maximum value from a list is an entirely different algorithm than obtaining the maximum value from 3 values. These differences presented a great opportunity to fully explain the functionality of loops in Python and introduce students to both Pythonic and range-based loops. Furthermore, they had the opportunity to start practicing with graphing since this lab required students to implement plots using pyplot. The assignment only presented basic instructions on how to create such plots, and they were encouraged to explore and experiment with the different options that pyplot presents (colors, labels, etc.)

This lab also required students to implement new functions that highlighted the usage of user input. One of those functions computed and displayed a message for each number in the list if they were multiples of another value. They were required to call this function with two versions, using a value entered by

the user and hard-coding the input. This particular problem allowed students to appreciate the importance of obtaining input from different sources.

2.2.3 Lab 3: Crime Investigation

Once students became familiarized with most of the fundamental material (by week 6), they were presented with labs that explicitly required data analysis techniques to be completed. The third lab significantly expanded over the previous ones by introducing real data and a real problem to solve; while allowing students to practice with while-loops, files, dictionaries, and sorting. The problem presented in this lab asked students to find multiple characteristics of the city of Philadelphia related to crime. To complete that task, the students were required to first collect crime data from the city of Philadelphia at [3] in CSV format. This data contained various details of each crime, such as the hour, location and type of crime. Once the data was gathered, students had to write their first program that included a menu. Menus were an excellent opportunity to introduce while-loops and the concept of sentinel loops. The menu ran endlessly until the user selected the exit option. Additionally, it had multiple options that call a function depending on the selected choice. The options presented in this lab are depicted in Table 2

Table 2: Concepts studied in Lab 3

Option	Concept(s) studied
Load data	Load contents of CSV file
Top 5 Crimes	Use dictionary to calculate frequencies, sort dictionary by value, extract parts of the dictionary
Plotting time of crimes	Use dictionary to calculate frequencies, aggregate data in groups by hour, plotting histogram
Best and worst streets	Use dictionary to calculate frequencies, sort dictionary by value, extract parts of the dictionary
Show crimes by date	Create a simple search function in a list
Exit	Create sentinel loops

Students were particularly excited while completing this lab since it was their first opportunity to work with real data. It also helped them understand the importance of using dictionaries, mainly when they are used as repositories of data to determine frequencies of words. Ultimately, the use of dictionaries to build histograms made it very clear that it would be challenging to complete this task with lists or any other structure covered in class before.

2.2.4 Lab 4: Star Wars JSON task

The fourth lab introduced more advanced data structures while working with JSON files. In this lab, students were required to collect information from a Star Wars API[10] and extract insights from it. To be able to connect to the API, they used the requests library [14]. This library has a method that receives an URL as input and returns an object that possesses a method which decodes the information collected to JSON format and can be stored in a dictionary. Students had to then write a program with a menu with the options depicted in Table 3.

Table 3: Concepts studied in Lab 4

Option	Concept(s) studied
Print names of all characters	Manipulate strings to change request address, JSON manipulation
Print the names of all characters and the movies in which they appear	String manipulation, nested loops, JSON manipulation
Print the names of all characters by gender	Submenu creation, JSON manipulation, plotting, dictionary of lists
Search for a specific character	String manipulation, nested loops, dictionary of dictionaries
Exit	Create sentinel loops

All the options that make requests to the API result in data stored in a dictionary for its proper manipulation. The dictionaries in this lab were more complex than in previous labs since their values can be lists and other dictionaries (dictionaries of lists and dictionaries of dictionaries). For example, the data to obtain the movies in which each character appears is a dictionary with keys *name* and *films*; the key *name* has a string with the name, but the key *films* have a list with the URLs of the movies in which that character appears. Requesting one of the URLs, it will return a different dictionary that has a key called *title* with a string value with the title of the movie. This intricate arrangement forces students to create loops that make requests to the different values inside the list providing an opportunity to practice with nested loops. Once the data was collected and manipulated, they were required to store all their data into JSON files. This requirement was made to help them understand the similarities between JSON files and dictionaries and the ease of how one can interchange from one to the other. This lab was extremely popular given the interest in the topic.

2.2.5 Lab 5: Twitter fun

In this lab, students completed a series of programs that required collecting information from the Twitter API. The first two programs started with simple tasks such as reading a small number of tweets from a specific topic, storing them in a list and JSON file, and opening the JSON file and obtain diverse information from the tweets such as username and number of favorites.

The other programs included more advanced tasks, such as asking the user for the number of tweets to read and the topic of the tweets, implementing a filter to remove curse words, limiting tweets by location, and obtaining tweets from specific users. To extract the tweets, students first had to create a Twitter apps account and obtain their 4 keys at [15]. They completed the assignment by using Twython[13] and its TwythonStreamer API. From that library, they implemented a TwythonStreamer class and used the method stream.statuses.filter with arguments follow, location and track. This lab was one of the most popular with the students since they were exceptionally excited to work with the Twitter API. This excitement overrode the discomfort of working with more complex data structures.

2.2.6 Lab 6: Regular Expressions to extract Twitter information

The final lab aimed to incorporate all topics covered in the course and adding regular expressions. Here students expanded their implementation from the previous lab to create a menu with the options presented in table 4.

Table 4: Concepts studied in Lab 6

Option	SubMenu	Concept(s) studied
Get Information	Use file	Usage of JSON files
	Get new data	Read input from user, Twython usage, JSON file creation
Extract information[1]	Number of tweets with numbers	Regular expressions
	Most popular hashtag	
	Tweets from any place in California	
Plot information	Plot number of tweets by user	Plot creation, dictionary usage, regular expressions
	Plot number of tweets by location (PA, NY or other)	
	Plot number of tweets by length ($\leq$ 50 characters, between 51 and 100, $>$ 100)	
Exit		Create sentinel loops

[1]Students created their own options. These are just examples.

Students worked in this lab for 2 weeks and displayed great enthusiasm while doing it. They appreciated the freedom presented in this lab. They were allowed to incorporate their choices for Extract Information and showed increased creativity while selecting those features. Furthermore, because they were free to add additional features, they created exciting plots and extra features.

3 Results

This class presented many opportunities for the students to encounter obstacles related to Computer Science concepts and other areas of knowledge. Most of the difficulties observed were closer to the computer science field than to the data science material. Students predominately struggled with applying computational solutions to problems they have solved non-computationally for a long time, i.e., addition or multiplication of numbers in a list. However, when students found themselves struggling, many of them visited me in my office or attended the student-led workshops. During these sessions, students were not presented with the solutions rather with aid to help them rationalize and abstract the problem they were trying to solve. It is significant to mention that the majority of the students that completed an assignment failed them on very few occasions. Most of the time they were diligent enough to correctly complete their assignments or ask for help to remove any doubts.

One issue that I found in this class was that some students would not work as hard as their peers since most labs were given in groups. These type of assignments resulted in some students obtaining high marks on their labs, but not on other assignments. I approached this problem by making two labs individual and including lab-related questions on the exams. It is important to mention that this problem is not unique to this class, but given the distinct levels of CS experience in the audience, some students felt less comfortable with the material than others. I found that implementing these solutions lead to significant improvement compared to when they were not used.

Students started visiting my office after they received grades for the first two assignments and realized where they stand in this class. Most of the students that attended office hours can succeed, but it is not always the case; especially if they are not consistent with their attendance to class/office-hours. Furthermore, I did not find any instances in which students struggled with the technologies provided, i.e., Anaconda, Twython, requests. Contrarily, since those technologies allowed them to access more exciting sources of data, they were very motivated to complete the installation steps and learn the commands necessary to use them.

In general, I found that a vast majority of students were able to complete

the work and were highly motivated by the assignment topics. I did not notice any particular differences among demographics or backgrounds. However, I did not produce a formal study, and formal research would be required to determine this accurately.

4 Conclusions

This class is in its 5th iteration and on each occasion in which I have presented it, students have described it using two words: useful and challenging. The majority of the students have never taken a computer science class and, for some, this will be their only interaction with the subject. However, many of them have mentioned, both personally and via a survey provided at the end of class, how useful this information has been in their future/current careers. For example, some students have used the concepts from the class to complete their capstone projects in biology and marketing. Furthermore, comments provided in the surveys collected at the end of the semester include: "the material we covered definitely helps me in my future career since I'm going to do analytics jobs"; "I like the course because many parts are useful in the future and I can also practice my logical skills"; "the course feels like it was a great intro to Data Science and is applicable to nearly any profession"; and "tons of hands-on experience + practical knowledge gained!". In the official evaluations, students have on average[2] mentioned that they have learned (3.97/5), increased interest in the topic (3.78/5), felt engaged to the material (4.05/5) and felt challenged by the material (4.02/5). This information is based on course evaluations, surveys, and anecdotal evidence; studies to scientifically measure any of those factors are outside the scope of this paper.

This class presents assignments that can be very intimidating to students, as evidenced by student feedback "the labs were way too challenging." However, the majority of the students have not only succeeded in completing the course but have also gained valuable skills that have helped them in future endeavors, as evidenced by such feedback as "really hard but useful." Given the challenging nature of the course, scaffolds such as frequent encouragement for the students to attend office hours and structuring the course so that labs could be completed during class time were put into place. This form of scaffolding the course was noted by the students in their survey responses, including: "very, very, very willing to help students'.

Based on the course outcomes, information obtained from the students who participated in the course, and the attainment of the course learning objectives, the course was a success. Ultimately, students received relevant information

[2]Average out of the 3 iterations presented at current institution (N=49)

that can be used in any other major and were simultaneously introduced to core computer science concepts. Should any other faculty members be interested in achieving those objectives, then it is recommended that they try this course at their respective institutions.

References

[1] Ruth E Anderson, Michael D Ernst, Robert Ordóñez, Paul Pham, and Ben Tribelhorn. A data programming CS1 course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 150–155. ACM, 2015.

[2] Tracy Camp, W Richards Adrion, Betsy Bizot, Susan Davidson, Mary Hall, Susanne Hambrusch, Ellen Walker, and Stuart Zweben. Generation CS: the growth of computer science. *ACM Inroads*, 8(2):44–50, 2017.

[3] City of Philadelphia. Crime incidents from the Philadelphia police department. `https://www.opendataphilly.org/dataset/crime-incidents`. Accessed: 01/19/2019.

[4] Christo Dichev, Darina Dicheva, Lillian Cassel, Don Goelman, and MA Posner. Preparing all students for the data-driven world. In *Proceedings of the Symposium on Computing at Minority Institutions, ADMI*, volume 346, 2016.

[5] Tony Gaddis. *Starting Out with Python*. Addison-Wesley Professional, 3rd edition, 2014.

[6] Dan Garcia. CS 10 the beauty and joy of computing. `http://cs10.org/fa16/`. Accessed: 01/19/2019.

[7] Yolanda Gil. Teaching parallelism without programming: a data science curriculum for non-CS students. In *Proceedings of the Workshop on Education for High-Performance Computing*, pages 42–48. IEEE Press, 2014.

[8] Mark Guzdial. A media computation course for non-majors. In *ACM SIGCSE Bulletin*, volume 35, pages 104–108. ACM, 2003.

[9] Steven Harold David Haddock and Casey W Dunn. *Practical computing for biologists*. Number 57: 004 HAD. Sinauer Associates Sunderland, MA, 2011.

[10] Paul Hallett. Star Wars API: SWAPI. `http://swapi.co/api/`. Accessed: 01/19/2019.

[11] Anaconda Inc. Anaconda: Python data science platform. https://www.anaconda.com/. Accessed: 01/19/2019.

[12] Jeremiah W. Johnson. Data science and computing across the curriculum. *J. Comput. Sci. Coll.*, 32(6):187–188, June 2017.

[13] Ryan Mcgrath. Twython: Python library to access Twitter data. https://github.com/ryanmcgrath/twython. Accessed: 01/19/2019.

[14] Kenneth Reitz. Requests: HTTP for humans. http://docs.python-requests.org/en/master/. Accessed: 01/19/2019.

[15] Twitter. Twitter apps. https://apps.twitter.com/. Accessed: 01/19/2019.

[16] Erica Weilemann, Philipp Brune, and Dany Meyer. Geek toys for non-techies? using robots in introductory programming courses for computer science non-majors. In *System Sciences (HICSS), 2016 49th Hawaii International Conference on*, pages 31–40. IEEE, 2016.

Low Code App Development*

Conference Workshop

Meg Fryling
Computer Science and Information Systems
Siena College, Loudonville, NY 12211
mfryling@siena.edu

The average cost of a software development project ranges from $434,000 for a small company to $2,322,000 for a large company[1]. In addition to high costs, 31.1% of projects are cancelled before completion, 52.7% will cost 89% more than their original estimates, and only 16.2% are completed on-time and on-budget[1]. Furthermore, recruiting software engineers has become increasingly difficult as demand is high and supply is low[3]. In a fast-paced world where organizations are struggling to compete, companies are looking for quicker and cheaper ways to meet their software needs. In response, no-code and low-code development platforms (LCDPs) have emerged with the promise that organizations can hire business professionals with no coding experience to build applications[2].

This workshop will provide an introduction to the Mendix App Platform, which uses a visual Model-Driven Development (MDD) approach to rapidly develop applications with little-to-no programming experience. Participants will learn how to build responsive browser, tablet, and mobile applications starting with back-end domain modeling. They will also learn about front-end development, automating business processes with microflows, and ensuring data is valid and consistent. The instructor will provide a brief overview of the platform followed by hands-on activities and lessons learned from the classroom.

References

[1] The Standish Group. Chaos report, project smart, 2014.

[2] Alison DeNisco Rayome. Low-code platforms: A cheat sheet. *TechRepublic*, 2018.

[3] Craig Torres. Demand for programmers hits full boil as U.S. job market simmers. *Bloomberg*, 2018.

Using NSFCloud Testbeds for Research

Conference Tutorial

D. Cenk Erdil
School of Computer Science & Engineering
Sacred Heart University
Fairfield, CT 06825
erdild@sacredheart.edu

In August 2014, National Science Foundation (NSF) has awarded $20 million to two separate testbeds, to support computing applications and related experiments for research, as part of the NSF CISE Research Infrastructure [2]. Called *Chameleon* [1] and *CloudLab* [3], the two testbeds have been in use since then; and have been awarded a second round of funding in September 2017. Research scientists and faculty in academic institutions, as well as staff of national labs, independent museums, libraries, professional societies directly associated with research or educational activities, and other similar institutions can utilize these testbeds.

Chameleon Cloud is highly reconfigurable experimental testbed spread over two sites, with more than 550 nodes. According to its website, it is *available to members of US Computer Science research community and its international collaborators working in the open community on cloud research.*

About 15,000 cores constitute CloudLab across three physical sites, with different focus on storage and networking, high-memory computing, and energy-efficient computing, all available on Internet2. CloudLab stack is based on Emulab [4], and allows provisioning resources at varying levels, all the way down to raw access to the hardware. It also interacts with GENI [5] infrastructure, another NSF system to support research in networks and distributed systems.

Both testbeds provide researchers typical web-based, console-style interfaces similar to industry cloud vendors, such as Amazon Web Services, Google Cloud Platform, and others. Moreover, both testbeds also allow researchers with control and visitibility to go down to *bare metal.*

More importantly, both testbeds provide *no-cost* and modern computational, data, and network infrastructure, and allow the academic research community to design, develop, and experiment with novel system design on the

cloud. A general expectation is that any research performed on these systems will result in publications in a broadly available journal or conference.

This hands-on tutorial session will provide researchers a quick refresher on cloud computing if needed, and will focus on classroom application of cloud computing tools in an academic setting; by providing simple exercises to help participants understand and create basic cloud instances on these testbeds provided by National Science Foundation.

Acknowledgements

The development of training material for this tutorial was made possible using the Chameleon testbed supported by the National Science Foundation.

Biography

Dr. Erdil has joined Sacred Heart University's School of Computer Science and Engineering in Fall 2017, and is currently the undergraduate program director of CS programs. His research interests include using cloud computing as artificial intelligence infrastructures, cyber-physical systems, computer science education, and health informatics. He is a senior member of ACM and a senior member of IEEE.

References

[1] Chameleon Project. `https://www.chameleoncloud.org`.

[2] CISE research infrastructure: Mid-scale infrastructure - NSFCloud. `https://www.nsf.gov/pubs/2013/nsf13602/nsf13602.htm`.

[3] CloudLab. `https://cloudlab.us`.

[4] emulab, a time- and space-shared platform for research, education, and development in distributed systems and networks. `https://www.emulab.net`.

[5] GENI: Global environment for network innovations. `https://www.geni.net`.

Networking and Distributed Computing in One Course*

Lightning Talk

Robert Montante
Bloomsburg University of Pennsylvania
Bloomsburg, PA 17815
bloomu.prof@gmail.com

This talk describes a course that is under development to introduce networking topics and coarse-grained parallel and distributed computing topics, in the context of a small "Beowulf" style cluster assembled from Raspberry Pi single-board computers. A cluster consists of two to four Raspberry Pis interconnected by an Ethernet switch, with one node serving as the head node, DHCP server, and interface to the system console and external systems, and remaining nodes serving as compute nodes.

The first third-to-half of the course will cover networking topics. The OSI and TCP/IP networking models are used to describe the communications issue between the nodes. Ethernet topics include 1000BaseT versus 100BaseT characteristics and their impact on Raspberry Pi communications. IPv4 topics include subnetting, DHCP configuration, and NAT; and a brief discussion of routing as it affects connection to external networks. Application-level topics include client server and peer-to-peer communications, with SSH and netcat as examples.

The remainder of the course introduces parallel and distributed computing. Topics will include parallel architectures, shared- and distributed-memory, message-passing systems, synchronization issues, and GPUs[3]. Theory will be covered in enough detail to motivate some example applications. Programming activities will focus on SIMD and MPI-based applications.

A motivation for this course is that ABET accreditation is expected to include networking and parallel computing topics in the CAC Version 2.0 criteria[1][2], starting with the 2019-2020 accreditation cycle. The course is anticipated as a required course, possibly replacing a current elective networks course.

Dr. Montante teaches courses in C and Assembly, Computer Organization, Networks, and Python in the Computer Science and Digital Forensics majors at Bloomsburg University. His professional interests include computer systems and networks. He serves as the adviser for the program's student ACM chapter, the programming-contest team coach, and the faculty director for the university's Freshman Learning Community in Digital Sciences. He has also been a "Python evangelist" on the campus.

References

[1] ABET. CAC criteria for 2018-2019. http://www.abet.org/wp-content/uploads/2018/02/C001-18-19-CAC-Criteria-Version-2.0-updated-02-12-18.pdf.

[2] ABET. Criteria for accrediting computing programs. http://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2017-2018/#3.

[3] NSF/IEEE-TCPP. Curriculum initiative on parallel and distributed computing. https://grid.cs.gsu.edu/~tcpp/curriculum/?q=system/files/NSF-TCPP-curriculum-version1.pdf.

Creating Opportunities in Technology for Young Adults With Autism*

Lightning Talk

Darlene Bowman

This lightning talk introduces you to several NYC high school students with Autism in District 75 who participated in filming the documentary "Great Kills HS –Autism At Work" [2]. Students speak passionately about their strengths, likes, hopes and dreams for the future. You will be amazed and moved by these young adults. It is the hope of the author that you will be compelled to help shape a world that supports these students in their quest to continue CS education and other vocational training upon graduation. Advocates, universities, corporations, teachers, parents and students must collaborate to open doors - creating opportunities in technology for young adults on the spectrum.

District 75 comprises self-contained schools with highly specialized instructional support for students with significant challenges including cognitive delays, emotional disabilities, sensory impairments, physical/multiple disabilities, as well as Autism Spectrum Disorders [1].

Teachers share in the development and dreams of our students and their parents. Students spend seven years (from ages 14 –21) preparing for life after high school. When they graduate, there are scant opportunities awaiting them, especially in computer science. One statistic shows nearly 42% of young adults on the spectrum never worked for pay during their early 20's [4].

We must decide as a society to focus on the need for employment opportunities for students with Autism and other disabilities after high school. The author has founded an innovative program, Ausome-Tech Industries, designed to support the transition of HS students on the spectrum into adulthood. Drexel Autism Institutes' definition of "a successful transition" means a person has a role to play in society, through employment or pursuit of further education [3]. This talk is designed to encourage others to actively engage in creating supportive employment opportunities for people on the spectrum.

Speaker Bio

The beauty of working with Special Needs students is that you never know when their extraordinary talents will shine.

— Darlene Bowman

Darlene Bowman has been working with students with disabilities for close to 20 years. She officially began her teaching career with the New York City Teaching Fellows program in 2004.

Darlene earned her undergrad and graduate degrees at CUNY - College of Staten Island where she is currently an Adjunct Writing Professor in the English Department.

By day, Mrs. Bowman teaches CS4ALL Software Engineering Program (SEP) to high school students with Autism. She loves integrating Science, Technology, Engineering, Art and Math into all academic subjects.

Darlene's students embody her pioneering spirit and were among the first in New York City's District 75 to participate in the "Hour Of Code" in 2013 - and every year since.

Another "first" came when she earned a $75K Software Engineering Program Technology Grant through CS4ALL, enabling her school to start the first Software Engineering Program in a self contained District 75 High School for students with Autism.

Her students have gone on to participate in Game Design challenges throughout New York City. The Great Kills HS has hosted Hackathon's with general education students from New Dorp - a local high school that also teaches the Software Engineering program. Darlene and her students even have a Web Design Hackathon currently featured on the CS4ALL Blueprint for all the world to see. (`https://blueprint.cs4all.nyc/resources/34/`)

The documentary - "Autism At Work" –was filmed at the Great Kills High School and features Darlene's students speaking about their aspirations for adulthood vocations. Two of the students featured in the film took a crash course in Final Cut Pro to assist with editing.

It is Darlene's hope to provide paid internship opportunities at top tech firms to her students through her startup, Ausome-Tech Industries, once they graduate at the age of 21. She also encourages people in all industries to open their doors to students like the wonderful young people in the film.

Darlene and her husband James have 4 young adult children, who grew up loving science, technology and coding!

References

[1] NYC Department Of Education District 75. www.schools.nyc.gov/special-education/schoolsettings.

[2] Darlene Bowman. Great Kills HS Autism At Work. https://youtu.be/MANpcuKpNlA.

[3] Mental Health Weekly Digest. Reports on autism from Drexel university provide new insights (parents' and young adults' perspectives on transition outcomes for young adults with autism). http://link.galegroup.com.proxy.library.csi.cuny.edu/apps/doc/A527109015/AONE?u=cuny_statenisle&sid=AONE&xid=d9276892.

[4] Anne M. Roux, Paul T. Shattuck, Jessica E. Rast, Julianna A. Rava, and Kristy A. Anderson. National autism indicators report: Transition into young adulthood. *A.J. Drexel Autism Institute*, 2015.

Partnership with Industry Professionals in the Design of Computer Information Science Course*

Lightning Talk

Nina Dini[1], *Elham Mahdavy*[2]
[1]*Department of Mathematics, Physics and Computer Science*
Springfield College, Springfield, MA 01109
ndini@springfield.edu
[2]*Product Manager ISO New England,*
1 Sullivan Rd, Holyoke, MA 01040
emahdavy95@gmail.com

Industry–College partnerships are increasingly being recognized as a new way of providing applied education opportunities for students majoring in computer science. A systems seminar, a capstone course in our computer science department, is designed for computer science majors to develop a database system for managing a small business' operations and data. They employ skills and knowledge from systems analysis, database design and management, and the visual studio programming courses.

Teams of students are assigned real world scenarios, and they use the software development life cycle (SDLC) to track the stages of the development process. Applying their Visual Studio programming skills, the teams construct a frontend interface for their database system. Teams design and implement the database system by using the Erwin data modeler, and use the Oracle Database XE 11.2 to create tables, enter data, and run SQL queries.

Teams collaborate closely with industry professionals, who help them use IT project management principles to guide the development process of the database system. The students prepare a portfolio for the project that includes writing a statement of work, defining the product and the scope of work, preparing a work breakdown structure (WBS) along with time management to deliver the product on time.

In addition, industry professionals are periodically invited during the semester to share their experience and knowledge in software development and assist the students with time management in relation to the SDLC process.

The teams present and submit their final software product and project documentation, in both oral and written form, at the end of the semester. Teams are evaluated when they present their database system to an assembly of faculty, industry professionals and students.

A Web Based Block Language for Modeling Dynamic Data Structure Algorithms*

Lightning Talk

Robert A. Ravenscroft Jr.
Department of Mathematics and Computer Science
Rhode Island College
Providence, Rhode Island 02908
rravenscroft@ric.edu

The web browser based Dynamic Data Structure (DDS) modeling program is being developed to aid in the teaching of dynamic data structures such as linked lists and trees. Because it is a browser based application, it can be widely distributed without the need for installing software. DDS allows the user build and manipulate graphical models of dynamic data structures by dragging and manipulating them on the screen, eliminating the need for the draw-erase-redraw cycle of static media such as whiteboards. Currently, versions for linked lists and binary trees are available while versions for doubly linked lists and heaps are in development.

While DDS was originally conceived as a tool to aid in classroom instruction, current efforts involve the development of features for use as a hands-on student learning tool. A drag-and-drop block language has been implemented for the DDS linked list tool and will be added to the binary tree tool. Users can drag the control structures into their algorithm, then type in the expressions used within these control structures. It allows a student to build semantically correct algorithms that can manipulate the linked lists that they modeled with DDS. As the code executes, or the user single steps through it, they can observe how each statement affects the linked list.

This block language feature is intended to provide a platform for hands-on student learning. Students can be provided pre-built scenarios in DDS on which they can experiment with manipulating a linked list to achieve a specified goal. Once they gain insight into how they should manipulate the

linked list, they can develop and implement an algorithm that achieves the specified goal. Algorithms can be saved with the DDS model, allowing the project to be submitted for evaluation.

This talk will present the DDS block language feature, discuss its usage in teaching computer science, and outline future development plans.

Biography

Dr. Ravenscroft earned his PhD in Computer Science at Brown University. He did a post-doc with the Symbolic Computation Group at Waterloo University. He has taught at University of Delaware, University of Rhode Island, and Millersville University. His interests include developing simple web based tools for teaching computer science and mathematics.

Supporting Material and Resources. The DDS tools and supporting material are available online at `http://dsviewer.org/dds/homepage`. Current versions of the DDS tools and tutorial material tools will be available online at the time of the talk. A short handout or set of slides will be made available to attendees at the talk.

Curriculum design for 'Introduction to Data Informatics' (a New Data-related Undergraduate Course at USC)*

Lightning Talk

Saty Raghavachary
University of Southern California
Los Angeles, CA 90007
saty@usc.edu

Abstract

Starting Fall 2018, the CS department at USC has begun offering INF250: Introduction to Data Informatics, a brand new undergraduate course. The course is offered as a core part of a new Informatics (BA) 'CS+X' program, where X would constitute a suitable specialization area in which Informatics principles can be meaningfully applied (students currently enrolled in the course, come from areas such as Psychology, Cognitive Science, International Relations, Social Science etc.). This new CS+X program, and the foundational core course, have been tailored to meet the growing demand for data scientists and engineers in a variety of areas, and correspondingly, increased request from undergraduate students for such a program (it is customary for universities to offer a data science track at the Master's level, not Bachelor's).

This lightning talk will introduce the design principles behind the new core Informatics course, the syllabus, assignments, and a brief status report on how the course is progressing. The audience will be solicited for feedback about the course design and content, and for possible future collaboration. The talk will benefit attendees who might be planning to offer a similar course, and others who might want to incorporate a part of it (a few lecture topics and assignments, for example) into an existing class, eg. on databases.

Description

The curriculum design for the course is based on a two-part scheme: a solid overview of data science principles and theory (including statistics, Big Data, data mining, machine learning and visualization), and a thorough list of application areas that span the gamut of human activity, including health/medicine, societal needs, environment, agriculture, manufacturing, entertainment, commerce, communication. Students are able to learn exactly how theory translates to practice, by being introduced to case studies from these widely different application areas. Orthogonal to the technical aspects such as data mining algorithms, the syllabus also includes topics on data governance, ethics of empowering data-driven algorithms to make decisions, data security and data privacy. As for assignments, they cover key areas of the data science pipeline, including exploratory data analysis (EDA), data mining, machine learning, and information visualization. Also, in keeping with the industry trend, the course is not heavy on coding from first principles; instead, students use existing tools (such as WEKA and Tableau) where appropriate, or use Jupyter notebooks (running locally, or on a cloud environment such as Google's Colab) to study, tweak and learn from the richly annotated code contained in the notebooks' cells.

Biography

Dr. Saty Raghavachary teaches undergraduate and graduate courses on programming, databases, data science and computer graphics, in the Computer Science department at the University of Southern California (USC). Prior to joining USC full-time, Saty worked at Autodesk for 2 years, and at DreamWorks Feature Animation for 16 years.

Interdisciplinary Programs*

Panel Discussion

Yana Kortsarts[1], Adam Fischbach[1], William J. Joel[2], Ting Liu[3]
[1]Computer Science Department
Widener University, Chester, PA 19013
{ykortsarts,jafischbach}@widener.edu
[2]Western Connecticut State University, Danbury, CT 06810
joelw@wcsu.edu
[3]Computer Science Department
Siena College, Loudonville, NY 12211
tliu@siena.edu

1 Summary

Computer science is a rapidly changing discipline and our goal is to provide opportunities for students to understand the complexity of the modern world through interdisciplinary learning, explore and make connections to other fields, and integrate various perspectives to allow for interdisciplinary problem solving. Integrating interdisciplinary thinking into the computer science curriculum is a challenging but rewarding task that benefits faculty and students. The panel will present successful experiences developing and managing various interdisciplinary programs. Active audience participation is encouraged. The panel will provide an opportunity for attendees to share their views and to exchange knowledge during a question-and-answer period that will follow individual presentations.

2 Yana Kortsarts And Adam Fischbach

We present our experience developing and managing interdisciplinary programs in computer information systems, computer forensics and digital media informatics –the results of successful collaboration with social science and business

faculty. The computer information systems major combines courses in computer science with courses in the School of Business Administration. Students learn about software development, database design, business management, and management information systems. The program provides students with a less theoretical and more applied curriculum, which gives them the foundation to design, build, and maintain computer information systems. The computer forensics minor is an interdisciplinary program that integrates criminal justice and computer science and combines both theoretical concepts and practical skills to prepare students for a career in the area of information security and digital forensics. The digital media informatics major is an interdisciplinary program run jointly by the computer science and communication studies departments. The program provides both broad and targeted perspectives on the field of informatics and helps students develop unique skills that can be adapted to the rapidly changing computer and media environment through four specialized concentrations: (1) audio-visual, (2) graphics, mobile, & web development, (3) gaming & artificial intelligence, and (4) digital writing. We describe the various stages in developing the interdisciplinary programs including an analysis of competitive academic programs, evaluation of current resources, qualifications and faculty considerations, the process of developing the program objectives and learning outcomes, and assessment strategies. We focus on common issues that arose during the development process such as the challenge of designing balanced curricula for interdisciplinary programs, the need for designing new courses and renovating existing courses. We also discuss the anticipated costs of the programs, required resources, recruitment strategies, and the administrative approval mechanism.

3 William J. Joel

At WCSU, the departments of Art, Communications, and Computer Science, recently established a new, interdisciplinary major: Digital & Interactive Media Arts (DIMA). Unlike other similar degree programs, at other institutions, DIMA is intended to be an equal blend of all three departments, and as such is governed by a Steering Committee with representatives from the three departments. Maintaining such a balance has necessitated such choices as ensuring that the level to which each discipline is represented in the DIMA core requirements is of equal rigor. Our CS department has five minor, four of which include, or will include, courses from other disciplines: Security, Digital Media, Informatics, Web Development. Our Graphics & Interactive Techniques Research Group (GITRG) has drawn students from Art, Music, DIMA, and Math, as well as CS. GITRG strives to engage students from as many disciplines as possible in order to foster novel solutions to research problems.

4 Ting Liu

At Siena College, we have a new Data Science Program supported by multiple departments, such as Computer Science, Math, Physics, Environmental, etc. since we believe that Data Science is an interdisciplinary science and requires contributions from different departments. The core courses, including data analysis, mathematical methods, and machine learning, of our Data Science program provides a solid theoretical foundation for students. In addition, Data Science students need pick 18 credits track courses that can be focused on one area, such as social science, business, biology, etc to practice what's been learned from core courses. We also collaborate with Business school for teaching computer related courses, such as Management Information System and database design and application for Business, for their students. Coordinated by Center for Undergraduate Research and Creative Activity (CURCA), professors from Computer Science department, Physics Department, and Business school worked together with our community partner, CARES, Inc (an organization administrate homeless shelters from 13 counties around Albany County in New York state) to help improve the quality of homeless data and build new tools for data analysis.

5 Biographies

Yana Kortsarts is a Professor of Computer Science and Chair of the Digital Media Informatics program at Widener University and has been actively involved in developing computer a forensics minor and managing a digital media informatics major.
Adam Fischbach is an Associate Professor of Computer Science and Chair of Computer Science Department at Widener University and has been actively involved in the computer science department's interdisciplinary efforts.
William J. Joel is a Professor of Computer Science in the Computer Science Department at Western Connecticut State University, and serves on the Steering Committee for the interdisciplinary degree, Digital & Interactive Media Arts, as well as being Director for the school's interdisciplinary Graphics & Interactive Techniques Research Group.
Ting Liu is an Assistant Professor of Computer Science and Steering Committee of Data Science program at Siena college. Dr. Liu has been actively involved in teaching interdisciplinary courses, developing interdisciplinary program, and collaborating with other department faculties for interdisciplinary research engaging with community partners.

A Survey of Several Advanced Mathematical Concepts Implemented in Students' Computer Science Projects*

Faculty Poster

Vladimir V. Riabov
Department of Mathematics and Computer Science
Rivier University, Nashua, NH 03060
vriabov@rivier.edu

Mathematics has a vital role in the development of computer science, electronic systems, and numerous practical applications. The objective of this poster is to review several advanced mathematical concepts and methods (modular arithmetic; Galois fields; graph theory; singular differential equations; strange attractors; fuzzy logic, and projective geometry) that contribute into the development of applications in cryptography, numerical methods, code complexity reduction, atmospheric dynamics, expert systems, computational visualization, and other areas.

The mathematical concepts, algorithms, and codes are examined by undergraduate and graduate students in various courses taught by the author. These concepts have paved the roads for students' research projects on various applications. Each student works on a selected project analyzing algorithms, creating computer codes (in Python, MATLAB, C/C++ or Java), running them at various parameters, comparing numerical results with known data, and presenting the findings to classmates and the research community. Many students published project summaries in the Rivier Academic Journal [1] and conference proceedings available from the web [2].

The opinions on why computer science students need general knowledge of mathematical concepts have been widely discussed in academia [3]. Several scholars [8] even recommended long lists of mathematical methods and formulas (ironically named as "Computer Science Cheat Sheets") that every computer science student should be familiar with. These "Cheat Sheets" cover mostly basic mathematical concepts (e.g., series, function-value order definitions, permutations, combinations, identities, recurrences, geometry, matrices, special

functions, calculus of derivatives and integrals, Cramer's rule, etc.). Only a few complex math methods are mentioned there [8]: brief reviews of the Number Theory, Graph Theory, and the Master Method for algorithm analyses, but the advanced concepts (e.g., modular arithmetic and Galois fields; fuzzy logic; strange attractors; pattern recognition, etc.) are not included in those reviews.

The theory of numbers plays probably a unique role in the theoretical computer science and various applications. Traditionally, the related topics (e.g., numerical systems, the Fundamental Theorem of Arithmetic, primes, and coprimes) are covered in the Discrete Mathematics course. In our pedagogical practice, the more advanced topics (modular arithmetic, abstract groups, rings, integer domains, and fields) are covered in the Computer Security elective course [7], due to the fact that modern encryption methods utilize the modular arithmetic and Galois field properties framed with the Fermat's Little Theorem and properties of Euler's totient function. Java Applets [7] have been found as an effective tool to introduce these advanced topics and cryptographically-secure message digest algorithms. Many students made overviews [1, 2] on the role of number theory in modern cryptography, coding theory, Advanced Encryption Standard, Remote Authentication Dial-In User Service protocol, and Wi-Fi security issues.

In the Software Quality Assurance course, the structured testing methodology [5] and graph-based metrics [6] have been reviewed by students and applied for studying the C-code complexity and estimating the number of possible errors and the required tests for various networking systems. Comparing different code releases, it is found that the reduction of the code complexity leads to significant reduction of errors and maintainability efforts [6].

Many students selected challenging topics for their research projects in various computer science courses. Here we only make overviews of a few outstanding students' projects that have been performed using the mentioned-above advanced mathematical concepts discussed in class.

David Snogles developed the Personal Encrypted Talk system for his final capstone project [1, 2]. Its primary goal was to secure Instant Messaging communications between two parties on the Internet. Secondary objectives were Java Cryptography Architecture research and the practical experience gained by the student in the development of a scalable Java-based GUI.

Robert Marceau studied Hoare's quicksort algorithm that has become a popular sorting algorithm due to the average performance of (nlog2n), limited use of extra storage (typically (log2n) recursive calls), and better performance on average compared to heapsort algorithm. The major drawback in the quicksort algorithm is the (n2) worse-case performance, which is exhibited for some initial permutations. Robert studied this performance and offered modifications to minimize the probability that the worst-case performance will

be exhibited [1, 2].

Maxim Sukharev-Chuyan studied a simple basic model of chaotic behavior in atmospheric layers known as the Lorenz system [4]. He developed a Java code for an animation of the water-wheel model of the strange attractors for the Lorenz system [1, 2]. The visualized simulations demonstrate chaotic behavior of the numerical solution of the Lorenz system of nonlinear ordinary differential equations [4].

Kevin Gill developed the Living Mars image project [1, 4] that included topics related to computer graphics, software development, and planetary science. The purpose of the project was to create a visualization of the planet Mars as could look with a living biosphere. The algorithms and methods used in generating shadows on digital elevation models were developed in his previous study [1, 2]. These include formulas that are common in computer graphics applications and are often provided by specific frameworks (i.e., OpenGL). The basics of model rendering included the structure of the source data and the interpolation of hypsometric-bathymetric tint colors. The primary algorithm is based on the calculation of shadows using ray tracing. These methods utilized the code [1, 2] from the Kevin's jDem846 open source project.

In the course evaluations, students stated that they became deeply engaged in course activities through examining the challenging problems related to the applications of the advanced mathematical concepts.

References

[1] The rivier academic journal archive. `https://www2.rivier.edu/faculty/vriabov/students_publicat.htm`.

[2] Rivier students' articles. `https://www2.rivier.edu/faculty/vriabov/students_publicat.htm`.

[3] T. Beaubouef. Why computer science students need math. *SIGCSE Bulletin*, 34(4).

[4] E. N. Lorenz. *The Essence of Chaos*. University of Washington Press, 1993.

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

[6] Vladimir V. Riabov. Methodologies and tools for the software quality assurance course. *Journal of Computing Sciences in Colleges*, 26(6):86–92, June 2011.

[7] Vladimir V. Riabov and Bryan J. Higgs. Running a computer security course: Challenges, tools, and projects: Poster session. *Journal of Computing Sciences in Colleges*, 25(6):245–247, 2010.

[8] S. Seiden. Theoretical computer science cheat sheet. *ACM SIGACT News*, 27(4).

Lessons Learned from Integrating POGIL into a CS1 Course*

Faculty Poster

Michael Jonas
Department of Applied Engineering and Sciences
University of New Hampshire, Manchester, NH 03101
michael.jonas@unh.edu

This work looks at lessons learned from transforming an existing introductory programming course to using the Process Oriented Guided Inquiry Learning (POGIL) model [1]. POGIL uses collaborative group learning that is organized into a set of labs with a rigorous structure in both student interaction and the material being worked on. The computing discipline is relatively new to POGIL and work is ongoing in developing lab material for various levels of course work. The most robust effort, to this point, has focused on introductory level material. Transforming that material to fit into the existing structure of a course, i.e. language, tools, platforms, and practices, is the first of many hurdles one needs to overcome.

In POGIL, the role of the instructor is transformed from controlling flow of information between teacher and students to facilitating information exchange among students. It can be challenging to change roles by stepping back to allow students to discover answers as they work out well structured problems. As facilitator it is important to balance time spent engaging students with enabling them to struggle along in a productive manner. Devising sound principles in promoting collaborative learning will drive student experience. From preventing strong personalities in hijacking a group, to encouraging gender equality, to promoting participation of reluctant students, all are challenges to overcome in order to create a workable environment for POGIL to flourish.

A critical element in developing a productive POGIL lab environment is creating good student working groups. Different techniques were applied with each offering their own benefits and shortcomings. Approaches governed how groups were constructed, either requiring directed student assignment by the instructor or allowing students to self-choose their preferred group. Both have

their pitfalls as either incompatible personalities could be forced to work together or an imbalance of strengths could be distributed among groups. Additionally, student roles in groups are also integral to POGIL and insuring that each role is fulfilled and that each student takes on every role through the course of the semester is important. Though seemingly easy to control, random events such as student absences or withdrawal from the course can not only change group dynamics but may require reshuffling or combining groups; doing so without adding too much disruption requires thought.

The most challenging component of successfully integrating POGIL into an existing introductory programming course is to insure student buy-in. One characteristic of POGIL that is universal is that the experience can be exhausting to both student and teacher. Activities are timed, roles need to be accepted and followed, and each activity builds on the previous. This leads to potential stress for students who may feel rushed, out of their element, or even bullied by strong personalities within their group. However, with careful facilitation, these potential pitfalls can be turned into positive experiences were exhaustion comes from a fun learning environment. This work aims to share these experiences so that anyone looking to integrate POGIL into their own curriculum can gain some useful insights.

References

[1] POGIL: 2019. `.https://www.pogil.org` Accessed: 2019-01-11.

DDS: A Web Based Tool for Modeling Dynamic Data Structures*

Faculty Poster

Robert A. Ravenscroft, Jr.
Department of Mathematics and Computer Science
Rhode Island College
Providence, Rhode Island 02908
rravenscroft@ric.edu

Dynamic Data Structures (DDS) is an HTML5 and JavaScript web browser application that allows the user to interactively model dynamic data structures such as linked lists and binary trees. DDS was conceived as a modeling tool to be used when teaching dynamic data structures, eliminating the need to model them on static media such as white boards. However, DDS is not an algorithm visualization system. Rather, it provides a platform for the user to build and manipulate a graphical model of a data structure in a manner of their choosing. Any action taken by the user corresponds to the execution of one statement in an algorithm. Only manipulations that are semantically valid in a language such as Java are allowed. Actions are implemented by dragging with a mouse (reference assignment, node layout) or by clicking buttons (node creation, garbage collection).

Tools such as DDS are rare. Even web based algorithm visualization tools are still uncommon. There are two systems are of interest. Data Structure Visualizations [1] is a web based package of visualization programs. These programs are typically command driven. The user enters a data value, presses a button, and watches how the operation modifies the data structure. Unlike DDS, they do not allow the user to directly interact with the data structure in the visualization. JSAV [2][3] is a JavaScript framework for developing visualizations. It has been used to develop student exercises that permit manipulation of data models as part of the solution. Some of these assignments provide user interaction with a data structure model that is similar to that provided by DDS. However, they are not general purpose data structure modeling tools.

The development of the DDS tool is an ongoing research project. The current versions model linked lists and binary trees and were used in a data structures course to evaluate their effectiveness as a classroom teaching tool [5][4]. DDS lived up to expectations in the classroom. Pre-built scenarios avoided the need to use class time to draw them on the board. With DDS, there were no concerns with introducing errors when drawing or manipulating a model. The typical white board draw-erase-redraw cycle was eliminated, and classroom time was not taken up with frequent attempts to correctly draw updates to the model as the data structure changed. The data files for the class examples were posted on the course web site, allowing student access to the examples after they left class.

Based on classroom experience and feedback from conference presentations, several improvements and enhancements are underway. Garbage collection of unreachable nodes has been improved. Layout of lists has been improved. A stack mechanism is being implanted to model recursion. The option to use either integer keys or text keys in nodes is in development. A snapshot feature has been added to the system that allows the user to capture and save an album of snapshots of the model at various points in time. This feature was motivated by the desire to use DDS as an active learning tool. The album will allow a student to maintain a history of manipulations that were used to solve a problem and then to submit the project file that contains the album. While DDS was not intended to be an algorithm visualization system, a simple block language has been prototyped for the linked list version to allow algorithm visualization. The block language will provide new potential applications of DDS as a tool for active learning.

Several additional features are under consideration and are likely to be implemented in one of the next development cycles. Automatic layout of binary trees needs improvement. An option to store key-value pairs in a node is needed to model the implementation of the dictionary abstract data type. A tabbed interface will allow multiple models to be stored within a single project. An animation feature that allows the user to record a sequence of manipulations is also under consideration. This would allow instructors to pre-record their demos. Or, instructors could record class room demos and post them online. Likewise, students could use the animation feature to record their solutions to data structure problems and submit them online. Finally, a version of DDS for doubly linked lists is in design.

This poster will present the current state of the DDS system and the benefits of its usage in the classroom. It will also present information about current and anticipated future work on the system. DDS is available online at http://dsviewer.org/dds/homepage and information will be available on its usage. New users are welcome to try it out and use it for teaching and learning.

It is hoped that this poster will help boost awareness of DDS and identify new collaborators to help evaluate its effectiveness as a teaching and learning tool. Suggestions for new capabilities as well as for improvements to existing capabilities are also welcome.

References

[1] D. Galles. Data structure visualizations. `www.cs.usfca.edu/~galles/visualization/`, retrieved 9 November, 2017.

[2] Karavirta, V., Shaffer, C. A. JSAV: the JavaScript algorithm visualization library. *Proceedings of the 18th Annual Conference on Innovation and Technology in Computer Science Education*, pages 159–164, 2013.

[3] Karavirta, V., Shaffer, C. A. Creating engaging online learning material with the JSAV JavaScript algorithm visualization. *IEEE Transactions on Learning Technologies*, 9(2):171–183, 2016.

[4] Robert A. Ravenscroft, Jr. Dynamic data structures, a web based tool for teaching linked lists and binary trees. *Journal of Computing Sciences in Colleges*, 33(6):97–106, 2018.

[5] Robert Ravenscroft. An HTML5 browser application for modeling and teaching linked lists: (lightning talk, abstract only). *SIGCSE '18: Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, page 1106.

The Use Of Virtual Desktop Infrastructures In A Graduate Computer Science Curriculum*

Faculty Poster

David Pitts, Vladimir V. Riabov
Department of Mathematics and Computer Science
Rivier University
Nashua, NH 03060-5086
{dpitts, vriabov}@rivier.edu

This poster will present the use of and experience with virtual desktop infrastructure (VDI) [2] in the Masters in Computer Science program at Rivier University. Many of the advantages of VDI in a university setting have been described in [3]. This poster will focus on our experience with VDI for Rivier's Computer Science graduate program. In addition to reducing hardware costs and system administration loads, VDIs also provide convenient means for both students and faculty to access software tools and applications from both home and the university. For the Computer Science program at Rivier University, the combination of VDIs and open source software gives us a good way to provide the students with the necessary tools, both in and out of class, and to perform the work that instructors expect. VDIs provide a convenient way to create the specialized Computer Science desktops (CSDs) only for the computer science students (students in other programs use the standard, "vanilla" desktops).

Rivier University started using VDI about six years ago with an in-house IT department and has continued the VDI utilization after adopting an outsourced IT model. The use of VDI became particularly important during an explosion of the graduate program due to a large influx of international students: the program went from around 40 students to almost 600 full-time students at its peak. The accommodation of VDI allowed us to meet the demands of this growth gracefully, quickly adding the necessary computer-based classrooms needed for this larger population of students.

Rivier University's VDI consists of thin-clients (we use Dell Wyse thin-clients) that connect to the Rivier's VDI server. There is a separate server

for on-campus use versus remote use. The connection to the VDI is managed through the normal Rivier University credentials. Once connected to the VDI, the user is offered a set of virtual desktop pools. For example, Rivier provides University Desktops, Computer Science Desktops, and Faculty/Staff Desktop pools. Selecting of the appropriate pool creates a new virtual desktop for the user.

The Computer Science and University desktops are not persistent [1]. Each time a student logs into one of the computer science desktops, the desktop is created a new VDI. This feature provides some protection against student's misadventures and numerous malware dangers (e.g., viruses and worms) that are ever present. However, at Rivier, faculty are provided with persistent desktop, allowing the replacement of desktop computers in faculty offices with a small Dell Wyse thin client.

Another important benefit of VDI is that, through the use of VMWare's Horizon View client [4], the Computer Science desktops are available not only through the Dell Wyse thin clients, but also through the personal devices (e.g., laptops, desktops, and even iPhones), allowing access to the computer science applications at home as well as on campus. Faculty can work on demonstrations and in-class activities at home on exactly the same environment that students will use during class. Online students, after installing the Horizon View client, have complete access to the Computer Science desktops. Further, for faculty, their persistent desktops replace the VPN that was in use at Rivier University, since the faculty member can connect to her/his persistent desktop from home and immediately have access to the Rivier network environment (e.g., file folders, printers, etc.).

Another benefit of VDI that we have not yet pursued is the support of virtual desktops with different operating systems installed, such as Linux or FreeBSD. Rather than devoting additional hardware resources or formatting existing systems as dual-boot systems, a new pool of virtual desktops may be created.

While there are many benefits to VDI, deployment of a VDI and set of desktops requires some planning. Applications for a virtual desktop are configured as layers [5], which allow the VDI administrators to install application with the necessary dependencies and add them to the OS layer for a specific virtual desktop pool. In the case of Rivier, the "vanilla" University Desktop pool uses a small number of application layers, while the number of layers required for the Computer Science Desktops is about 30 layers. The more layers required by a virtual desktop, the longer it takes to create. When classes of 20-plus students attempted to connect to the Computer Science Desktops simultaneously for a class, the demand on the VDI system can be quite significant. Further, instructors and IT administrators must carefully plan for licenses required to

simultaneously run the large number of virtual desktops.

In conclusion, VDI has been of great benefit to Rivier University, particularly in support of the sudden spike of the number of students in the computer science programs.

References

[1] N. Maloney. Persistent vs. non-persistent VDI. http://blog.accessitautomation.com/persistent-vs.-non-persistent-vdi.

[2] Steele C. Rouse, M. and J. Madden. Desktop virtualization. https://searchvirtualdesktop.techtarget.com/definition/desktop-virtualization.

[3] S. A. Vieira. Why virtual desktop at CCRI? finding sustainability for desktop support. *SIGUCCS '12 Proceedings of the 40th Annual ACM SIGUCCS Conference on User Services*, pages 81–86, 2012.

[4] VMWare. Horizon 7. https://www.vmware.com/products/horizon.html.

[5] D. Wilkinson. Citrix application layering –user layers, WilkyIT. https://wilkyit.com/2018/01/16/citrix-application-layering-user-layers/.

Making (and Keeping) It Simple: Learning to Find Initial Problem Simplifications for Incremental Development in a First Programming Course*

Faculty Poster Abstract

John H. E. Lasseter
Department of Mathematics
Computer Science
Hobart
William Smith Colleges
Geneva, NY 14456
lasseter@hws.edu

This poster presents the results to date of an initiative to teach novice programmers the identification of simplifications of a programming problem, as a first step in an incremental development process. This is part of a broader work in progress to develop a curriculum for a first programming course, which teaches an explicit process of programming as an integral part of the course. The process that students learn to employ draws from the agile programming principles of incremental development and continual testing. At its core is the identification of candidate simplifications of a problem as a fundamental problem-solving strategy. A simplification may be quite radical, but it is chosen such that the programmer can implement a complete solution, from design and implementation to testing. The cycle is then repeated with an identified improvement to the existing program that grows the finished product closer to a full solution.

Identification of the problem simplification step is taught throughout the full semester, at varying levels of granularity, including algorithmic problem solving, functional and/or data type decomposition, and whole program con-

struction. At each of these levels, the initial simplification is one that satisfies the following criteria:

- It should be congruent to the original problem in the sense that it shares broad stroke characteristics in its interface.
- It should solve a subset of the original problem domain. Concretely, this requires the co-development of test cases on which the program should execute correctly.
- The simplified problem should be one that the student clearly knows how to solve. Specifically, it must be possible for the student to implement a complete solution of the simplified problem, which passes the (likely very small) subset of test cases described by the simplification.

Exactly how one finds such a candidate simplification is not obvious, least of all to be-ginning programmers. This kind of identification is another skill that students must develop over time. The core strategies that I use to teach it depend on the granularity of the problem. They fall into three broad categories: templates and algorithmic patterns, method specification and prototype, and object-oriented design patterns.

Templates and patterns of algorithm implementation and whole program construction.

Students begin the first week or so of the course learning a few simple "idioms" of program construction. Their first programs execute straight-line sequences from command line arguments, with interactive I/O templates introduced around the third or fourth week of the course. At a smaller granularity, they learn templates for the basic forms of multi-way conditionals, definite loops, interactive I/O loops, forward and backward array traversals, and so on. Although the patterns must necessarily remain abstract, the templates themselves are working or nearly-complete code examples. Further, the students are required to ensure that they produce only working code from these patterns. At root, this is an adaptation of the agile practices of test-driven development and incremental growth. The simplification task becomes one of identifying the candidate program template, which, initially, is drawn from a very small pool of possibilities. Concomitantly, this involves the identification of suitable tests. Along with a repertoire of templates and patterns, students practice relating a chosen pattern to a given problem, identifying the subset of the problem instances that the basic template solves (and those it does not).

Demand-driven method decomposition and rapid prototype.

The insertion of a call to an invented method is an effective way of delaying a programming concern. The corresponding prototype method—implemented just enough to com-pile—forms the body of candidate simplifications here. This aspect of problem simplification shares a similar character to the specification driven "design recipes" advocated in Felleisen et al. It draws heavily on ideas from design by contract, i.e., the practice of designing methods in terms of both their type signature as well as their pre and post-condition behavioral requirements.

Object-oriented design patterns.

Our version of CS1 teaches objects late in the semester, so exposure to more industrial-scale patterns can prove overwhelming to students. Indeed, the very concept of this kind of design pattern—standard decompositions into class definitions that communicate with each other according to a generalized pattern of interaction—is not at all obvious to beginning programmers, who often struggle with the basic ideas of classes as definitions of new data types. The goal at the novice level to inculcate in students a sense of the utility of this kind of decomposition for larger program construction problems. Students learn just a few simple pat-terns for such things as basic graphics and GUI constructions. Most of these do not rise to the level of the canonical patterns familiar from works such as Gamma et al, though I have had success in teaching a spare version of the model/view/controller architecture.

Validation and Future Work

The success of the template and method prototype approaches was investigated through an analysis of student performance on two exams, whose results will be part of the poster. The data gives evidence that students were able to apply a repertoire of templates and patterns to the solution of new problems. Anecdotally, I observed that the results were significantly better than exam scores observed for students taught without this emphasis, but concrete measurements of the latter approach remain as future work. The same exam results also give evidence that students are able to use some aspects of method call and prototype to aid in initial simplifications and problem solving. These results are more mixed, however. Students showed a consistent ability to specify and use a method' s type signature, but they showed much less success in specifying (and implementing) the behavioral requirements of a method.

Students' Misconceptions of Gradient Descent Algorithm in an Machine Learning Course*

Faculty Poster

Karen Jin
Department of Applied Engineering and Sciences
University of New Hampshire
Manchester, NH 03101
karen.jin@unh.edu

Machine learning has transformed many areas of computer science. It is becoming an important topic in computing education, but the knowledge of how to teach machine learning effectively is limited [1]. Gradient descent and its variants are one of the most important optimization algorithms used in machine learning. It is often among the early topics covered in a machine learning course. The concept of calculating the gradient, or the partial derivative of a cost function for each input dimension, also applies to the backpropagation algorithm in training a simple neural network.

In this poster, we present our experience of teaching the gradient descent algorithm in an introductory-level machine learning course. The course was taken by graduate students in an IT major, many of them are IT professionals with a proficient programming background. When the gradient descent algorithm was first presented, students appeared to understand the algorithm well. They were able to describe how algorithm work at a high level as well as to apply the related tools from machine learning toolkits to solve actual problems. However, when the concept of stochastic gradient descent and backpropagation were later introduced in the course, students' initial misunderstanding of gradient descent became apparent, preventing the students from establishing a correct understanding of the later concepts.

We describe in detail the students' main misconceptions about the gradient descent algorithm and discuss factors that may have contributed to these misconceptions. We also discuss potential solutions from the teachers' perspective to help students avoid these mistakes. Our experience shows students

benefit from working through concrete examples and implementation exercises. Hands-on active learning activities in a collaborative setting are particularly helpful for students to learn fundamental machine learning algorithms such as gradient descent and backpropagation. We also suggest that students should be prepared with adequate math prerequisite in order to have a solid understanding of machine learning algorithms, regardless of their prior programming experience.

References

[1] Rebecca Fiebrink Ben Shapiro, Peter Norvig. *Workshop in Research on Learning about Machine Learning.* ICER, 2017.

Open Source as an Extracurricular Activity*

Faculty Poster

Gregory W. Hislop[1], Joanna Klukowska[2], Lori Postner[3]
[1]Department of Information Science
Drexel University
Philadelphia, PA 19104
hislop@drexel.edu
[2]Department of Computer Science
New York University
New York, NY
joannakl@cs.nyu.edu
[3]Department of Mathematics,
Computer Science and Information Technology
Nassau Community College
Garden City, NY
lori.postner@ncc.edu

This poster discusses efforts at three institutions to encourage student engagement with Free and Open Source Software (FOSS) projects via extracurricular activities. This approach helps address the considerable student interest in open source while avoiding the limits of available time in a computing curriculum for new material.

Free and Open Source Software (FOSS) has become a driving force in the software industry, as exemplified by IBM' s recent acquisition of Red Hat. Almost all major technology companies are active contributors to open source, and surveys indicate that at least 94% of all application systems incorporate open source [2]. In addition, FOSS has become a leading source of innovation in software engineering including tools, frameworks, and processes. Humanitarian FOSS (HFOSS) is open source software that exists to address societal needs such as healthcare, education, disaster management, economic development, etc. The openness of FOSS makes real-world computing accessible to students

to an extent that is unmatched, and a faculty community has emerged to develop approaches to HFOSS education.

Mozilla Open Source Student Network

The Mozilla Foundation, known for the Firefox browser, is dedicated to supporting an open internet as a global public resource. The Open Source Student Network [1] is a Mozilla initiative to encourage and support student clubs that will help students learn about, contribute to, and create open source projects. At present this initiative is focused on the United States and Canada, and includes student club efforts at several dozen campuses.

The Open Source Student Network is designed to be student-led, and students can sign up as a step to organizing FOSS activities on their campus. To help establish an initial cadre of student clubs, Mozilla has run a program during 2017 and again in 2018 to work with a diverse group of students on establishing open source extracurricular activities on their campuses.

While the Mozilla initiative focuses on students, there is value in faculty involvement in fostering FOSS extracurricular efforts. Student clubs typically require a faculty advisor, and for an area like FOSS where students may have great interest but little experience, faculty input and guidance may be particularly helpful.

Initial Experiences

The presenters are faculty involved with HFOSS education at three institutions. Students from all three institutions participate in the Mozilla program. This section summarizes experience across the three campuses.

Organization: Each institution has multiple student clubs related to computing, and Drexel and NCC opted to start FOSS as an activity within an existing Women in Computing club. NYU started a new FOSS club called BUGS. Adding activities to an existing club is an easier way to start, but doesn't provide as much visibility or focus on open source. The separate club requires more start-up effort, but has created some campus-wide visibility related to open source.

Activities: Many students have little understanding or experience with open source, so there is need for initial activities to be very introductory. All three clubs started with events that provided an overview of open source, Git, and GitHub. While most students are already using GitHub, is appears that many are not comfortable beyond very basic operations. An initial set of activities from one of the sites is as follows:

- Overview of Open Source –description, licensing, impact, FOSS as a career
- Git Basics –Use of Git locally for version control, conflict resolution
- GitHub Basics –GitHub basics, workflows, feature branches, pull requests
- Open Source Business Models –FOSS professionals, why companies contribute

Having outside speakers as part of open source activities seems to be well-received by students. These speakers have come from FOSS organizations and also corporations that have significant FOSS efforts.

Sustainability: The demands of other activities and the continual turnover in student leadership present challenges for every extracurricular activity. The initial efforts at all three schools are still ad hoc and likely will fizzle out without continued faculty effort. Long term success will require development of a sustainable program of activities and self-renewing organization and student leadership. Thus far, none of these clubs has achieved that for open source activities.

Acknowledgement

This material is based on work supported by the National Science Foundation under Grant Nos. DUE-1525039, DUE-1524877, and DUE-1524898. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References

[1] Mozilla. Mozilla open source student network. `https://opensource.mozilla.community/` January 2019.

[2] Tidelift Professional Open Source Survey Results. How to make open source work better for everyone, 9 key insights from the 2018 tidelift professional open source survey. `https://tidelift.com/about/2018-Tidelift-professional-open-source-survey-results` December 2018.

Developing and Managing Interdisciplinary Programs*

Faculty Poster

Adam Fischbach, Yana Kortsarts, Suk-Chung Yoone
Widener University
Chester, PA 19013
{jafischbach,ykortsarts,syoon}@widener.edu

We present our experience developing and managing new interdisciplinary programs in computer forensics and digital media informatics –the results of successful collaboration with social science faculty. The computer forensics minor is an interdisciplinary program that integrates criminal justice and computer science and combines both theoretical concepts and practical skills to prepare students for a career in the area of information security and digital forensics. The digital media informatics major is an interdisciplinary program run jointly by the computer science and communication studies departments. The program provides both broad and targeted perspectives on the field of informatics and helps students develop unique skills that can be adapted to the rapidly changing computer and media environment through four specialized concentrations: (1) audio-visual, (2) graphics, mobile, web development, (3) gaming artificial intelligence, and (4) digital writing. We describe the various stages in developing the interdisciplinary programs including an analysis of competitive academic programs, evaluation of current resources, qualifications and faculty considerations, the process of developing the program objectives and learning outcomes, and assessment strategies. We focus on common issues that arose during the development process such as the challenge of designing balanced curricula for interdisciplinary programs, the need for designing new courses and renovating existing courses. We also discuss the anticipated costs of the programs, required resources, recruitment strategies, and the administrative approval mechanism.

Computer science is a rapidly changing discipline and our goal is to provide opportunities for students to understand the complexity of the modern world through interdisciplinary learning, explore and make connections to other

fields, and integrate various perspectives to allow for interdisciplinary problem solving. Integrating interdisciplinary thinking into the computer science curriculum is a challenging but rewarding task that benefits faculty and students. Development of the interdisciplinary programs in computer forensics and digital media informatics agree with broad recommendations presented in the 2011 Summary Report "What works in facilitating interdisciplinary learning in science and mathematics" , by the Association of American Colleges and Universities, Project Kaleidoscope.

The poster will consist of three parts. The first two parts will be devoted to detailed descriptions of the curriculum and development process for each interdisciplinary program, including the anticipated costs of the programs, required resources, recruitment strategies, and the administrative approval mechanism. For each program, we will describe the various stages in the development including an analysis of competitive academic programs, evaluation of current resources, qualifications and faculty considerations, the process of developing the program objectives and learning outcomes, and assessment strategies. The third part of the poster will focus on common issues that arose during the development process such as the challenge of designing balanced curricula for interdisciplinary programs, the need for designing new courses and renovating existing courses, and various common administrative issues.

Using Jupyter Notebooks in a Big Data Programming Course*

Faculty Poster

Roland DePratti
Department of Computer Science
Central Connecticut State University
New Britain, CT 06053
roland.depratti@ccsu.edu

In a Big Data Programming course, students often need basic instruction in a new programming language, i.e. Python, Scala, R. Traditional programming language instruction involves a textbook and an Interactive Development Environment (IDE). In a course that already included two textbooks and instruction on Big Data frameworks, the author was looking for an effective way to deliver instructional text and the interactive development capabilities of an IDE that would not add additional cost to the student.

Many data scientists and researchers have found computational notebooks to be a valuable way to centrally capture documentation, code and visualization of their work. Jupyter Notebooks is a popular, open-source computational notebook [5]. Researchers have published findings on how Jupyter notebooks have been used to collaborate and share results [2][1]. Some computer science educators have studied the use of computational notebooks in the classroom [3], but the author was not aware of an assessment of students' views on how computational notebooks compare to textbooks. This poster describes two phases of a project to answer that question. The first phase developed and implemented notebooks, surveyed and analyzed initial findings as part of a larger study that examined the used of computational notebooks in preparing students to complete statistics projects used in analytics-based courses [4]. The second phase analyzed student free-form feedback on the effectiveness of using the notebooks, identified enhancements to improve shortcomings noted in the feedback results, and developed a list of implementation architecture choices when using notebooks.

To capture their views on notebooks versus textbooks, students completed a survey about their experiences with Jupyter Notebooks and textbooks. The

survey asked students, using a Likert-scale, to compare the Jupyter Notebooks used in class to textbooks on clarity, completeness, usefulness in learning the material, and required effort to learn the material. Results demonstrated that students rated the notebooks higher on these qualities. As an example, students rated notebooks highest on usefulness. Notebooks can provide instructional material, executable examples, assignments and test out the correctness of the assignments in the same document.

Students also were able to provide free-form feedback on the course notebooks. Much of this feedback identified topics they would like to have included in the notebooks, as well as areas where they would like more examples. It also identified the need for an enhancement in the correction checking mechanism. The correction checking mechanism was overly sensitive to string formatting when assignments requested them to return a string. Due to that sensitivity, it tended to flag some correct answers as incorrect. As the result of all the feedback, the notebooks have been enhanced to better meet the students' needs.

Based on this work, the author generated a list of characteristics that others can use when evaluating the use of notebooks in their courses. These characteristics include:

- An "interactive textbook" for data exploration that students enjoy using. (Pro)
- Supports a large amount of programming languages and frameworks (114 kernels). (Pro)
- Large open-source community and support. (Pro)
- Large set of add-on tools, i.e. NBGrader that provides auto-grading capabilities. (Pro)
- Each notebook has its own global memory, careful notebook design is important. (Con)
- Multiple implementation architecture choices available (Pro)

Overall, Jupyter Notebooks were found to be an effective way to give hands-on training in a programming language without the need for additional textbooks. As language and framework developers continue to build kernels that allow them to run in a computational notebook, there is the need for future research to inform educators on best practices in using computational notebooks and their role in the classroom compared to the tools we have been using for decades.

References

[1] Eric Shook, Davide Del Vento, Andrea Zonca, and Jun Wang. GISandbox: A science gateway for Geospatial computing. *Proceedings of the Practice and Experience on Advanced Research Computing.*

[2] Michael B. Milligan. 2018. Jupyter as common technology platform for interactive HPC services. *Proceedings of the Practice and Experience on Advanced Research Computing (PEARC '18).*

[3] Ben Glick and Jens Mache. Using Jupyter notebooks to learn high-performance computing. *Comput. Sci. Coll.*

[4] Davis, M., Dancik, G. and DePratti, R. Autograding. Interactive tools for learning R/Python: Preparation for statistics projects. *Proceedings of the 30th Annual International Conference on Technology in Collegiate Mathematics, (to appear March 2019).*

[5] Jupyter. `www.jupyter.org`.

Identifying Skill Sets for Bioinformatics Graduate Students –A Text Mining Approach*

Faculty Poster

Richard Shang and Mohammed Ghriga
Department of Technology, Innovation and Computer Science
Long Island University
Brooklyn, NY 11201
{Di.Shang,Mohammed.Ghriga}@liu.edu

This project proposes a set of skills to serve as a guideline for bioinformatics curriculum design at the graduate level. Bioinformatics, also known as computational biology, is a burgeoning inter-disciplinary field with a demonstrated market need for highly trained experts who can analyze biological data with skills in computation and informatics. Current trends in bioinformatics incorporate machine learning, large data set analytics and artificial intelligence in the diagnosis, treatment and prevention of illness. Students in graduate programs of Bioinformatics typically expect that courses will prepare them for future job markets with employable skills. In an effort to identify the skill sets sought after by employers in Bioinformatics filed, we apply a text mining approach to analyze required qualifications of Bioinformatics jobs posted online. Using the keyword "bioinformatics", we searched on Google Jobs and collected required qualifications of 38 Bioinformatics jobs. All the jobs indicate that a master's degree is required or preferred. Among the job posts, 14 are under the title "Analyst", 13 under "Scientist", and 11 under "Software Engineer".

We first identify skills in high demand by analyzing the frequency of words stated in the job qualifications. As shown in Figure 1, among others, "programming", "data", "Python", "R" are most frequently mentioned in job posts, which suggests the importance of computing capability in the field.

We next conduct bigram and word correlation analysis to further identify the skills required in the job posts. Our analysis suggests besides the bioinformatics expertise and computing skills, communication skills are also highly

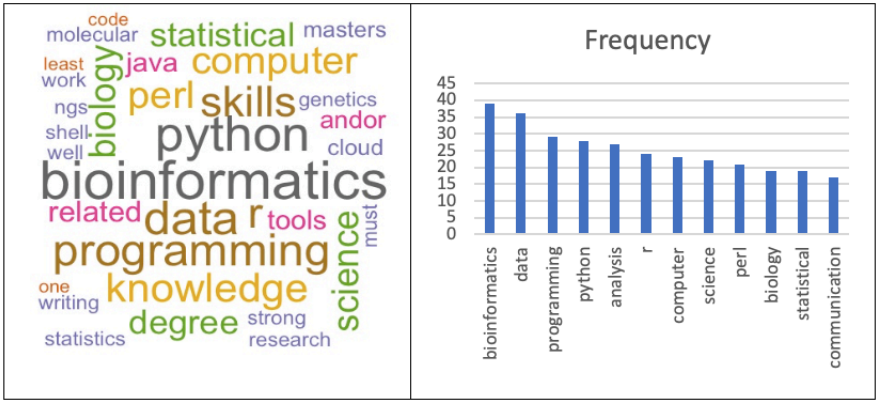


Figure 1: Word Cloud and Word Frequency

desired in the candidates for the job positions (shown in Figure 2). In addition, experience with cloud computing services is preferred in many of the job positions.

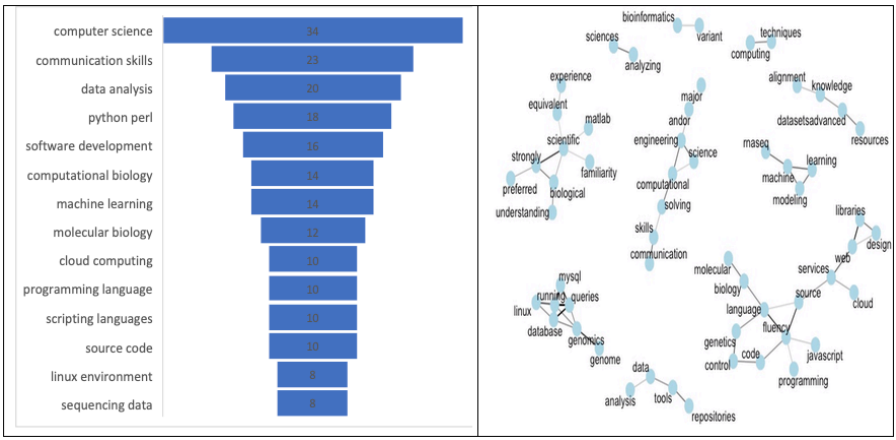


Figure 2: Bigram and Word Correlation

Based on the preliminary results from our analysis, we propose a set of skills which can be essential in the design of Bioinformatics curricula and recommend related courses to accommodate the need for each skill in Table 1. Bioinformatics curricula are expected to impart students with essential domain knowledge in biology, computer science and statistics, and proficiency in computational

and analytical techniques.

Table 1: Required Skills and Recommended Courses

Required Skills	Recommended Courses
Molecular biology; genomics	Bioinformatics / Genomics
Python	Computational Genomics (BioPython)
Data analysis; R statistical	Analysis of Genomic Data (R)
Sequence data; database queries	Database and Big Data
Machine learning	Machine learning in Bioinformatics
Unix environment; shell scripts	Unix for Bioinformatics
Cloud computing	Cloud Computing for Bioinformatics
Communication skills	Capstone Project

Our findings are based on a small sample of Bioinformatics job postings and as such should be interpreted with caution. Yet, we believe they do provide some valuable insights into the skill sets sought after by employers in Bioinformatics filed. In our on-going study we plan to crawl a large number of job posts so that our analysis results can more accurately reflect the job market needs.

Teaching Hands-On Computer Organization and Architecture Using Single-Board Computers*

Faculty Poster

D. Cenk Erdil
School of Computer Science & Engineering
Sacred Heart University
Fairfield, CT 06825
erdild@sacredheart.edu

This work describes details in design and implementation of an upper-level (core-Tier2) computer organization and architecture course with a hands-on component. During revising the course, we (re-)aligned it with the Architecture knowledge area specified in ACM/IEEE joint task force computing curricula [1], and implemented core bodies of knowledge specified, with respect to computer organization and architecture. In particular, the following knowledge areas constitute computer architecture curriculum: (i) digital logic and digital systems, (ii) machine level representation of data, (iii) assembly level machine organization, (iv) memory system organization and architecture, (v) interfacing and communication; as well as the following optional knowledge areas: (vi) functional organization, (vii) multiprocessing and alternative architectures, (viii) performance enhancements.

Computer organization [2, 3], and computer architecture [4, 5] courses are ideally offered as two separate courses with a prerequisite relationship between, and typically in Sophomore, and Junior years, respectively. Due to credit requirements, some computer science departments opt in offering a combined organization and architecture course.

Based on these considerations in our design, as the sole hardware-focused course in computer science department of a liberal arts college, this course was offered as a required course; as part of the four-year undergraduate degree requirements, in computer science, software development, information technology, game design and development programs, as well as an elective course for

a masters degree in computer science. After the redesign, it has been offered first in Fall 2016, and has been continuously offered since then. Data presented in this paper were collected during the inaugural year of offering the course for two cohorts: in Fall 2016, and Spring 2017 semesters.

Modules of the hands-on component are based on three particular contemporary curricular guidelines: flipped classroom, lead learner, and project-based learning. Using these main approaches, the laboratory component of computer architecture course has been designed to complement the theoretical part and also provide students hands-on skills in working with single-board computers, as well as a set of sensors, components, and associated programming interfaces that interact together with the computers.

Another important aspect of this course is to introduce a project component that bodes well with the theoretical part of the course, and hands-on modules, and allows students to design and implement an idea based on what they have learned in class. Students also design and implement a small project with a mandatory hardware component using a single-board computer. As most of the students take this course before their senior year, students reacted well to the idea of designing a small-scale project that would serve as training for their senior capstone projects.

During hands-on modules that started in the first week of the semester, students begin to use Arduino-based [6] single-board computers, and learn how to use small electronic circuits as well as a breadboard and other IC-modules that second-year engineering students typically experiment in a digital design laboratory. This has been a positive learning experience for students as the computer science curricula offered in the department is mostly classical, and did not focus on computer architecture related knowledge areas to a great extent, whereas a large portion of the theoretical part in the course assumed some level of experience.

References

[1] ACM/IEEE-CS joint task force on computing curricula, computer science curricula 2013. Technical report. Final Report, December 20, 2013.

[2] S. P. Dandamudi. *Fundamentals of Computer Organization and Design.* Texts in Computer Science. Springer New York, 2006.

[3] O. Garcia. Computer organization and architecture and the laboratory sequence. *Computer*, 10(12):91–96, December 1977.

[4] D. C. Hyde. Teaching design in a computer architecture course. *IEEE Micro*, 20(3):23–28, May 2000.

[5] C. M. Kellett. A project-based learning approach to programmable logic design and computer architecture. *IEEE Transactions on Education*, 55(3):378–383, August 2012.

[6] Arduino Project. Arduino reference manual. `https://www.arduino.cc`.

Challenges and Successes of Offering Computer Science Courses in Urban High Schools: Perspective of Principals and Administrators*

Faculty Poster

Sarbani Banerjee [1], Neal Mazur[1]
Christopher Shively [2], Joseph Zawicki[3]
[1]Department of Computer Information Systems
[2]Department of Elementary Education and Reading
[3]Department of Earth Science and Science Education
State University of New York at Buffalo State
Buffalo, NY 14222
{banerjs, mazurnm, shivelct, zawickjl}@buffalostate.edu

This research presentation reports on study of the perspectives of principals and administrators of urban high schools in introducing computer science (CS) courses in their schools. These schools serve mostly high needs populations, with more than 70% of students receiving free and reduced-priced lunches, are located around the city of Buffalo, NY, one of the five poorest cities in the country. The interviews and surveys are focused on the challenges and successes of the principals and administrators of these schools as they successfully implement or fail to introduce rigorous CS course(s) in their high schools. A rigorous CS course in high school may extend from a half-year introductory CS course (e.g. Exploring CS [1] or CS Discoveries [4]) to a pre-AP CS course or to a full-year AP CS Principles or AP CS A (Java) course.

Many of the teachers of these schools have participated in the Computer Science for High Schools (CS4HS) professional development workshops, funded by Google, that are offered at Buffalo State College since 2012 [9]; these multiple-track workshops are designed to acquaint participants with introductory CS curriculum including problem solving, Scratch programming, and Web Design [1] as well as the new AP CS Principles curricula [2]. A Service Learning

program at the Computer Information Systems Department at the authors' college sends 15-20 students each semester to these schools to help the participating CS4HS teachers either with their CS clubs or in their CS classrooms. The college students provide near-peer CS teaching and learning for the high school students of Buffalo Public Schools and Buffalo Charter Schools.

A 'Western New York Principals' Summit', attended by more than 65 high school principals and administrators of the WNY area, was organized by the authors in spring of 2017 and included invigorating keynote addresses by Jan Cuny, NSF Program Director and Chris Stephenson, Director of CS Education at Google. A major goal of the summit was to encourage principals to initiate and expand CS education in their schools through the interaction with the keynote speakers and from listening to success stories provided by principals who have supported strong CS programs at their schools. Since then many schools in the WNY area have started offering CS courses but progress has still been slow in the inner-city schools that mostly serve the minority populations that are underrepresented in computing.

The levels of introducing computer science informally or formally vary greatly from school to school in the urban areas. Many participating CS4HS teachers have established computer clubs, imparting informal

CS education, few have been teaching introductory CS courses, and fewer have started teaching AP CS Principles course since last year. Many schools have not been able to offer any CS courses despite of the trained CS4HS teachers in their schools and many inner-city schools have never sent their teachers to any CS professional development workshops.

The initial offerings of the AP CSP course in 2016-2018 saw a doubling of the number of female as well as underrepresented minority students taking AP computer science courses/exams from that of the previous year [4]. There is clearly both a strong interest and a strong need for engaging underrepresented populations in CS [5]. In contrast, multiple barriers exist to the implementation of CS courses in many K-12 settings [7][3][8][6]. An availability of up-to-date, functional computer labs, the lack of well-prepared faculty, adequate time in student and teacher schedules, the status of CS as an elective, have all contributed to the current dearth of CS courses in many districts. For this research study, a survey instrument with 34 questions has been designed and was administered to 24 high school principals of Buffalo Public and charter schools, that enroll inner-city students. Eighty percent of the students in these schools are from underrepresented populations in CS. The findings will inform the CS ecosystem at multiple levels. Details, specifically related to the barriers of CS course offerings will be addressed. Suggestions for and resources supporting overcoming the challenges will be provided and identified. The presentation will be arranged to have a prioritized summary and analysis

of the principals' survey and interviews addressing issues related specifically to offering CS courses, and next steps for schools - including both those who have implemented a CS course as well as those planning to do so in the future.

References

[1] Exploring computer science. https://csforallteachers.org/exploring-computer-science.

[2] Astrachan, O., and Briggs, A. The CS principles project. *ACM Inroads*, 3(2):38–42, 2012.

[3] Century, J., Lach, M., King, H., Rand, S., Heppner, C., Franke, B., Westrick, J. (2013). Building an operating system for computer science. http://outlier.uchicago.edu/computerscience/OS4CS/.

[4] Code.org. https://code.org/educate/csp, Last Retrieved January 2, 2019.

[5] J. Cuny. CS principles professional development: Only 9,500 to go! lessons learned from our CS10K summer 2013 PD. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, page 543 –544, 2014.

[6] Google Gallup Report (2015). Searching for computer science: Access and barriers in U.S. K-12 education. https://services.google.com/fh/files/misc/searching-for-computer-science_report.pdf.

[7] Jennifer Wang, Hai Hong, Jason Ravitz Sepehr Hejazi Moghadam. Landscape of K-12 Computer Science education in the U.S.: Perceptions, access, and barriers. *SIGCSE' 16*, page 645 –650, 2016.

[8] Loewus, L. (2015). Survey: Principals differ on definition of computer science. http://blogs.edweek.org/edweek/curriculum/2015/01/survey_computer_science_lacks.html.

[9] Mazur, N.M., Banerjee, S. and Santa Maria, R. Computer science for all in western New York: Building a community of practice. *The Journal of Computing Sciences in Colleges*, 32(6), 2017.