The Journal of Computing Sciences in Colleges

Papers of the 25th Annual CCSC Northeastern Conference

April 17-18, 2020
Ramapo College of New Jersey
Mahwah, NJ

Baochuan Lu, Editor
Southwest Baptist University

Mihaela Sabin, Regional Editor
UNH at Manchester

Volume 35, Number 8

April 2020

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.

Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

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.

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(2022), (317) 456-2125, dnaugler@semo.edu, 5293 Green Hills Drive, Brownsburg IN 46112.

Lawrence D'Antonio, Northeastern Representative (2022), (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 (2022), 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, 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

2020 CCSC Northeastern Conference Steering Committee

Lawrence D'Antonio, Conference Chair	Ramapo College of New Jersey
Ben Fine, Conference Chair	Ramapo College
Jim Teresco, Program Chair	Siena College
Ali Erkan, Papers Chair	Ithaca College
Yana Kortsarts, Papers Chair	Widener University
Susan Imberman, Lightning Talks Chair	The City University of New York
Joan DeBello, Panels Chair	St. John's University
Bonnie MacKellar, Tutorials and Workshops Chair	St. John's University
Ting Liu, Tutorials and Workshops Chair	Siena College
Dan Rogers, Faculty Posters Chair	The College at Brockport
Ingrid Russell, Speakers Chair	University of Hartford
Mike Gousie, Speakers Chair	Wheaton College (Massachusetts)
Karl Wurst, Student Unconference Chair	Worcester State University
Darren Lim, Encore Chair	Siena College
Sandeep Mitra, Undergraduate Posters Chair	The College at Brockport
Alice Fischer, Undergraduate Posters Chair	University of New Haven
Aparna Mahadev, Undergraduate Posters Chair	Worcester State University
Stefan Christov, Undergraduate Posters Chair	Quinnipiac University
Liberty Page, Undergraduate Posters Chair	University of New Haven
Mark Hoffman, Registration Chair	Quinnipiac University
Rick Kline, Registration Chair	Pace University
Frank Ford, Programming Contest	Providence College
Del Hart, Programming Contest	SUNY Plattsburgh
Scott Frees, Career Fair Coordinator	Ramapo College
Kevin McCullen, Vendors Chair	SUNY Plattsburgh

Regional Board — 2020 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 — 2020 CCSC Northeastern Conference

Chris Alvin	Furman University
Barbara Bracken	Wilkes University
William Campbell	UNC Pembroke (retired)
Kailash Chandra	Pittsburg State University
Stefan Christov	Quinnipiac University
Mary Courtney	Pace University
Lawrence D'Antonio	Ramapo College of New Jersey
Garrett Dancik	Eastern Connecticut State University
Elise Deitrick	Codio
Dan DiTursi	Siena College
Peter Drexel	Plymouth State University
Benjamin Fine	Ramapo College of New Jersey
Alice Fischer	University of New Haven
Robin Flatland	Siena College
Timothy Fossum	Rochester Institute of Technology
Seth Freeman	Capital Community College
Martin Gagne	Wheaton College
Alessio Gaspar	University of South Florida Polytechnic
Micheal Gousie	Wheaton College
Nadeem Hamid	Berry College
Scott Harrison	St. John Fisher College
Delbert Hart	SUNY Plattsburgh
Michalina Hendon	Bloomsburg University
Mark Hoffman	Quinnipiac University
Karen Jin	University of New Hampshire
William Joel	Graphics Research Group/WCSU
Erin Johnson	CodeCrew
Jeremiah Johnson	University of New Hampshire
Sotirios Kentros	Salem State University
Bo Kim	Southern New Hampshire University
Zach Kissel	Merrimack College
Bradley Kjell	Central Connecticut State University
Devorah Kletenik	City University of New York
Daniel Krutz	Rochester Institute of Technology
David Levine	St. Bonaventure University
Jingsai Liang	Westminster College
Yi Liu	University of Massachusetts Dartmouth
Mihaela Malita	Saint Anselm College

Kevin McCullen	SUNY Plattsburgh
Paul-Marie Moulema	Western New England University
Muath Obaidat	City University of New York
Pat Ormond	Utah Valley University
Greta Pangborn	Saint Micheal's College
Sofya Poger	Felician University
Jennifer Polack	University of Mary Washington
Daniel Rogers	The College at Brockport
Nicholas Rosasco	Valparaiso University
Christopher Stuetzle	Merrimack College
Sheng Tan	Trinity University
Jim Teresco	Siena College
David Voorhees	Le Moyne College
Marc Waldman	Manhattan College
Songmei Yu	Felician University
Junxiu Zhou	Northern Kentucky University

An Overview of Data Analytics: Spreadsheet Modeling, Visualization, and Supervised and Unsupervised Learning[*]

Carolyn C. Matheus
Computing Technology Department
Marist College
Poughkeepsie, NY 12601
`Carolyn.Matheus@Marist.edu`

Abstract

Data science and analytics have emerged as thriving fields. As businesses and individuals produce massive volumes of data as a byproduct of online activity, a growing need exists for professionals trained to capitalize on the potential of big data by understanding how to use analytic techniques to generate valuable information from large collections of data. At the same time, online education is one of the fastest growing segments of higher education. The number of students working online toward Master's degree increases each year. The Association to Advance Collegiate Schools of Business (AACSB) and the joint task force of the Association for Computing Machinery (ACM) and Association for Information Systems (AIS) have called for data analytics in graduate curriculum. To meet these demands, this paper provides an overview of a skills-based online graduate course in which students learn statistical techniques for approaching big data. The hands-on curriculum focuses on spreadsheet modeling, data visualization, rudiments of data management and data analysis, and an introduction to data mining and predictive modeling, combined with state-of-the-art software, real world data sets, and the skills necessary to use the tools. This paper provides an overview of the course goals and curriculum, data sets and software tools used for visualization and analytics, and the online platform used as the content management system for course delivery.

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Data science and data analytics have emerged as thriving fields. As businesses, governments, and individuals produce massive volumes of data as a byproduct of online activity, there is a growing demand for professionals trained to capitalize on the potential of big data by understanding how to use analytic techniques to generate value from large collections of data [4]. Important characteristics of such people include creativity, curiosity, analytical abilities, statistical expertise, and communication skills combined with knowledge of how to drill down data to tell a meaningful story to stakeholders [6, 7]. At the same time, online education is one of the fastest growing segments of higher education. An estimated 3.5 million students were working toward their degree online in 2016; this number is expected to increase to 5 million by 2020. Business continues to be the most popular major for undergraduates and graduates, accounting for approximately 25% of enrollment in online degree programs [5].

The Association to Advance Collegiate Schools of Business (AACSB) included data analytics in Business Standard 9 for curriculum content and Accounting Standard A7 for Information Technology Skills and Knowledge for Accounting Graduates [9]. Likewise, the joint task force of the Association for Computing Machinery (ACM) and Association for Information Systems (AIS) included data management and analytics as necessary components of graduate degrees in Information Systems, including integrating and preparing data for analytical use, applying analytics methods, and analyzing data using advanced contemporary methods [8]. This paper provides an overview of the development and delivery of this online graduate course titled Analytics. This course introduces a range of data driven disciplines and technologies to help students understand how analytics can be used to make better, faster business decisions. Students in this course are exposed to spreadsheet modeling, data visualization, data management and analysis, and an introduction to data mining and predictive modeling. The course incorporates real world data sets and scenarios from different domains, and state of the art software coupled with the skills necessary to use the tools. This paper provides an overview of the course goals and curriculum, data sets and software tools used for visualization and analytics, and the online platform used for content management and delivery.

2 Course Goals and Management

This course aims to change the way students think about data and its role in business, gaining an understanding of how organizations use analytics to solve problems and support decision making. Students become familiar with the concepts of relational data manipulation and querying, dimensional analysis, and database access. They learn to interpret and use quantitative and decision support techniques through the use of spreadsheet models, data visualization techniques, and become familiar with data mining and predictive models through the use of specialized software. Content is delivered online using the iLearn content management system, which was adapted from the open-source Sakai platform for learning management systems [2]. Through the iLearn interface, students can access all course materials by utilizing a series of functional tools, including: a video introduction by the instructor, a calendar of assignments and due dates including an assignment submission site, an email system, an online gradebook, a tool for interactive discussion forums, a tests and quizzes tool, and a synchronous chat room. Additional functionality includes a tool for podcasts; a polling tool for posting questions, anonymous voting, and gathering results; a web-content tool for linking to internal resources or external websites; a news tool for RSS feeds; and an optional tool that provides early alert and detection of academically at-risk students. The learning modules landing page guides students through the course content, which is largely based on an organized series of video tutorials, supporting lecture slides, and assigned readings. Each module is delivered as its own dynamic web page. The mandatory content is delivered asynchronously to allow students the flexibility to view content and complete assignments on their own schedule. Tools for synchronous communication, including live chats and video communication, are available through the course interface. This course adheres to institutional and Quality Matters guidelines for online courses. Quality Matters includes metrics for ensuring courses meet rigorous requirements and guidelines to ensure the highest level of online instruction, including guidelines for developing learning objectives, assessing and measuring goals, course technology, delivery of instructions materials, interaction with and support for learners, and accessibility and usability [1].

3 Course Curriculum and Data Sets

The curriculum is delivered via an online platform as three distinctive learning modules. Module 1 teaches spreadsheet modeling using Microsoft Excel, Module 2 teaches data visualization using Tableau, and Module 3 teaches supervised and unsupervised learning utilizing Weka. Each Module includes a series of video tutorials, lecture slides and videos, assignments, discussion questions, and interactive discussion forums with classmates and the professor. The software platforms (i.e., Excel, Tableau, and Weka) were collaboratively chosen by faculty and administration from Information Systems and Business, as well as feedback from industry professionals, based on currently trending needs for skillsets of graduates entering and exceling in the workforce. Two different data sets were used across the three learning modules: a large open breast cancer data set and a weather data set. The breast cancer data set includes data points of attributes related to breast cancer such as age, menopause, tumor size, details about nodules, and breast density. The weather data set includes a variety of weather data points including temperature in Fahrenheit, dew point, humidity, visibility, precipitation, and wind speed. The following sections provide a detailed overview of how the data sets were incorporated into the three learning modules.

3.1 Spreadsheet modeling

Using the large weather data set, students learn how to perform advanced mathematical calculations including: sort, filter, and format data; create, insert, and edit charts to graphically display results; conditional formatting and advanced IF functions; create array and related formulas; create macros; and generate pivot tables. Students are guided through a series of video tutorials and lectures delivered through the course site that teach and demonstrate the lessons. They then apply techniques for manipulating the data by answering a series of assigned questions. Assignments include learning how to freeze lock rows and columns, password protect files, create formulas, and calculate averages for columns which must be displayed in a designated cell, such as =AVERAGE(C2:C6721). In one question, students are asked to add a new column for Celsius temperature and generate a formula to convert temperatures in Fahrenheit to Celsius; for example, =(C2-32)*5/9. In addition, students are asked to generate a formula to calculate the difference in hours between April 4 at 16:00:00 and May 25 and 8:00:00. Students learn to visualize their results by generating charts and graphs to display the data and learn to edit the axes (see Figure 1).

Advanced spreadsheet tasks include conditional formatting, nested countifs, and pivot tables. Conditional formatting instructions include: Use conditional formatting for the column of DPf (Dew Point, Fahrenheit). If the value is above 67°, denote with a green dot. If the value is between 33°and 67°, denote with a yellow dot. If the value is below 33°, denote with a red dot. Figure 2 presents a visual example that demonstrates select rows.

Students learn to create nested IF statements to denote data ranges by adding a new column to the spreadsheet and creating an IF statement that accounts for the following challenge: If the TmpF (Temperature in Fahrenheit) is between 45°and 70°, then the value is comfortable; if the temperature is over 70°, then the value is too warm; if the temperature is lower than 45°, then the value is too cold. The formula summarizes the data in a chart that provides a live count of the ranges denoted. Figure 3 presents a visual example.

Students also learn to use pivot tables for extracting information from the data. Pivot tables allow the ability to quickly extract this type of information and present it in an easy to understand chart. Students are tasked with generating a pivot table to show the average TmpF and DPf by year (see Figure 4).

3.2 Visualization

Tableau is a software suite for data visualization [3]. Students receive free academic license for the course. In Tableau, students learn how to import data into Tableau, edit metadata, blend and drill down data, create data subsets, sort and group data, and set parameters for filtering and formatting data. Students also learn how to evaluate and interpret their observations and draw conclusions to summarize the meaning of data patters through interactive dashboards and story features. Students can optionally join the course's asynchronous open discussion forum or synchronous live chat room to discuss questions with peers and the professor. Using the weather data set, students address a series of assignment challenges. Sample questions include:

- Use a filter to remove the data in 2008 and describe whether there is a visual change in the shape of the data points for only the 2009 data, and explain why.
- Create a set which contain all of the dates in January. Drag the set to the column between Date and Time. Describe the result and explain why the result looks like this.
- The Time column incorrectly contains a date of 1899/12/30. Find a solution to fix this problem. Drag the Time dimension to a column and TmpF to a row. Change the TmpF calculation to Average. Write a short report that describes when it is coldest in a day and when it is warmest in a day.
- Create a Night Time group from midnight to 6:00 am. Create a short report explaining whether the Night Time group has the lowest temperature.

At each step students are required to show their work as they critically evaluate their output. Students can show their work by uploading their progress to Tableau server or produce a series of screenshots to embed in their assignments to show their process. Figure 5 depicts an example of the Tableau interface with data dimensions and measures as well as a bar chart demonstrating trends in average TmpF and DpF data by quarter. Figure 6 presents the data using a different visual style.

The final challenge is to pull together findings and presents a cohesive story with the data. Students can push their charts and graphs from previous assignments into a dashboard and edit the layout to make a presentation with the data. Students are expected to evaluate the data, provide a critical evaluation of the data trends, and present their findings in a way that is visually appealing through charts, graphs, and a narrative that tells a story with the data. Students are informed, This assignment is intended to bring it all together, to tell a story with the data. The narrative of the story is yours. Make sure you provide a rationale and explanation of why you choose to present the data you are presenting in a particular way. Figures 7 and 8 show examples of dashboard stories created to visually showcase trends in the data and different techniques for charts and graphs.

Students also participate in discussion forums where they virtually interact with their peers. These graded forums are designed to help the students as they progress through the content of the course. An example forum question for the Tableau learning module is: Please see the attached trend line and discuss with your classmates what this trend line means. The column is AVG(DPf) and row is AVG(Tmp F). Please pay attention to R-square and P value, and interpret the result. Figure 9 demonstrates the trend line referenced in the discussion question. Responses must include a clear explanation of the trend line, with examples, explanations, and interpretations of R-square and P values. For example, the graph shows evidence of a strong positive linear relationship between TmpF and DpF. That is, as temperature increases, dewpoint also increases. There are numerous factors to look at when drawing this conclusion, including p-value, R-squared, and additional factors of regression equations. The output for the trend line model shows $p < 0.05$, indicating a high level of certainty that the data are related and results are statistically significant. In addition, an R-squared value of .89 indicates the trend line has a positive slope with high statistical significance, suggesting a good fit of the trend line for this relationship.

3.3 Supervised and unsupervised learning

Weka is an open source program with machine learning algorithms for data mining [10]. Weka includes tools for data pre-processing, regression, association rules, and visualization, as well as classification and clustering algorithms. Using an open breast cancer data set, students learn how to normalize data, run cross-validation and training/testing decisions trees, and visualize the results. Using the weather data set, students learn how to run and interpret data mining clustering algorithms. Figure 10 provides an example of the Weka interface for supervised learning using an open breast cancer data set. The following sections provide an overview and specific examples of questions, tasks, and assignments students complete for supervised and unsupervised learning tasks using Weka.

Supervised learning with decision trees. Students complete tasks related to decision trees, which can be used for calculating probabilities and evaluating conditions for predicting the likelihood of an outcome. Students use Weka to evaluate a large open breast cancer data set. After viewing a series of video tutorials, student run the data set with J48 decision trees using the cross validation in Weka and interpret the difference in results compared to using J48 by 66% as training and 34% as testing. They are then asked to answer questions and provide documentation of their findings, including:

What is the difference between running cross validation and testing/training? A model answer should incorporate aspects of the following explanation: Cross validation, which is better suited for smaller data sets (e.g., less than 1000 data points), holds out 10% of the data and uses 90% for training. Each data point is tested one time and used for training 9 times. The results of these 10 runs are then averaged, and then Weka runs the entire data set as a test against this average. Put another way, 10-fold cross-validation processes the data by dividing it into 10 folds. During each iteration, one-fold is held out during the ten trials and the results are then averaged. The eleventh iteration is the final step of the process whereby all of the data is used to obtain the actual classifier. Another technique for building a classifier from a data set is training/testing. The data can be split, where the larger percentage of data is used to train (e.g., 66% used for training), while the rest of the data (e.g., 34%) is used to test and refine the final classifier. This technique is better suited for larger data sets.

Provide a side by side comparison of instanced, attributes, number of leaves, size of tree, and correct and incorrect classifications. Figure 11 presents an example of this information, which is generated as output in Weka.

Provide a screenshot that visualizes your cross validation tree. Figure 12 presents an example of a visualized tree for this data.

Unsupervised learning with clustering algorithms. Students complete tasks related to clustering algorithms, including Simple K-means, Farthest First (FF), Hierarchical Clustering (HC), and Expectation Maximization (EM). After completing required readings, watching a series of video tutorials, and viewing a narrated tutorial with the professor working with the data set, students complete related tasks and answer directed questions. For this assignment, students use a revised weather data set that includes the aforementioned weather data points as well as data related to public transportation stations (e.g., daily weather conditions, type of transportation station, attributes related to the station such as whether it is an indoor or outdoor station, number of passengers using the stations, usage on weekday versus weekend, distance between stations, etc.). The goal is to use clustering algorithms to examine the data and determine how weather conditions might predict the use of public transportation stations. For example, students run the weather data using the simple K-Means algorithm and answer questions such as:

How many clusters are formed from this data set, and how many instances are there of each cluster? A model answer should reference the output, which shows two clusters depicted as Cluster 0 and Cluster 1 (see Figure 13)

Briefly describe what the clusters mean and what they represent. A model answer should include information about how the clusters were formulated. For example, The K-Means algorithm calculates the centroids of the number of clusters, and the individual distances measured from the centroids. The algorithm assigns data points to one of the groups based on the distance between stations, number of passengers, time, and weather until a constant within the clusters becomes clear. In the current example, two clusters are formed (Cluster 0 represents busy stations, and Cluster 1 represents slower stations). Students are also asked to evaluate the meaning and importance of standard deviations (SD). For example: Why is the SD in busy stations larger than less busy stations? Model answers should reference how a SD is calculated and what it means regarding the current data set. For example, a higher SD indicates the data points (i.e., number of passengers per station) are spread out over a wider range of values. Busier station, represented as Cluster 0, have higher utilization rates (e.g., number of passengers). Therefore, the SD is higher than Cluster 1 because of the higher influx of passengers in busy stations compared to less busy stations.

Discussion forums. In addition to completing assigned questions, students participate in discussion forums where they read and view additional content and answer questions posed to them, as well as virtually interacting with their peers in the class. The forums are designed in a way to help students along as they progress through the content of the course. Below are example discussion forum questions students are challenged with:

- Besides Weka, there are a lot of data analysis and machine learning tools on the market, for example, SPSS, SAS, KNIME, R, SPARK, HADOOP, and SAP. In this discussion share with your classmates which tool(s) you have used in your work or personal projects. If you have not used any data analysis tools, please talk about the type data you have in your work and which tool(s) may be useful for your future work.
- Please read the definition of association rule learning. Discuss whether association rule is supervised or unsupervised learning. Also, think about the data you used in your own work and whether there is any project or research question you could use association rule for mining the answer.
- Using the resources provided (e.g., additional readings, slides, and supplemental video lecture), as well as additional research you may choose to conduct independently, please discuss the similarities and differences between the following four algorithms: K-Means, Farthest First (FF), Hierarchical Clustering (HC), and Expectation Maximization (EM). A model answer must describe how the algorithms compute clusters and what types of research and variables they are best suited for.

4 Discussion and Conclusion

Results of evaluations regarding student perceptions of the course's effectiveness have been positive across five semesters. On a scale of 1 - 5 (1 = strongly agree, 5 = strongly disagree) students were asked to rate their perceptions of the course, instructor, and additional demographic information. Specifically, they were asked to rate their perception of the extent to which the instructor: meets the stated course objectives; releases content in a timely fashion; effectively answers questions; is available to help students; uses instructional materials to enhance learning; effectively presents course materials; provides clear instructions (readings, discussions assignments) for navigating the course site; overall is an effective teacher. Students were also asked about the number of college credits they have taken to date; whether the course is being taken as part of the major, a minor, or an elective; self-reported level of effort; self-reported perception of work load; level of interest in the course content before and after course completion; and perception of the effectiveness and completeness of the syllabus. The overall course mean, aggregated across all items and semesters, is 1.74.

References

[1] Quality Matters. 2018. Retrieved from `https://www.qualitymatters.org/`.

[2] Sakai Learning Management System. 2018. Retrieved from `https://www.sakaiproject.org/about`.

[3] Tableau. 2019. Retrieved from `https://www.tableau.com/`.

[4] Penny R Clayton and Jeremy Clopton. Business curriculum redesign: Integrating data analytics. *Journal of Education for Business*, 94(1):57–63, 2019.

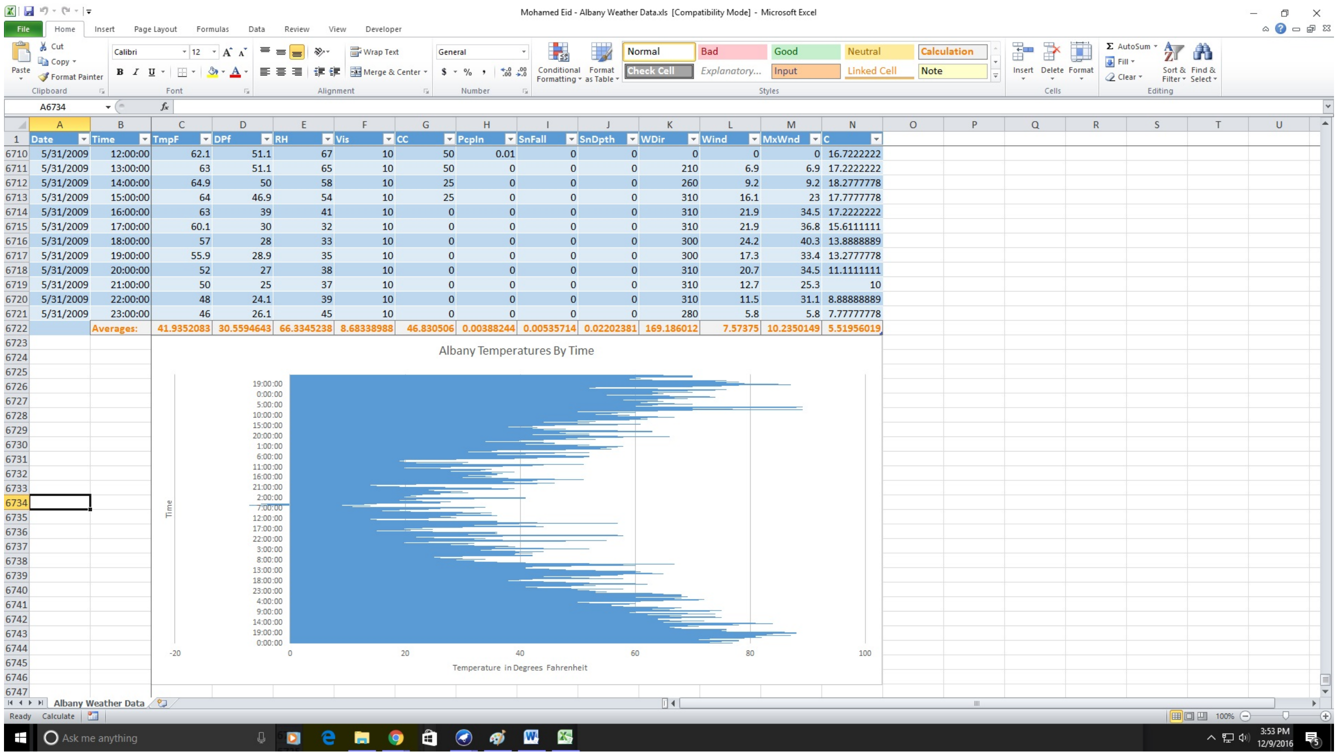
[5] D Clinefelter and C Asianian. Online college students: Comprehensive data on demands and preferences, 2016.

[6] Thomas Davenport and DJ Patil. Data scientist: The sexiest job of the 21st century-harvard business review. *Harvard Business Review*, 2013.

[7] Seth Stephens-Davidowitz and Andrés Pabon. *Everybody lies: Big data, new data, and what the internet can tell us about who we really are.* HarperCollins New York, 2017.

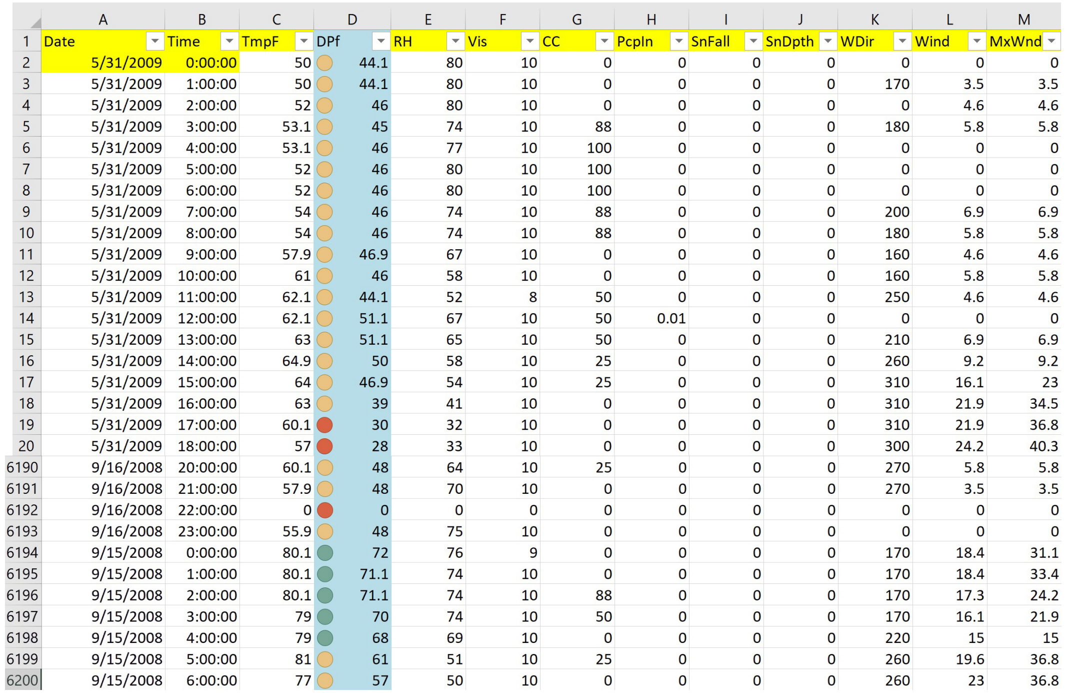
[8] Heikki Topi, Helena Karsten, Sue A Brown, João Alvaro, Brian Donnellan, Jun Shen, Bernard CY Tan, and Mark F Thouin. Msis 2016 global competency model for graduate degree programs in information systems. *Communications of the Association for Information Systems*, 40(18), 2017.

[9] MA Vasarhelyi, N Tschakert, J Kokina, and S Kozlowski. How business schools can integrate data analytics into the accounting curriculum. *The CPA Journal*, 22(2):156–177, 2017.


Mohamed Eid - Albany Weather Data.xls [Compatibility Mode] - Microsoft Excel

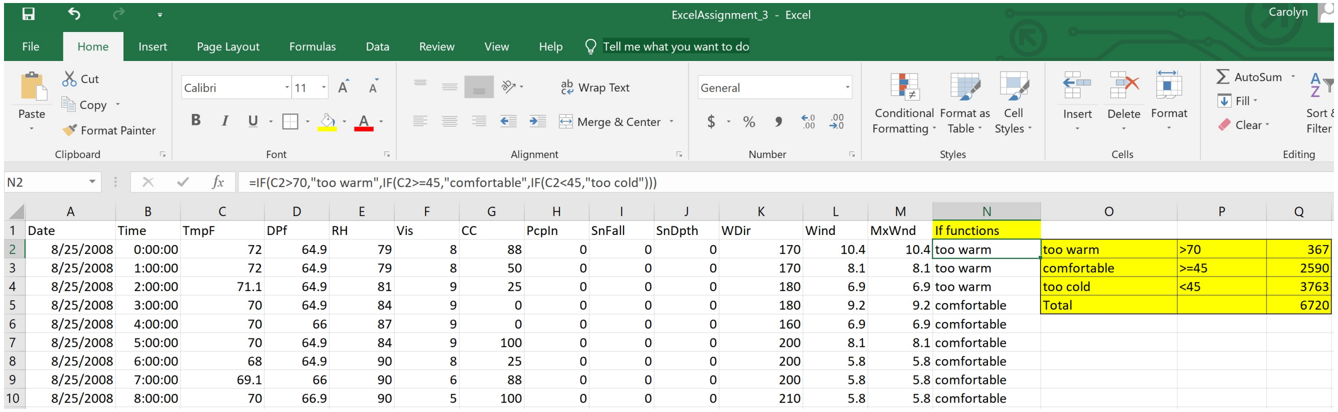
	Date	Time	TmpF	DPf	RH	Vis	CC	PcpIn	SnFall	SnDpth	WDir	Wind	MxWnd	C
6710	5/31/2009	12:00:00	62.1	51.1	67	10	50	0.01	0	0	0	0	0	16.7222222
6711	5/31/2009	13:00:00	63	51.1	65	10	50	0	0	0	210	6.9	6.9	17.2222222
6712	5/31/2009	14:00:00	64.9	50	58	10	25	0	0	0	260	9.2	9.2	18.2777778
6713	5/31/2009	15:00:00	64	46.9	54	10	25	0	0	0	310	16.1	23	17.7777778
6714	5/31/2009	16:00:00	63	39	41	10	0	0	0	0	310	21.9	34.5	17.2222222
6715	5/31/2009	17:00:00	60.1	30	32	10	0	0	0	0	310	21.9	36.8	15.6111111
6716	5/31/2009	18:00:00	57	28	33	10	0	0	0	0	300	24.2	40.3	13.8888889
6717	5/31/2009	19:00:00	55.9	28.9	35	10	0	0	0	0	300	17.3	33.4	13.2777778
6718	5/31/2009	20:00:00	52	27	38	10	0	0	0	0	310	20.7	34.5	11.1111111
6719	5/31/2009	21:00:00	50	25	37	10	0	0	0	0	310	12.7	25.3	10
6720	5/31/2009	22:00:00	48	24.1	39	10	0	0	0	0	310	11.5	31.1	8.88888889
6721	5/31/2009	23:00:00	46	26.1	45	10	0	0	0	0	280	5.8	5.8	7.77777778
6722		Averages:	41.9352083	30.5594643	66.3345238	8.68338988	46.830506	0.00388244	0.00535714	0.02202381	169.186012	7.57375	10.2350149	5.51956019

Figure 1: Class Schedule



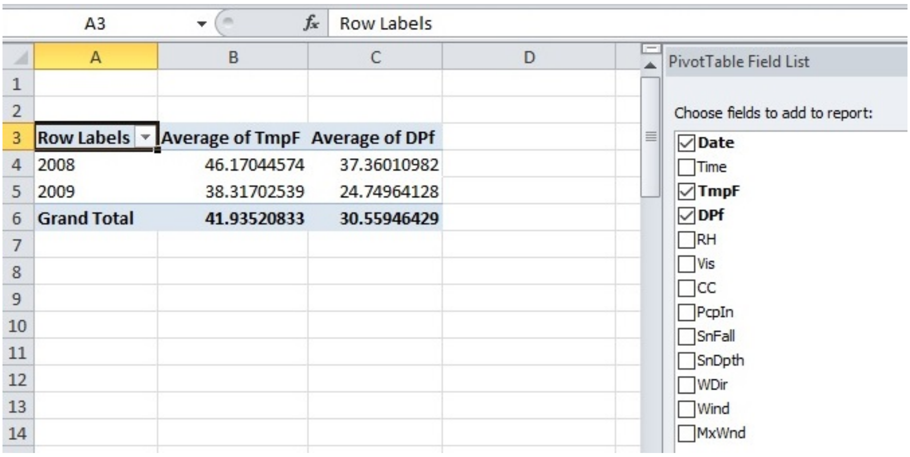
	A	B	C	D		E	F	G	H	I	J	K	L	M
1	Date	Time	TmpF	DPf		RH	Vis	CC	PcpIn	SnFall	SnDpth	WDir	Wind	MxWnd
2	5/31/2009	0:00:00	50	🟡	44.1	80	10	0	0	0	0	0	0	0
3	5/31/2009	1:00:00	50	🟡	44.1	80	10	0	0	0	0	170	3.5	3.5
4	5/31/2009	2:00:00	52	🟡	46	80	10	0	0	0	0	0	4.6	4.6
5	5/31/2009	3:00:00	53.1	🟡	45	74	10	88	0	0	0	180	5.8	5.8
6	5/31/2009	4:00:00	53.1	🟡	46	77	10	100	0	0	0	0	0	0
7	5/31/2009	5:00:00	52	🟡	46	80	10	100	0	0	0	0	0	0
8	5/31/2009	6:00:00	52	🟡	46	80	10	100	0	0	0	0	0	0
9	5/31/2009	7:00:00	54	🟡	46	74	10	88	0	0	0	200	6.9	6.9
10	5/31/2009	8:00:00	54	🟡	46	74	10	88	0	0	0	180	5.8	5.8
11	5/31/2009	9:00:00	57.9	🟡	46.9	67	10	0	0	0	0	160	4.6	4.6
12	5/31/2009	10:00:00	61	🟡	46	58	10	0	0	0	0	160	5.8	5.8
13	5/31/2009	11:00:00	62.1	🟡	44.1	52	8	50	0	0	0	250	4.6	4.6
14	5/31/2009	12:00:00	62.1	🟡	51.1	67	10	50	0.01	0	0	0	0	0
15	5/31/2009	13:00:00	63	🟡	51.1	65	10	50	0	0	0	210	6.9	6.9
16	5/31/2009	14:00:00	64.9	🟡	50	58	10	25	0	0	0	260	9.2	9.2
17	5/31/2009	15:00:00	64	🟡	46.9	54	10	25	0	0	0	310	16.1	23
18	5/31/2009	16:00:00	63	🟡	39	41	10	0	0	0	0	310	21.9	34.5
19	5/31/2009	17:00:00	60.1	🔴	30	32	10	0	0	0	0	310	21.9	36.8
20	5/31/2009	18:00:00	57	🔴	28	33	10	0	0	0	0	300	24.2	40.3
6190	9/16/2008	20:00:00	60.1	🟡	48	64	10	25	0	0	0	270	5.8	5.8
6191	9/16/2008	21:00:00	57.9	🟡	48	70	10	0	0	0	0	270	3.5	3.5
6192	9/16/2008	22:00:00	0	🔴	0	0	0	0	0	0	0	0	0	0
6193	9/16/2008	23:00:00	55.9	🟡	48	75	10	0	0	0	0	0	0	0
6194	9/15/2008	0:00:00	80.1	🟢	72	76	9	0	0	0	0	170	18.4	31.1
6195	9/15/2008	1:00:00	80.1	🟢	71.1	74	10	0	0	0	0	170	18.4	33.4
6196	9/15/2008	2:00:00	80.1	🟢	71.1	74	10	88	0	0	0	170	17.3	24.2
6197	9/15/2008	3:00:00	79	🟢	70	74	10	50	0	0	0	170	16.1	21.9
6198	9/15/2008	4:00:00	79	🟢	68	69	10	0	0	0	0	220	15	15
6199	9/15/2008	5:00:00	81	🟡	61	51	10	25	0	0	0	260	19.6	36.8
6200	9/15/2008	6:00:00	77	🟡	57	50	10	0	0	0	0	260	23	36.8

Figure 2: Conditional Formatting Example


N2 =IF(C2>70,"too warm",IF(C2>=45,"comfortable",IF(C2<45,"too cold")))

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Date	Time	TmpF	DPf	RH	Vis	CC	PcpIn	SnFall	SnDpth	WDir	Wind	MxWnd	If functions			
2	8/25/2008	0:00:00	72	64.9	79	8	88	0	0	0	170	10.4	10.4	too warm	too warm	>70	367
3	8/25/2008	1:00:00	72	64.9	79	8	50	0	0	0	170	8.1	8.1	too warm	comfortable	>=45	2590
4	8/25/2008	2:00:00	71.1	64.9	81	9	25	0	0	0	180	6.9	6.9	too warm	too cold	<45	3763
5	8/25/2008	3:00:00	70	64.9	84	9	0	0	0	0	180	9.2	9.2	comfortable	Total		6720
6	8/25/2008	4:00:00	70	66	87	9	0	0	0	0	160	6.9	6.9	comfortable			
7	8/25/2008	5:00:00	70	64.9	84	9	100	0	0	0	200	8.1	8.1	comfortable			
8	8/25/2008	6:00:00	68	64.9	90	8	25	0	0	0	200	5.8	5.8	comfortable			
9	8/25/2008	7:00:00	69.1	66	90	6	88	0	0	0	200	5.8	5.8	comfortable			
10	8/25/2008	8:00:00	70	66.9	90	5	100	0	0	0	210	5.8	5.8	comfortable			

Figure 3: Class Schedule


A3 — f_x Row Labels

Row Labels	Average of TmpF	Average of DPf
2008	46.17044574	37.36010982
2009	38.31702539	24.74964128
Grand Total	41.93520833	30.55946429

PivotTable Field List

Choose fields to add to report:

- [✓] Date
- [ ] Time
- [✓] TmpF
- [✓] DPf
- [ ] RH
- [ ] Vis
- [ ] CC
- [ ] PcpIn
- [ ] SnFall
- [ ] SnDpth
- [ ] WDir
- [ ] Wind
- [ ] MxWnd

Figure 4: Pivot Table

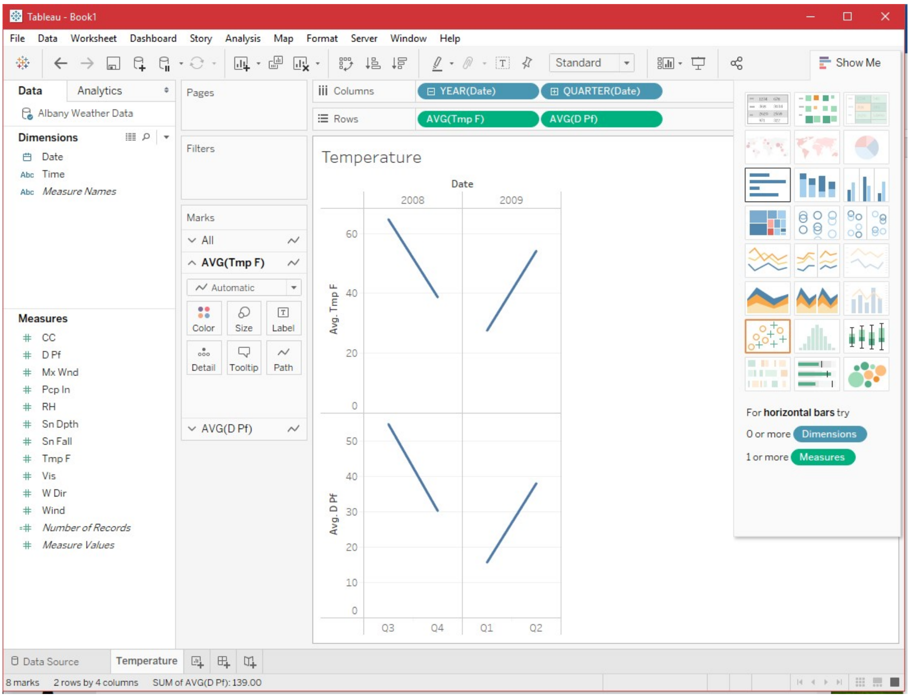


Tableau - Book1

Data source: Albany Weather Data

Columns: YEAR(Date), QUARTER(Date)

Rows: AVG(Tmp F), AVG(D Pf)

Dimensions: Date, Time, Measure Names

Measures: CC, D Pf, Mx Wnd, Pcp In, RH, Sn Dpth, Sn Fall, Tmp F, Vis, W Dir, Wind, Number of Records, Measure Values

Marks: All, AVG(Tmp F), Automatic, AVG(D Pf)

For horizontal bars try 0 or more Dimensions, 1 or more Measures

8 marks   2 rows by 4 columns   SUM of AVG(D Pf): 139.00

Figure 5: Bar Charts Showing Average TmpF and DpF by Quarter

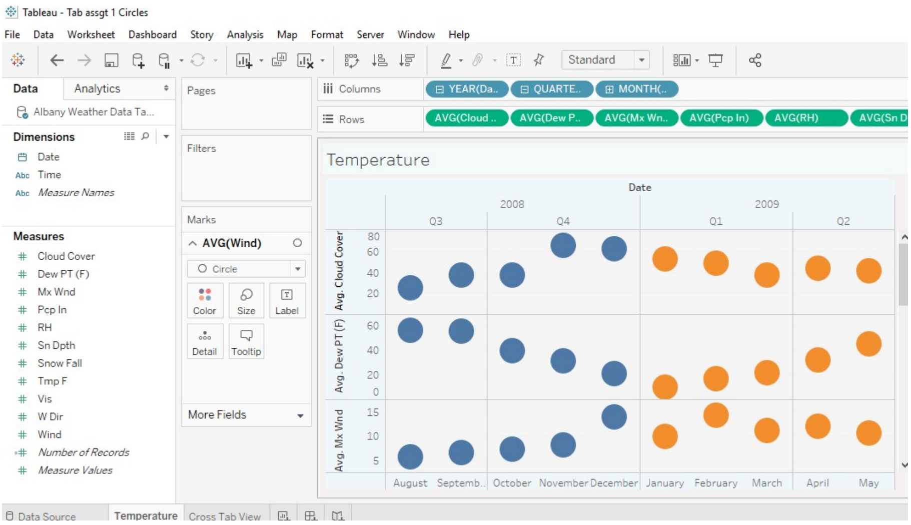


Figure 6: Dot Chart Showing Weather Trends by Year

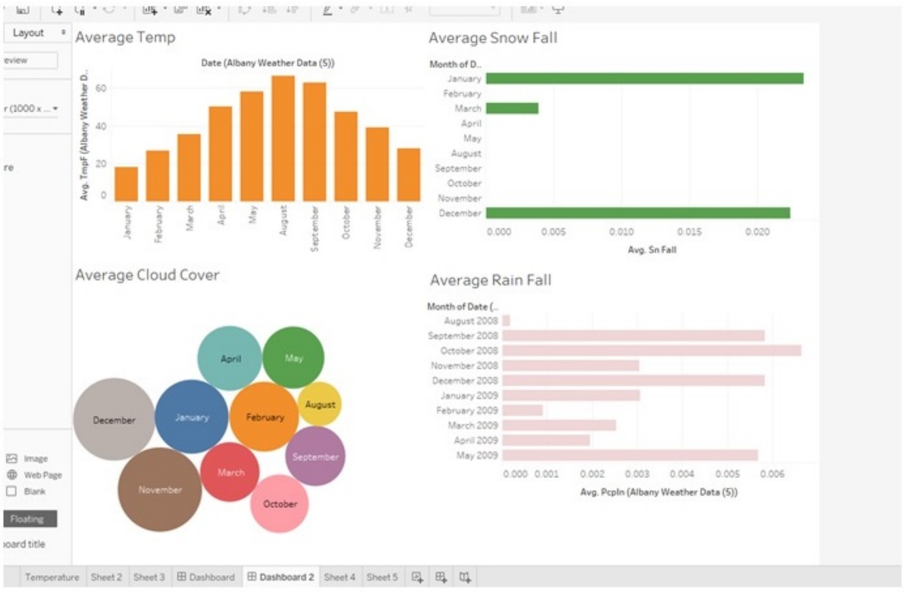
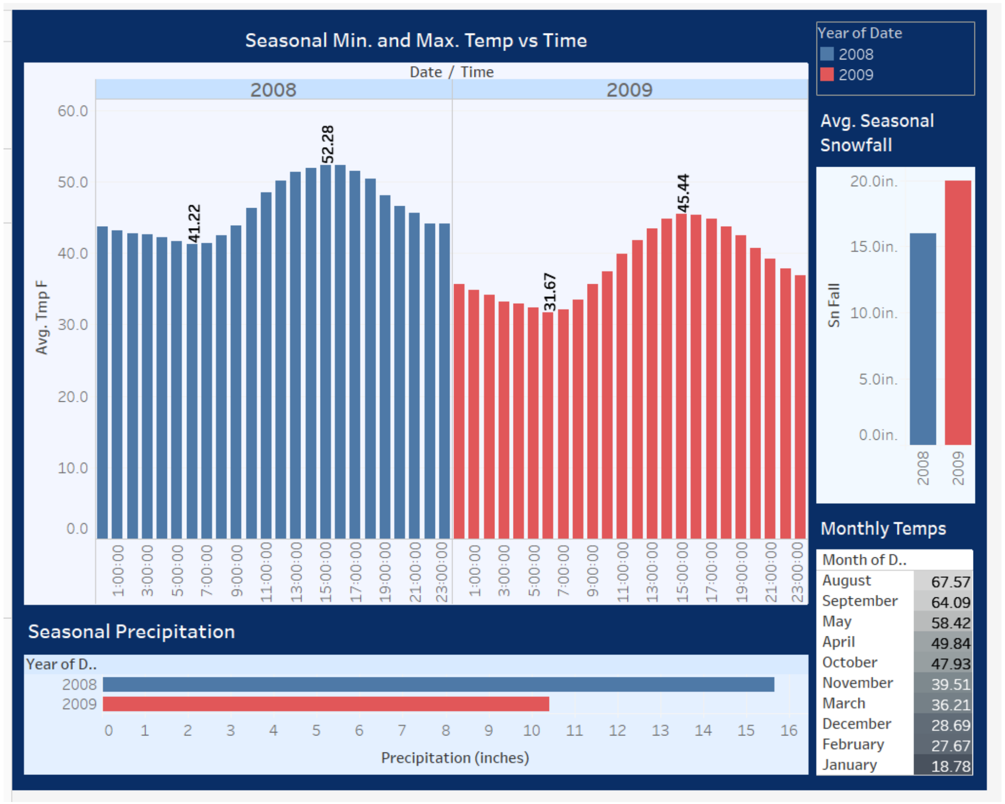
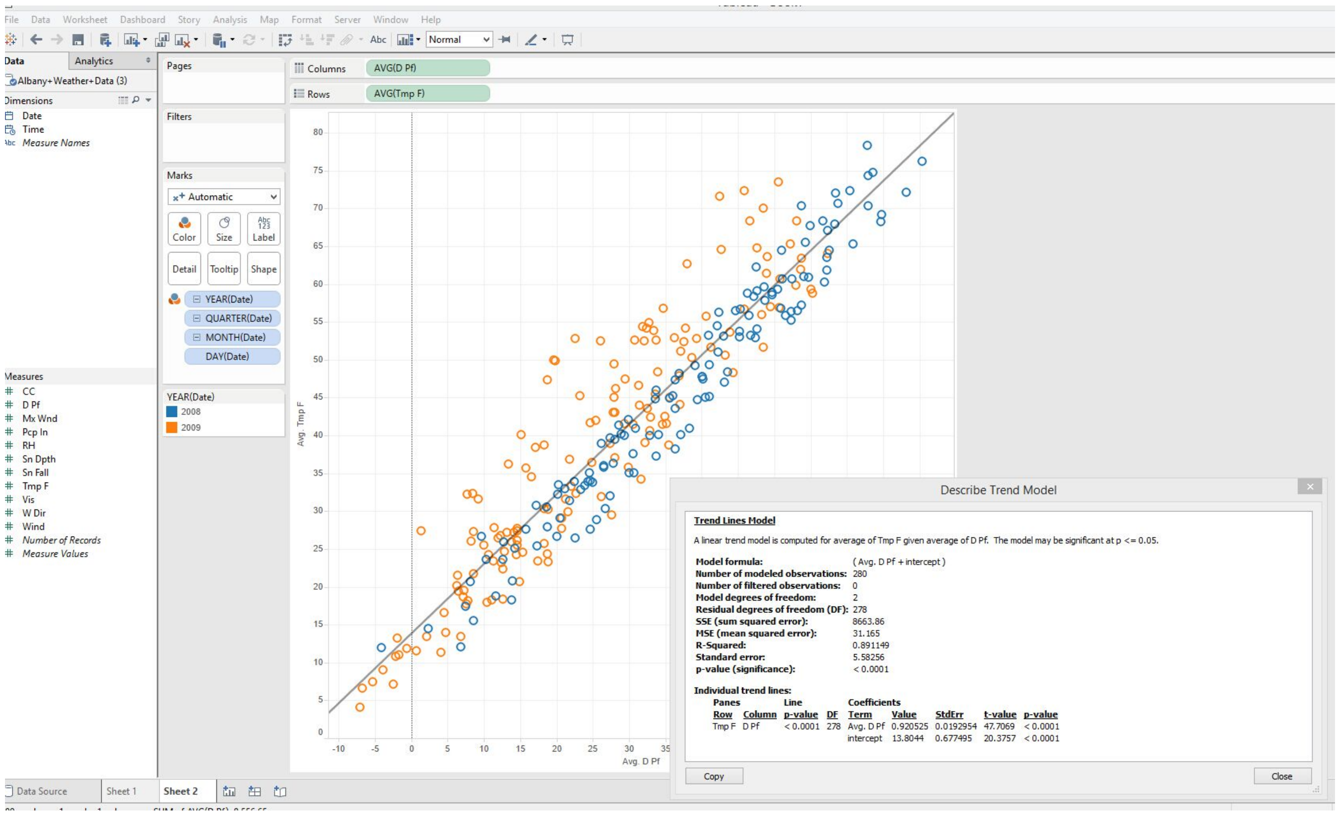


Figure 7: Tableau Storyline Showcasing Graphs Used for Visual Analysis of Data



Month of D..	
August	67.57
September	64.09
May	58.42
April	49.84
October	47.93
November	39.51
March	36.21
December	28.69
February	27.67
January	18.78

Figure 8: Tableau Interface Showcasing Key Weather Trends



Trend Lines Model

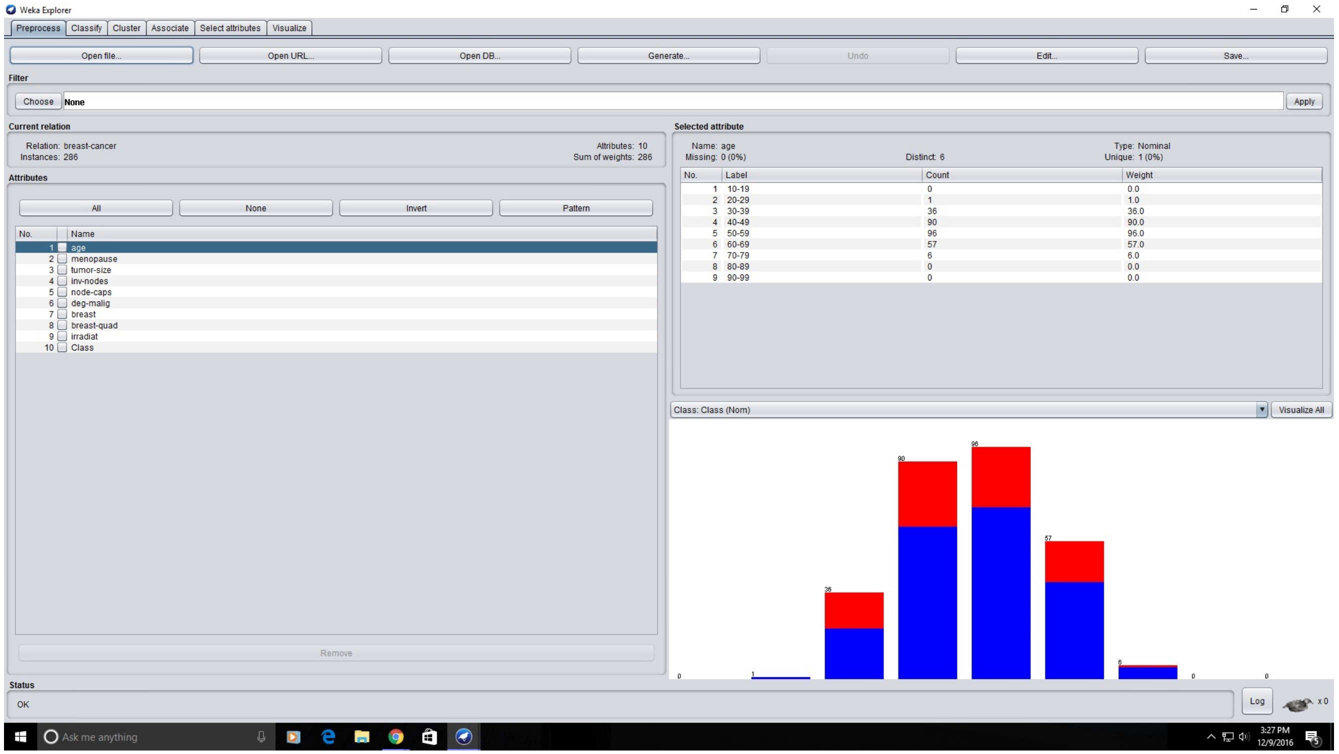
A linear trend model is computed for average of Tmp F given average of D Pf. The model may be significant at p <= 0.05.

Model formula:	(Avg. D Pf + intercept)
Number of modeled observations:	280
Number of filtered observations:	0
Model degrees of freedom:	2
Residual degrees of freedom (DF):	278
SSE (sum squared error):	8663.86
MSE (mean squared error):	31.165
R-Squared:	0.891149
Standard error:	5.58256
p-value (significance):	< 0.0001

Individual trend lines:

Panes		Line		Coefficients				
Row	Column	p-value	DF	Term	Value	StdErr	t-value	p-value
Tmp F	D Pf	< 0.0001	278	Avg. D Pf	0.920525	0.0192954	47.7069	< 0.0001
				intercept	13.8044	0.677495	20.3757	< 0.0001

Figure 9: Class Schedule



Filter

None

Current relation

Relation: breast-cancer
Instances: 286
Attributes: 10
Sum of weights: 286

Attributes

No.	Name
1	age
2	menopause
3	tumor-size
4	inv-nodes
5	node-caps
6	deg-malig
7	breast
8	breast-quad
9	irradiat
10	Class

Selected attribute

Name: age
Missing: 0 (0%)
Distinct: 6
Type: Nominal
Unique: 1 (0%)

No.	Label	Count	Weight
1	10-19	0	0.0
2	20-29	1	1.0
3	30-39	36	36.0
4	40-49	90	90.0
5	50-59	96	96.0
6	60-69	57	57.0
7	70-79	6	6.0
8	80-89	0	0.0
9	90-99	0	0.0

Class: Class (Nom)

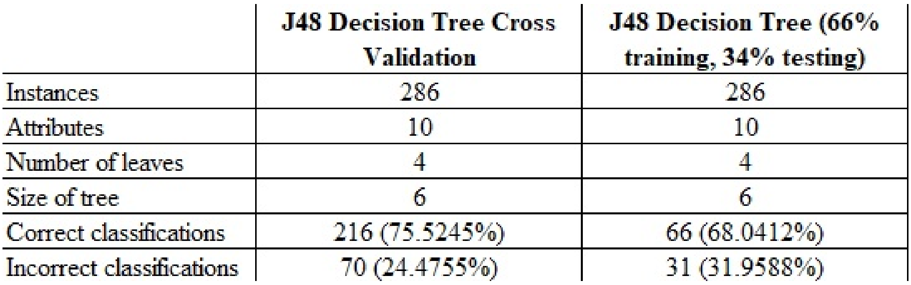
Status

OK

x 0

3:27 PM
12/9/2016

Figure 10: Weka Interface for Data Preprocessing and Supervised Learning



	J48 Decision Tree Cross Validation	J48 Decision Tree (66% training, 34% testing)
Instances	286	286
Attributes	10	10
Number of leaves	4	4
Size of tree	6	6
Correct classifications	216 (75.5245%)	66 (68.0412%)
Incorrect classifications	70 (24.4755%)	31 (31.9588%)

Figure 11: Chart of Decision Tree Output in Weka

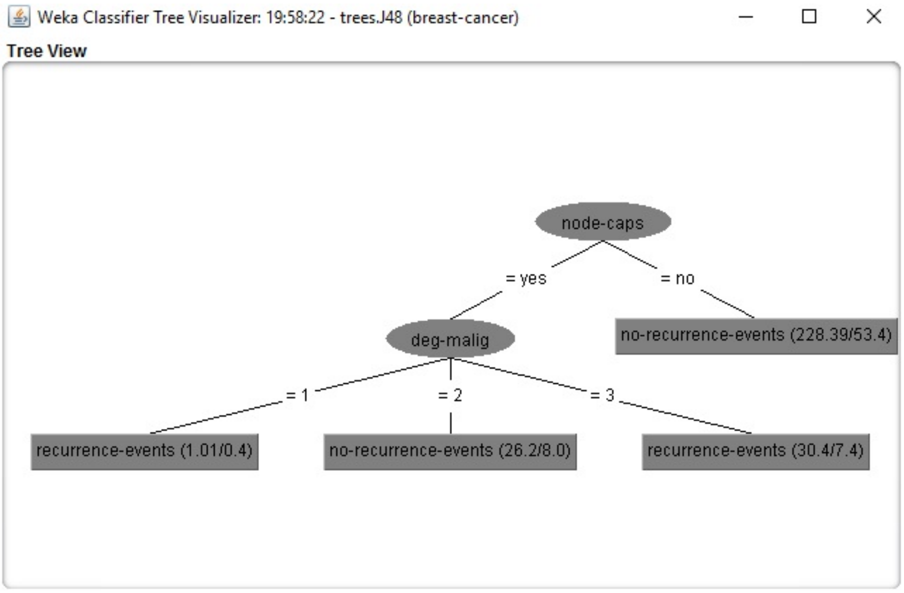
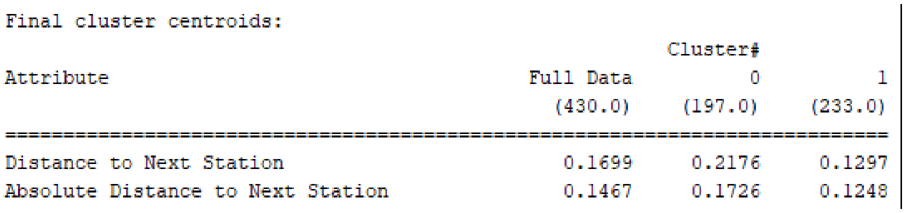

Weka Classifier Tree Visualizer: 19:58:22 - trees.J48 (breast-cancer)
Tree View
node-caps
= yes
= no
deg-malig
no-recurrence-events (228.39/53.4)
= 1
= 2
= 3
recurrence-events (1.01/0.4)
no-recurrence-events (26.2/8.0)
recurrence-events (30.4/7.4)

Figure 12: Decision Tree Visualization in Weka



```
Final cluster centroids:
                                                              Cluster#
Attribute                                          Full Data         0           1
                                                     (430.0)   (197.0)     (233.0)
==================================================================================
Distance to Next Station                              0.1699    0.2176      0.1297
Absolute Distance to Next Station                     0.1467    0.1726      0.1248
```

Figure 13: Cluster Output in Weka

Teaching Database for Freshmen: A Two-Thread Model*

Yang Wang, Margaret McCoey, Thomas Blum
Department of Mathematics and Computer Science
La Salle University
Philadelphia, PA 19141
{wang, mccoey, blum}@lasalle.edu

Abstract

We address the challenge of teaching a database course for freshmen CS/IT majors: namely their limited background, coupled with the material's mixture of language (e.g., SQL) and abstract theory (e.g., normalization) – hard even for upperclassmen. In this paper, we propose a new two-thread model: one thread of theory and another of practice which are relatively independent and self-contained (compared to a typical "Lecture + Lab" design). To provide students extensive exercises on SQL, the practice thread has labs that expose them to SQL statements starting Week 1. When applicable, labs purposely anticipate related components in the theory thread. Also we approach abstract concepts (e.g., normalization) using an example-based method that generalizes patterns extracted from examples. We have applied this two-thread model with our freshmen for the last few semesters, and their responses have been very positive. We share some of this feedback as well as the learned lessons with the hope to enlighten the teaching of database courses and other freshmen courses in other institutions.

1 Introduction

It is formidable to teach introductory major courses since freshmen have a limited knowledge base in their majors at the same time they are adjusting

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

to college life. In particular, teaching a database course [1] for CS/IT freshmen combines low-level intricacies (e.g., SQL syntax) and high-level abstraction (e.g., ER modeling and normalization), each of which is hard to learn on its own. Nevertheless striking the right balance between theory and practice can provide students with a broad overview of the field and expose them early to real-life problem solving.

In this work, we present our design of a freshmen database course. One finds in the literature some work [4, 9] concerning curricular design for CS/IT freshmen; however, none cover the subject of databases. On the other hand, the literature on database pedagogy often focuses on a particular aspect of the material. For instance, there are works on the use of visualization tools (e.g., [1, 2]) to aid the understanding of SQL queries, as well as investigations into the degree of difficulty students encounter learning various types of SQL statements [7]. Alternatively, some works expand on the database topic, such as designing a set of labs to treat database security and auditing [12, 5] or integrating newer technologies (e.g., Cloud-based database [3] and mobile applications [6]). Distinct from the existing literature, we propose an *overall* course design tailored to freshmen that reduces the barriers of SQL-syntax tediousness and database-concept abstractness. We propose a *two-thread* model: one thread of practice and another of theory that are relatively independent yet are presented to the students in an intertwined manner. The practice thread provides an extensive, semester-long exposure to SQL language and database management system. When applicable, we purposely seed the lab experiences with ideas ahead of the related lectures. The practice thread thus illustrates, anticipates, and motivates the theoretical concepts. Moreover, we pursue an example-based approach in the theory thread allowing students to see the patterns and generalize the concepts on their own.

The rest of this paper is organized as follows. Section 2 presents the background, major learning objectives and challenges of the class. In Section 3 we present the detailed approach. In Section 4, we summarize the major principles of the course design. Section 5 discusses the feedback of students and lessons learned, and Section 6 concludes this paper.

2 Background, Objectives and Challenges of the Class

In contrast to most universities, we offer freshmen-level networking and database courses. This curriculum is designed to expose freshmen to a wider spectrum of core computer-science concepts including the binary number sys-

[1]A database course includes areas related to database theory, design, implementation, and management. In the remainder of this paper references to the term database will assume these listed areas.

tem, databases, algorithms, networking, and programming, so they might determine their major and career path earlier. Furthermore, compared to other subjects, database (or networking) is more closely tied to the students' daily lives (e.g., student course registration system). Consequently, it is easier to motivate students to dive into the technologies behind those applications. In addition, teaching database at freshmen level enables a smooth transition to downstream courses such as Open-Source Application Development, and .NET Programming. Another benefit to broadening the introductory sequence is the scheduling of students who change majors, who pick up a CS/IT minor, or who transfer from another institution.

This paper concentrates on our design for the database class. The major objectives of this course are: (i) Present students with an overview of database technologies and concepts; (ii) Prepare students to master skills in Entity Relationship (ER) Modeling, normalization, and query languages; (iii) Prepare students to acquire the ability to create transactional database solutions for real-life problems.

Given the background of our students and the course curriculum, we face multiple challenges in designing this course. First, as a freshmen course, balancing the depth, complexity, and interests in the topics covered requires care – especially for students with little to no CS/IT background. The course design should avoid creating frustration in the freshmen transition period yet still prepare them for downstream courses. Second, as a critical skill, mastering the SQL language demands extensive and repeated exercises (especially for freshmen). Yet in a traditional database class, SQL is introduced only after covering most of the basic database theories. Third, one must cultivate careful examples to illustrate database concepts to students less practiced in abstraction. Though database ideas appear to be straightforward (e.g., a table for employees), the mathematical and logical foundations for databases (e.g., relational algebra, normalization) are known to be abstract and hard to comprehend. Lastly, given the breadth of database topics, scrutiny must be given to selecting a subset of them that can: provide an overview of the field to students; expose them to state-of-the-art technologies; and prepare them with ability to model real-life scenarios.

3 Course Design

In this section, we present the detailed design of the database course.

3.1 The Two-Thread Model

We refer to our design philosophy as a "Two-Thread" Model: with theory and practice threads. Note this method is distinct from a typical CS/IT "Lecture

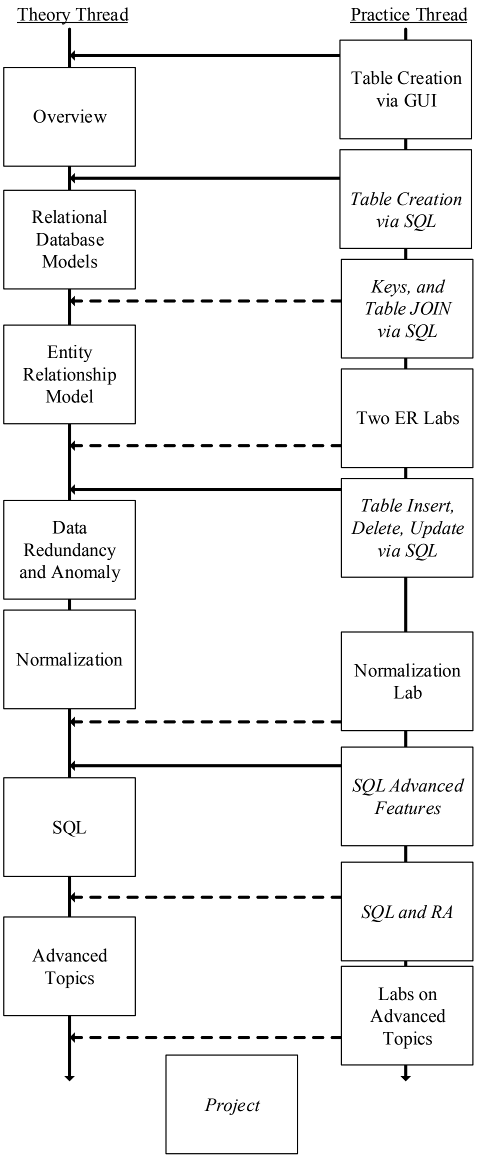


Figure 1: Two-Thread Model

+ Lab" where labs solidify the understanding of theories already presented in the lecture [10, 11]. On one hand, our practice-thread labs are relatively independent and contain a considerable amount of self-learning knowledge (mainly on SQL). On the other hand, our labs often prepare the way for the related theories.

Figure 1 depicts the major components of the two-thread model. The theory thread covers topics including overview of database technologies, relational database models, ER modeling, data redundancy and anomaly, normalization, SQL, and advanced database topics in sequence. The practice thread covers practices on table creation via GUI, table creation via SQL, keys and table JOIN operations via SQL, ER modeling exercises, table insert, delete and update via SQL, normalization lab, SQL advanced features, Relational Algebra (RA) related SQL statements, and labs for advanced database topics in order. The vertical lines in Fig. 1 show the connections between theory modules and the related practice-thread exercises. A solid vertical line indicates that the lab exercise is placed ahead of the related theory. Note that we introduce SQL at an early stage to allow a semester-long extensive practices on SQL. The detailed list of labs are shown in Table 1 which is further elaborated below along with the discussion of related modules of the theory thread.

3.2 Modules

Overview: This module starts with a lab exercise (i.e., Lab 1 in Table 1) before addressing abstract concepts. A SQL script was prepared to create a user and database for each student in MySQL before the semester begins, and students are guided to create tables via the GUI interface of PhpMyAdmin. After the lab exercise, students are introduced to basic concepts and technologies in database including the functions, history, and classification of databases.

Relational Database Model: Module 2 also starts with a lab (Lab 2 in Table 1) that instructs students to create tables using SQL statements – the aforementioned early exposure to SQL. Also, students are led to consider the relationships among the entities/tables. After the lab, important concepts of relational database including table, dependency, keys (primary, composite, foreign, candidate, super, and secondary), relationships (1:1, 1:M, and M:N), integrity rules, and relational algebra (RA) are introduced to students. This module ends with another exercise (i.e., Lab 3 in Table 1) that revisits the prior lab with the re-consideration of primary key/foreign key and the introduction the SQL statement for table JOIN (which helps in validating the referential integrity).

ER Modeling: This module introduces ER modeling with the emphasis on how relationships (1:1, 1:M, and M:N), connectivity, cardinality, relationship strength, and participation are embodied in ER modeling. The lecture part

Table 1: Labs Associated with Each Module

Module	**Lab**	**Software**	**Skills Covered**
1	1. Introduction to MySQL	MySQL, phpMyAdmin	Familiar with phpMyAdmin, table creation, insertion, and view (through GUI)
2	2. Create Database/Tables with SQL Statements	MySQL, phpMyAdmin	SQL statements for creating tables, and relationships
2	3. Keys and Integrity	MySQL, phpMyAdmin	Familiar with primary/composite/foreign keys and the implications of entity/referential integrity
3	4. ER Modeling in Visio	Microsoft Visio	ER modeling in Chen's notation and Crow's foot notation (in Microsoft Visio)
3	5. ER Modeling: From model to implementation	MySQL (Workbench)	Familiar with the process from database design to implementation
4	6. Data Redundancy and Anomaly	MySQL, phpMyAdmin	Familiar with SQL statements for insert/delete/update and associated anomalies
5	7. From NF1 to NF3	MySQL, phpMyAdmin	Familiar with the normalization process from NF1 to NF3
6	8. SQL Statements	MySQL, phpMyAdmin	Concepts of index, constraints, and advanced SELECT statements
6	9. SQL Statements and Relational Algebra (RA)	MySQL, phpMyAdmin	Review of RA and associated SQL statements (SELECT, JOIN, UNION, INTERSECT, EXCEPT)
7	10. Transactions and Deadlock	MySQL, phpMyAdmin	SQL Lab based on a shared set of tables and procedures that go into a wait state
7	11. Security and Privacy	MySQL, phpMyAdmin	Gaining access to database through invalid data
7	12. NoSQL Database	MongoDB	Familiar with basic concepts in NoSQL database
7	13. Cloud-based Database	Amazon AWS	Familiar with basic concepts in Cloud-based database

Table 2: A Motivation Example for Data Redundancy and Anomaly

<u>SID</u>	**FName**	**LName**	**PhNO**	**DmNO**	**DmLoc**
089	Allen	Aversion	404-123-3421	8	location 1
076	Curry	Charles	404-334-7892, 707-676-7651	11	location 2
023	David	Davenport	678-453-3214	8	location 1

Table 3: Intuitive Solution

<u>SID</u>	**FName**	**LName**	**PhNO1**	**PhNO2**	**DmNO**	**DmLoc**

covers both Chen's and Crow's foot notations. As opposed to using complex data [8] in our example-based approach, we use simplified examples to ensure the sole difficulty lies in the mapping of problem-to-model rather than the problem's inherent complexity. Two exercises (i.e., Labs 4 and 5 in Table 1) are adopted in this module: the first uses Microsoft Visio and a real-life case study; the second uses MySQL Workbench and takes ER modeling to table-creation.

Data Redundancy and Anomaly: Module 4 starts with an exercise (Lab 6 in Table 1) on SQL statements for insert, delete and update, followed by failing attempts of such operations on tables with associated anomalies. It prepares students for lectures on data redundancy and three types of data anomalies. Again, we use an example-based approach to explain redundancy and anomaly. The example shown in Table 2 records student data including ID (SID), first and last names (FName and LName), phone numbers (PhNO), dormitory number (DmNO) and location (DmLoc). By examining the redundancy and anomalies in this example, the ultimate goal is to establish the connection between data redundancy and anomalies as well as prepare students for the concept of normalization.

Table 4: NF1

<u>SID</u>	**FName**	**LName**	**<u>PhNO</u>**	**DmNO**	**DmLoc**
089	Allen	Aversion	404-123-3421	8	location 1
076	Curry	Charles	404-334-7892	11	location 2
076	Curry	Charles	707-676-7651	11	location 2
023	David	Davenport	678-453-3214	8	location 1

Normalization: Normalization is one of the most abstract concepts in database design. With our freshmen, we avoid starting it from a conceptual

SID	**FName**	**LName**	**DmNO**	**DmLoc**

Table 5: NF2-Student

SID	**PhNO**

Table 6: NF2-Phone

SID	**FName**	**LName**	**DmNO**	**DmLoc**

Table 7: NF3-Student

DmNO	**DmLoc**

Table 8: NF3-Dorm

perspective. Instead, based on a table with anomalies, we guide them toward a common-sense solution that removes the unnecessary dependencies. Generalizing this process yields normalization. For instance, to explain NF1 to NF3, we resume the discussion of Table 2, which fails to meet the NF1 requirements due to the non-atomic value for phone numbers. From a practical perspective, students are made to realize the issue it causes – the hardness in operations (e.g., query) on phone numbers. One intuitive solution is to create two or more attributes to record multiple phone numbers as shown in Table 3, which, however, creates scalability issues (e.g., a student with three phones) or null values (i.e., a student with only one phone). Theoretically, this intuitive solution fails to meet NF1 due to repeated groups (i.e., multiple attributes for phone numbers). Further discussion leads to the solution in Table 4 (that meets NF1 rules)[2]. Continued discussion on the insert/delete/modify anomalies in Table 4 leads to Tables 5 and 6 (that meet NF2 demands) after "breaking" the table to remove redundancy. Further pursuit in removing the redundancies in Table 5 results in Tables 7 and 8 (that meet NF3 requirements). Via this process of applying the logic of normalization to an actual problematic table, students are able to comprehend the concepts of NF1 to NF3, and generalize the methodology for normalization. This module ends with a lab where students need to normalize a table from NF1 to NF3 (i.e., Lab 7 in Table 1).

SQL: Module 6 starts with a lab on advanced SQL features such as constraints. After the lab, we formally introduce SQL and cover features including index, constraint, functions (including aggregation), views, advanced SELECT statements and RA-related SQL statements (i.e., JOIN, UNION, INTERSECT, MINUS). This modules ends with a lab on RA-related statements connecting them with the RA operators covered previously.

Advanced Database Topics: Module 7 has some built-in flexibility depending on student composition. It includes concurrency and deadlock, security and privacy, and new advances such as NoSQL and Cloud technologies. For concurrency/deadlock, students partake in a paper explanation and complete an SQL lab (Lab 10) using a shared table to demonstrate the concepts. For security/privacy, students watch videos that describe database security holes,

[2]The primary key of each table is underlined hereafter.

and then complete a lab (Lab 11) demonstrating intrusion into the system. The lab also includes a discussion of ethical issues involving access to personal information. In addition, we cover the basic concepts of NoSQL and that is followed by a lab based on mongoDB (Lab 12). For Cloud technologies, the concepts are explained along with a lab based on Amazon EC2 (Lab 13).

3.3 Project

The project is another vital component of our design, where students are asked to apply major skills (e.g., ER modeling, normalization, and SQL) to create and implement a solution to a real-life problem (in a DBMS other than MySQL). This design allows students to apply discrete skills that they learned in an integrated manner, to become familiar with the life cycle of database development, and to obtain exposure to a secondary DBMS (of their own choice).

4 Design Principles

In this section, we summarize the major design principles of this course.

First, given the background of freshmen and the diverse topics in database, we carefully select a subset of topics that present the big picture of the field. The covered topics are fundamental, and *just-enough* to prepare students for downstream classes and self-learning.

Second, we adopt an innovative two-thread approach toward the theory and practice of databases. We place labs ahead of the related lectures (when possible) to seed students with hands-on experience. It makes use of SQL's straightforwardness by arranging self-learning SQL labs from the start. Upon reaching the actual SQL lectures, students already have extensive SQL experience.

Third, with freshmen in mind, we avoid frustration by making the course design "practical, relevant and simple". Within the theory thread, we proceed by generalizing abstract concepts from simplified examples and/or utilizing labs to prepare students in advance. Within the lab thread, we incorporate another strategy: we allow solutions that violate the best-practices of database design in earlier labs while gradually guiding students to meet common standards.

Lastly, a vital component of our design is a term project that allows students to apply all the discrete skills in an integrated manner. This project also exposes students to the life cycle of database design and prepares them to model and implement a database solution to a real life scenario.

5 Feedback and Lessons

Over multiple semesters of teaching this class using above design, the averaged evaluations for this course rated it as 92.5% (=4.629/5) in terms of the overall value that it has contributed to learning. Over 58% of the surveyed students considered this course as "very valuable" (i.e., the highest rating). Some representative feedback on various aspects of this class include:

1. *"This course gives me confidence in the computer database area, and also better understanding in the computer programming field."*
2. *"Learned a lot about database management: normalization, SQL etc."*
3. *"Fun to take, very engaged, a good foundation for database management."*
4. *"... I can develop a database not only on paper; but on a computer."*
5. *"Easy to understand, very useful."*

In addition, we share some lessons in our trials and errors that led to the final design above. First, we note that the idea of placing the lab ahead of lecture is a result of various trials. In classes such as networking, we arrange the lab exercise after a lecture having students to the concepts (e.g., the subnet-masking lab after the subnet-masking lecture). With databases, however, we found that the straightforwardness of basic SQL allows students to pick up new statements on their own. Hence we could incorporate SQL-related labs from the semester start, and/or place labs ahead of the lectures. Second, for abstract concepts (e.g., normalization), we experimented with two teaching methods: 1) teaching the concepts (e.g., NF1, NF2, NF3) followed by exercises; and 2) generalizing concepts from examples. The latter approach led to better learning outcomes as reflected in Assignments and Tests. The difference could be partially attributed to the students' background (i.e., freshmen). Third, the project originally optional (as bonus) was later made mandatory as we observed (in both surveys and in learning outcomes) that students who completed the project obtained more confidence and satisfaction upon applying all the learned skills to create their own database solutions.

6 Conclusion and Future Work

In this paper, we present our design for a freshmen database class. Though we have modules on recent advances, the key to our design lies more than the novelty of topics: we select a subset of topics that covers the essential aspects of the field for freshmen; we adopt a two-thread model that provides students with a semester-long exposure to SQL; to supplement students with hands-on experiences, we also occasionally place the lab practices ahead of the respective theory lectures. Our teaching in past few semesters with this design has led to satisfactory feedback from students. We are closely monitoring the students' performance in downstream classes to ensure that this course provides solid preparations.

References

[1] Maurizio Cembalo, Alfredo De Santis, and Umberto Ferraro Petrillo. Savi: A new system for advanced sql visualization. In *Proceedings of the 2011 Conference on Information Technology Education*, SIGITE '11, pages 165–170, 2011.

[2] Ryan Hardt and Esther Gutzmer. Database query analyzer (dbqa): A data-oriented sql clause visualization tool. In *Proceedings of the 18th Annual Conference on Information Technology Education*, SIGITE '17, pages 147–152, 2017.

[3] Edward P. Holden, Jai W. Kang, Geoffrey R. Anderson, and Dianne P. Bills. Databases in the cloud: A status report. In *Proceedings of the 2011 Conference on Information Technology Education*, SIGITE '11, pages 171–176, 2011.

[4] Päivi Kinnunen, Maija Marttila-Kontio, and Erkki Pesonen. Getting to know computer science freshmen. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, pages 59–66, 2013.

[5] Lei Li, Kai Qian, Qian Chen, Ragib Hasan, and Guifeng Shao. Developing hands-on labware for emerging database security. In *Proceedings of the 17th Annual Conference on Information Technology Education*, SIGITE '16, pages 60–64, 2016.

[6] Qusay H. Mahmoud, Shaun Zanin, and Thanh Ngo. Integrating mobile storage into database systems courses. In *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, pages 165–170, 2012.

[7] Toni Taipalus and Piia Perälä. What to expect and what to focus on in sql query teaching. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 198–203, 2019.

[8] Paul J. Wagner, Elizabeth Shoop, and John V. Carlis. Using scientific data to teach a database systems course. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '03, pages 224–228, 2003.

[9] Yang Wang, Thomas Blum, and Margaret McCoey. Teaching a networking class for freshmen: Course design and lessons learned. In *Proceedings of the 15th Annual Conference on Information Technology Education*, SIGITE '14, pages 9–14, 2014.

[10] Yang Wang, Thomas Blum, and Margaret McCoey. Teaching network administration in the era of virtualization: A layered approach. In *Proceedings of the 18th Annual Conference on Information Technology Education*, SIGITE '17, pages 97–102, 2017.

[11] Yang Wang, Margaret McCoey, and Heng Zou. Developing an undergraduate course curriculum on information security. In *Proceedings of the 19th Annual SIG Conference on Information Technology Education*, SIGITE '18, pages 66–71, 2018.

[12] Li Yang. Teaching database security and auditing. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, SIGCSE '09, pages 241–245, 2009.

Integrative Learning in CS1: Programming, Sustainability, and Reflective Writing*

Jeffrey A. Stone[1], *Laura Cruz*[2]
[1]*Information Sciences and Technology*
Penn State University
Center Valley, PA 18034
stonej@psu.edu
[2]*Schreyer Institute for Teaching Excellence*
Penn State University
State College, PA 16802
lxc601@psu.edu

Abstract

Computer Science and related disciplines produce artifacts which touch virtually every aspect of modern life, yet assignments in CS1-level courses are often limited in social engagement. Programming assignments focused on sustainability concepts offer an opportunity for demonstrating the applicability and social relevance of computing. This article reports on a year-long, integrative learning study which uses sustainability-themed programming projects in introductory programming courses. The results of this mixed-methods study suggest that while students perceive the course and its programming assignments as beneficial in their understanding and appreciation of sustainability concepts, students were limited in their ability to transfer the knowledge obtained into both personal and community contexts.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Computer Science (CS) and related disciplines produce artifacts which touch virtually every aspect of modern life, yet assignments in introductory programming courses are often narrow in scope and limited in social engagement. Most introductory, CS1-style courses focus on building core programming skills, often drawing on established problem sets from related disciplines. An alternative method is to construct projects and course content which involve socially relevant contexts. Sustainability is one context that is both universally applicable and offers a "real world" environment for demonstrating the impact of computing. Sustainability entails informed decision-making about the environmental and societal impacts of alternative choices, for both individuals and organizations. Sustainability is a critical element of professional responsibility as well as personal behavior. Professional organizations such as the ACM and IEEE codify expectations of sustainability in their codes of ethics [6] and in computing curricula guidelines [14].

Sustainability has great potential for demonstrating the wider applicability of computing. By constructing programming assignments themed around sustainability concepts, introductory students have the opportunity to see how computing can be used to address a set of multidisciplinary, socially relevant problems. "Computing for the social good" has been seen as a way to both engage introductory students and to increase the diversity in CS student enrollments [15]. Providing a multidisciplinary context to computing assignments has also been shown to increase success and recruitment rates for women and underserved populations [7, 12]. By providing students with basic information on critical environmental topics, sustainability-themed projects allow students to draw integrative connections between sustainability, the practice of programming, and their own communities, lives, and behaviors.

This article describes a year-long integrative learning approach to educate introductory students about the social relevance of programming. This study was intended to provide increased student engagement, to educate introductory computing students about sustainability, and to encourage students to integrate sustainability into daily practices and decisions. The study uses a mixed-methods approach to assess the impact of a series of sustainability-focused programming projects on students' perceptions, practices, and behaviors. The goals of this study were to be accomplished with only minor revisions to the existing course.

2 Literature Review

2.1 Sustainability in CS Education

Despite the recognition of sustainability as an important element of both curricula and practice, the CS Education literature has few examples of pedagogical approaches which integrate sustainability. Abernethy and Treu [1] described efforts in both the introductory and upper division courses to build students' awareness of the role of computing in sustainability, including discussions of carbon footprints, power consumption, and e-waste. Erkan, Pfaff, Hamilton, and Rogers [5] used a set of sustainability-themed projects for a Data Structures course to educate students about both data structures and algorithms (theory) and how those tools can be used to answer important sustainability questions (application). Cai [4] and Hamilton [8] described a series of efforts to integrate so-called "green computing" content into the curriculum, as a course, a module, and as a source of senior design projects. Recognized barriers to integrating sustainability concepts include concerns over faculty knowledge, accessible resources, and competing curricular priorities [4, 13].

2.2 Integrative Learning

The use of sustainability-themed projects in introductory programming courses is about integrative learning. *Integrative learning* is learning that allows students to see cross-disciplinary connections for their knowledge and to make more educated judgments and decisions (i.e. is interdisciplinary) [9, 10]. More than just synthesizing related knowledge and perspectives from multiple disciplines, integrative learning also involves knowledge and perspectives obtained through cultures, subcultures, and life experience [11]. The integration of sustainability into computing curricula allows for interdisciplinary considerations of computing use and application to be explored, providing a fertile platform for integrative learning.

2.3 Reflective Writing

One method for assessing integrative learning outcomes is asking students to critically assess their learning experiences. Reflective writing provides an opportunity for students to build metacognitive skills, i.e. self-awareness and understanding of their own learning processes [17]. Reflective writing therefore provides opportunities for building critical thinking and information synthesis skills [3, 19]. For university students, reflective writing offers a significant step towards transforming academic knowledge into informed action.

In CS Education, reflective writing has been used to build student engagement [2], as a mechanism for formative and summative course feedback

[17], and as a means of building metacognitive skills in students [19], among others. This study uses reflective writing to assess integrative learning outcomes, specifically the impact of sustainability-themed programming projects on student perceptions, practices, and behaviors. Reflective writing provides an outlet for students to convey the movement from simple knowledge acquisition to applications both programming-related (e.g. write a simple program) and in a broader context (e.g. recognizing community applications).

3 Methodology

The research study was carried out in two course sections over two semesters (2018-2019). One of the sections was a CS1-style introductory Java course for Information Sciences and Technology (IST) majors while the other course was a CS1-style introductory C++ course for Engineering majors. All research procedures were approved by the Penn State Office of Research Protections.

3.1 Sustainability-Themed Assignments

The pedagogical approach used in this study is a modified form of the approach found in [18]. A series of eight (8) programming projects were used, each of which was focused on a specific sustainability topic. The projects were structured to provide a sustainability context for the assessment of programming skills, though an expected secondary benefit was also the acquisition of knowledge about specific sustainability problems and concepts. See Table 1 for a list of topics. The projects themselves - including project directions and sustainability resources - are available at https://sites.psu.edu/sustainabilitycis/.

Each set of project directions employed a similar structure. Each project was focused on one of the 17 UN goals for sustainable development. The directions began with a brief description of the specific UN goal, followed by a brief (1-2 paragraph) introduction to the sustainability topic. This brief introduction included a short, pre-existing YouTube video on the topic. Following the introduction, the specific programming problem was introduced along with test cases. These integrative projects allow students to see the applicability of computing in the context of a diverse, engaging, and socially relevant problem domain, while still focusing on the traditional, programming skill-focused learning outcomes of these CS1-style courses.

3.2 Survey

A custom post-test survey was delivered to each course section during the last week of the semester. The survey included questions on students' perceptions of the impact the sustainability-themed programming assignments had on their

Table 1: Project Topics and Skills

Project Topic	Skills Assessed
Wind Power	Basic I/O, Arithmetic
Good Health	If Statements
Air Pollution	Cascading If
Water Catchment Systems	While loops
Trees and Carbon	For loops
Sustainable Planting	Functions/Methods
Sustainable Greenspace	Arrays
Ocean Acidification	File I/O, Arrays, Structs/Classes

sustainability knowledge and practices. The survey was delivered electronically through Qualtrics and responses were analyzed using SPSS.

3.3 Reflective Writing Assignments

At the conclusion of each project, each student was required to complete a reflective writing assignment of approximately 200-500 words. These follow-up writing assignments were low stakes exercises, taken together worth far less to the final grade (5%) than the programming assignments themselves (35%). The combination of programming assignment and post-reflection represents the integrative learning activities for this study.

These reflective writing assignments provided directions on the meaning of reflection, as well as formatting expectations and a series of 4-5 prompts for students to address. The instructor-provided prompts focused on the sustainability topic referenced in the programming project, and included assignment-specific questions related to these three dimensions:

- What did you learn about the sustainability topic? (*Content Knowledge*)
- How could the topic be integrated in your own community, and how could you begin that process? (*Applications*)
- How did you integrate sustainability into your life during this project, and did the project encourage you to consider new ways to integrate sustainability into your daily life? (*Practices*)

The student reflections for each of the eight projects were de-identified and their content analyzed by a team of five independent, external raters using a modified version of the framework analysis method [16]. Each paper was rated using a custom rubric which rated the submission on the aforementioned three dimensions – *Content Knowledge, Applications*, and *Practices* – using a four-level scale for each (4=Comprehensive, 3=Satisfactory, 2=Limited, 1=Incomplete).

4 Results

A total of 16 students (80.00%, N=20) participated in the research study, though not all respondents chose to participate in all data collection steps. A total of 120 reflective writing submissions were collected from the participants.

4.1 Survey Results

A total of 14 students (87.50%, n=16) completed the post-test survey. Respondents were primarily male (92.86%, n=14), in the 18-30-year-old range (100.00%), and White/Caucasian (92.86%). A majority of respondents reported being second-year students (71.43%, n=14). In order to assess the perceived impact of the course and its assignments, participants were asked to rate their level of agreement with a series of statements using a five-level Likert-style scale (1=Strongly Agree, 2=Agree, 3=Neutral, 4=Disagree, 5=Strongly Disagree).

Most respondents reported that the course and its assignments positively impacted their understanding of sustainability. A majority of respondents agreed/strongly agreed with the statement, *Because of this course, I have a greater understanding of the basic ideas and concepts behind sustainability* (85.71%, n=14). A slightly smaller majority of respondents agreed/strongly agreed with the statement, *Assignments and activities for this course enhanced my understanding of sustainability* (64.29%, n=14).

Respondents also responded the course helped them see the applications of sustainability in their own community. A majority of respondents agreed/strongly agreed with the statement, *Because of this course, I can see the potential application(s) of sustainability practices in my own community* (57.14%, n=14). A majority of respondents also agreed/strongly agreed with the statement, *Assignments and activities for this course helped me to see the potential application(s) of sustainability practices in my own community* (64.29%, n=14).

Recognition of the wider applicability of programming was also perceived to be impacted by the course. A majority of respondents agreed/strongly agreed with the statement, *Because of this course, I understand the applicability of programming to solve complex social problems* (64.29%, n=14). A majority of respondents also agreed/strongly agreed with the statement, *Assignments and activities for this course helped me to see the applicability of programming to solve complex social problems* (71.43%, n=14).

Participants were asked to describe three ways in which their behaviors or actions regarding the environment and/or sustainability were altered by the course. Five responses (41.67%, n=12) indicated the course content had no impact on their sustainability-related behaviors or actions. Of the remaining seven responses, most referenced an increased attention to water use and/or the use of water catchment systems (41.67%, n=12), an increased desire to be involved in personal planting/gardening (33.33%), and an increased attention to electricity use, such as turning off unused lights (33.33%). The following response is an example:

...This course has taught me to better understand my role in terms of sustainability throughout my daily life...I am using my utility bills to better understand my energy and water usage and have a goal each month to lower my usage. I now look for better ways to involve my time within the community to volunteer for "clean up" events as the season changes.

4.2 Reflective Writing Ratings

The 120 reflective writing submissions resulted in 433 sets of ratings among the five raters (276 for the C++ course, 167 for the Java course). Preliminary analysis led to the removal of 14 outlier ratings. Descriptive statistics for the 419 remaining sets of ratings are provided in Table 2.

Table 2: Descriptive Statistics for Reflections

Rating Dimension	N	Mean	Median	SD
Content Knowledge	419	3.60	4.00	0.58
Applications	419	3.04	3.00	0.79
Practices	417	2.83	3.00	0.88
Total Score (Mean)	380	3.14	3.25	0.45

Participating students were, on average, rated as satisfactory or higher in each dimension, with the highest ratings for the *Content Knowledge* dimension (the acquisition of sustainability knowledge) and lowest for the *Practices* dimension (application of that knowledge towards personal behaviors). Examination of the boxplot (Fig. 1) shows that *Applications* had much more variability in scores than the other dimensions. The *Practices* dimension shows the lowest mean scores, with 75% of the *Practices* scores at or below satisfactory.

Independent Samples t-Test analyses detected significant differences between the two courses along each of the three dimensions - *Content Knowledge* ($t = -2.365$, df=350, $p < 0.05$), *Applications* ($t = -2.337$, df=372, $p < 0.05$), and *Practices* ($t = -3.942$, df=375, $p < 0.01$) - as well as for total mean score ($t = -4.882$, df=310, $p < 0.01$). In all dimensions, ratings for the C++ course

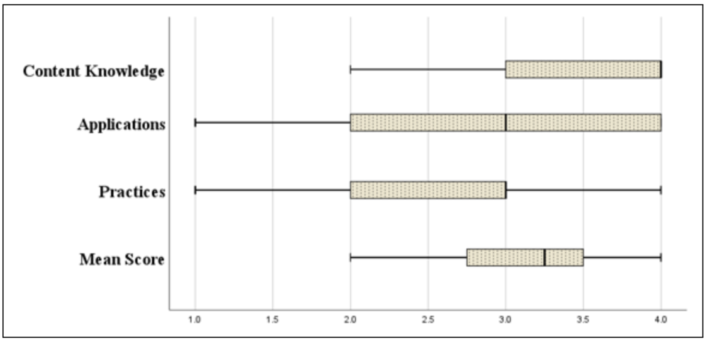


Figure 1: Boxplot of Reflective Ratings

(Engineering majors) were significantly lower than for the Java course (IST majors). The smallest difference between semesters was found in the *Content Knowledge* dimension (0.14) with the largest difference in the *Practices* dimension (0.33). See Table 3.

Table 3: Reflection Rating Means by Course

Rating Dimension	C++ Course (Mean)	Java Course (Mean)
Content Knowledge	3.55	3.69
Applications	2.97	3.15
Practices	2.71	3.04
Total Score (Mean)	3.06	3.28

5 Discussion

The results suggest students perceived that the course and its programming assignments aided their understanding of sustainability, potential applications of sustainability in their community, and the applicability of programming to complex social problems. However, reported changes in students' sustainability-related behavior (i.e. personal applications of sustainability) were relatively light, as indicated by both the open-ended survey comments and the *Practices* ratings for the reflective writing assignments.

The significant differences between courses in all three dimensions – *Content Knowledge*, *Applications*, and *Practices* – was somewhat surprising. It seems reasonable to expect that Engineering students would have a greater familiarity and comfort level with sustainability information than computing majors, especially since the C++ course used in this study is intended for second-year Engineering students. However, in all cases, the Engineering students were rated at a lower level than their computing student counterparts. It may be that prior knowledge was a differentiator; future research will attempt to uncover the impact of prior (pre-course) knowledge on the desired outcomes.

The overall decline in mean ratings for the *Applications* and *Practices* dimensions, as compared to the *Content Knowledge* dimension, may suggest a difficulty in moving from a single application of the sustainability concept (e.g. write a program to compute the power output of a wind turbine) to larger, community applications and personal practices (e.g. how could wind power benefit my community?) Students may be struggling to move from lower-order thinking (i.e. memorization) towards higher order integration and application skills. Future research will examine the potential barriers for introductory students to translate program-contextual knowledge into larger, integrative applications.

6 Conclusion

The purpose of this article was to describe year-long integrative learning approach to educate introductory computing students about the social relevance of programming. Sustainability-themed programming assignments were used to engage and enlighten students about a topic of universal importance, providing a means for students to see the greater applicability of computing and programming in general. Through both surveys and reflective writing assignments, the authors hoped to uncover if students were able to integrate the information obtained from programming projects into a greater context, i.e. potential community applications and changes in personal practices. The survey results suggest that students perceive the programming assignments and the course as beneficial in their understanding and appreciation for sustainability concepts and applications, though the reflective writing results indicate that students were not as able to transfer knowledge obtained into recognition of wider applications and changes in personal practices. More research is needed to determine how best to help introductory computing students to better integrate the contextual knowledge they acquire.

7 Acknowledgements

The authors would like to thank the Penn State Schreyer Institute for Teaching Excellence for their support of this research. The authors would also like to thank Dr. Michael Murphy, Research Associate, Penn State Schreyer Institute for Teaching Excellence, for his invaluable contributions to this project, as well as Mrs. Kathleen Morgan, Penn State Lehigh Valley, for her assistance with data collection.

References

[1] Ken Abernethy and Kevin Treu. Integrating sustainability across the computer science curriculum. *Journal of Computing Sciences in Colleges*, 30(2):220–228, 2014.

[2] Anne G Applin. A learner-centered approach to teaching ethics in computing. In *ACM SIGCSE Bulletin*, volume 38, pages 530–534. ACM, 2006.

[3] Veronica A Burrows, Barry McNeill, Norma F Hubele, and Lynn Bellamy. Statistical evidence for enhanced learning of content through reflective journal writing. *Journal of Engineering Education*, 90(4):661–667, 2001.

[4] Yu Cai. Integrating sustainability into undergraduate computing education. In *Proceedings of the 41st ACM technical symposium on Computer science education*, pages 524–528. ACM, 2010.

[5] Ali Erkan, Tom Pfaff, Jason Hamilton, and Michael Rogers. Sustainability themed problem solving in data structures and algorithms. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 9–14. ACM, 2012.

[6] Association for Computing Machinery. *ACM Code of Ethics and Professional Conduct*. 2018. `https://www.acm.org/code-of-ethics`.

[7] Mark Guzdial. Exploring hypotheses about media computation. In *Proceedings of the ninth annual international ACM conference on International computing education research*, pages 19–26. ACM, 2013.

[8] Margaret Hamilton. Learning and teaching computing sustainability. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 338–338. ACM, 2015.

[9] Mary Taylor Huber and Pat Hutchings. Integrative learning: Mapping the terrain. the academy in transition. *Association of American Colleges and Universities*, 2004.

[10] Mary Taylor Huber, Pat Hutchings, and Richard Gale. Integrative learning for liberal education. *Peer Review*, 7(3/4), 2005.

[11] Julie Thompson Klein. Integrative learning and interdisciplinary studies. *Peer Review*, 7(4):8–10, 2005.

[12] Jane L Lehr. Liberal studies in engineering programs–creating space for emergent & individualized pathways to success for women in computing disciplines. *age*, 26:1, 2015.

[13] Samuel Mann, Logan Muller, Janet Davis, Claudia Roda, and Alison Young. Computing and sustainability: evaluating resources for educators. *ACM SIGCSE Bulletin*, 41(4):144–155, 2010.

[14] ACM/IEEE-CS Joint Task Force on Computing Curricula. *Computer Science Curricula 2013*. ACM Press and IEEE Computer Society Press, 2013.

[15] Cyndi Rader, Doug Hakkarinen, Barbara M Moskal, and Keith Hellman. Exploring the appeal of socially relevant computing: are students interested in socially relevant problems? In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 423–428. ACM, 2011.

[16] Jane Ritchie, Jane Lewis, Carol McNaughton Nicholls, Rachel Ormston, et al. *Qualitative research practice: A guide for social science students and researchers*. sage, 2013.

[17] Jeffrey A Stone. Using reflective blogs for pedagogical feedback in cs1. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 259–264. ACM, 2012.

[18] Jeffrey A Stone. A sustainability theme for introductory programming courses. *International Journal of Modern Education and Computer Science*, 11(2):1, 2019.

[19] Kate Whalen and Antonio Paez. Development of a new framework to guide, assess, and evaluate student reflections in a university sustainability course. *Teaching & Learning Inquiry*, 7(1):55–77, 2019.

plain

Simple Agent Analyses for CS1 using British Square*

Courtney Brown[1], Chris Alvin[2], Lori Alvin[1], John Harris[1]
[1]Mathematics Department
[2]Computer Science Department
Furman University
Greenville, SC 29613
{courtney.brown, calvin[†], lori.alvin, john.harris}@furman.edu

Abstract

Digitization of board games has resulted in renewed interest in designing and implementing intelligent agents to play synergistically with a player or as an antagonist. In this paper, we consider the game British Square, a simple board game played on a 5×5 grid. We formally analyze the British Square in a 3×3 setting en route to proposing several agents for play on arbitrarily sized boards. Using a sequence of simulations, we compare the utility of these agents against one another as well as address questions of fairness in the game. We wish to re-introduce the community to board game analysis as a means of inspiring students to explore casual games with simple AI strategies.

1 Introduction

Over the past few years, there has been a resurgence in the playing of board games among families, friends, and in gathering spaces (combination board game bakeries, bars, etc.). This surge of interest has also been seen in the digitization of such games. Currently, there are more than 300 games tagged with the moniker "Board Game" on the digital gaming distribution site, Steam [6].

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

†Corresponding author

Many types of games exist in digital form: classic games (Risk, Monopoly, etc.), abstract strategy games, and even campaign-based dungeon crawlers (Gloomhaven). An important feature of digitized board games is the ability to play as a single player. In these cases, there needs to be a game mode in which other players are controlled by agents.

In this paper, we consider the game British Square [4], a piece placement game played on a 5×5 grid. This game is of particular interest because it was published in 1978 and does not currently possess a digital version. In Section 2, we introduce the rules of British Square and discuss questions of interest related to game outcome (win, loss, or draw). In Section 3 we prove several interesting results on a 3×3 board en route to considering the 5×5 board and the possibility of an agent player. In Section 4 we describe our simple AI strategies, and we evaluate those strategies in Section 5.

2 The Game: British Square

British Square is a two-player game that is played on a 5×5 grid (25 squares). Players take turns placing pieces onto squares (or cells) in the grid following a set of simple rules:

(i) Each square can hold at most one piece.

(ii) A player cannot place a piece in a square that shares an edge with a square containing an opponent's piece. We refer to this rule as the *adjacency rule.*

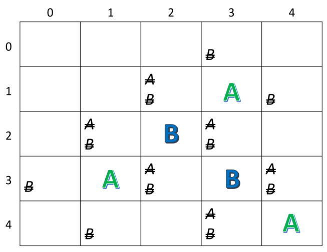


Figure 1: A partially played 5×5 British Square board.

The large As and Bs indicate squares that contain player $\mathcal{A}$'s and player $\mathcal{B}$'s pieces, respectively. The small As and Bs with strikethroughs indicate spaces that are not available to player $\mathcal{A}$ and Player $\mathcal{B}$, respectively (because of the adjacency rule). For example in the partially played board in Figure 1, player $\mathcal{B}$ cannot play in square $(0, 3)$ since it shares an edge with the $(1, 3)$ square that already contains an A.

Play continues until neither player is able to place any additional pieces, and the player with the most pieces on the board is the winner. In terms of gameplay, there is one more rule that is of consequence for our analysis:

(iii) The initial move by player $\mathcal{A}$ cannot be in the center square.

3 Formal Analysis of the 3×3 Board

In order to gain insight about a problem, it is often helpful (for researchers and students alike) to examine smaller cases first. In this section we give a formal analysis of the game when played on a 3×3 board. Many of these results are appropriate for study and analysis by students.

We will label the cells of the 3×3 grid 1 through 9 from top-left to bottom-right as shown in Figure 2. We will also refer to the first player as player $\mathcal{A}$ and the second player as player $\mathcal{B}$.

1	2	3
4	5	6
7	8	9

Figure 2: Labeling of the 3×3 board.

We will refer to their game pieces as A and B, respectively. We also define a *complete board* to be a board with assigned player pieces such that there are no more legal moves available for either player.

Lemma 3.1 *Given a complete 3×3 board, if player $\mathcal{A}$ has a piece in the center position 5, then (i) player $\mathcal{B}$ has pieces in exactly two corner cells and no other cells; (ii) player $\mathcal{A}$ occupies exactly two corner cells; (iii) player $\mathcal{A}$ wins.*

Suppose we have a complete board with an A piece in the center cell. We know due to the adjacency rule that player $\mathcal{B}$ cannot occupy any of the even numbered cells. Hence all pieces labeled B must be in the corner cells.

Assume for the moment that player $\mathcal{B}$ occupies all four corner cells in the complete board. By the adjacency rule, the only cell that could contain A is the center cell. However, because $\mathcal{A}$ plays first and cannot play in the center square on the first move, this cannot occur. Hence a complete board cannot have B pieces in all four corners.

Next assume player $\mathcal{B}$ occupies exactly three corner cells in the complete board; without loss of generality we can assume those positions are 1, 3, and 7. This means that no even cells can contain A pieces, and so the first A piece must have been placed in the remaining corner cell 9. Further, as player $\mathcal{A}$ is the first player, after player $\mathcal{B}$ has played in exactly two corner cells, one corner cell would still be open and accessible to player $\mathcal{A}$ on the third turn. Hence it is not possible for B pieces to occupy exactly three corner cells when an A piece is in the central cell. Therefore we conclude that player $\mathcal{B}$ can control at most two corner cells on such a board.

Finally suppose exactly one corner cell on the complete board contains a B piece. There is no configuration of pieces such that player $\mathcal{A}$ can block off player $\mathcal{B}$ after only one of $\mathcal{B}$'s moves. Regardless of $\mathcal{A}$'s first two moves, there will always be at least one additional corner for player $\mathcal{B}$ to choose on their second move. We conclude that player $\mathcal{B}$ must have pieces on exactly two corner cells and no other cells.

We know exactly two corner cells contain B pieces and no other cells contain B pieces. Without loss of generality, we may assume that the B pieces are either in cells 1 and 7 or they are in cells 1 and 9. Player $\mathcal{A}$ will control cells 3, 6, and 9 in the former case, while in the latter case they will control cells 3 and 7. In both cases, $\mathcal{A}$ occupies exactly two corner cells. Further, $\mathcal{A}$ wins in both cases.

We now know that player $\mathcal{A}$ will win if they control the center square. As we will see in Theorem 3.4, if player $\mathcal{A}$ knows what to do, player $\mathcal{A}$ can always win. Before we see why this is true, we make a few more interesting observations.

Lemma 3.2 *On the* 3×3 *board, (i) player* $\mathcal{A}$ *can win even if player* $\mathcal{B}$ *controls the center cell; (ii) the center cell does not need to be occupied for there to be a winner; (iii) it is possible for player* $\mathcal{B}$ *to win; (iv) it is possible for the game to end in a tie.*

The completed boards in Figure 3 through Figure 6 show that each of these statements is true. The subscripts in each case indicate on which turn the piece was played. For instance, A_2 means that this was the piece played on player $\mathcal{A}$'s second turn.

B_2		A_2
	B_1	
A_1		A_3

Figure 3: Player $\mathcal{A}$ wins even though B is in the center.

	A_1	A_2
B_2		A_3
B_1		A_4

Figure 4: There is a winner ($\mathcal{A}$) even though the center is unoccupied.

B_2		A_2
B_4	B_1	
B_3		A_1

Figure 5: Player $\mathcal{B}$ wins this game.

B_1		A_1
B_2		A_2
B_3		A_3

Figure 6: The game ends in a tie.

We now prove a critical result about ownership of a middle space in a 3×3 game.

Lemma 3.3 *If the center cell is occupied on a* 3×3 *game board, then the game cannot end in a tie.*

Suppose we have a completed game board. We know by Lemma 3.1 that if player $\mathcal{A}$'s piece is in the center, then Player $\mathcal{A}$ wins (not a tie). Therefore, we consider the case where player $\mathcal{B}$ controls the center cell. This means that there are no player $\mathcal{A}$ pieces in any of the even numbered cells. That is, A's can only appear in the corners. Consider now the possibilities for what could have been player $\mathcal{A}$'s first two moves.

If player $\mathcal{A}$'s first two moves were in opposite corners (say 1 and 9), then the completed board could not have B pieces on any of cells 1, 2, 4, 6, 8, or 9. That is, the only places for B are 3, 5, and 7. Since it would be impossible for player $\mathcal{B}$ to claim all of 3, 5, and 7 before player $\mathcal{A}$ could make a third move, and since we are assuming that there is a B in the center square, it must be that player $\mathcal{B}$ owns the center square and exactly one of 3 or 7. This leaves the other of 3 or 7 to be available for player $\mathcal{A}$, which gives $\mathcal{A}$ three squares while $\mathcal{B}$ has only two. Player $\mathcal{A}$ wins in this case.

Suppose now that player $\mathcal{A}$'s first two moves were in adjacent corners (say 1 and 7). If cell 6 is occupied (by $\mathcal{B}$ since $\mathcal{A}$ cannot be there), then cells 3 and 9 must also be occupied by player $\mathcal{B}$; therefore $\mathcal{B}$ wins. If cell 6 is not occupied, then player $\mathcal{A}$ occupies one of 3 or 9, and player $\mathcal{B}$ occupies the other — meaning that $\mathcal{A}$ wins. In all cases there is a winner, and so a tie is not possible if the center cell is occupied.

Theorem 3.4 *Player $\mathcal{A}$ always has a winning strategy on a 3×3 board.*

We show that for any state of the board at any point in the game, player $\mathcal{A}$ always has a choice that will lead to an eventual win. To start the game, $\mathcal{A}$ should choose to claim a corner, say cell 1. The only cells player $\mathcal{B}$ can choose are cells 3, 5, 6, 7, 8, and 9. Due to symmetry, we only need to consider cases where $\mathcal{B}$ chooses 5, 7, 8, or 9.

Case 1. If player $\mathcal{B}$ initially chooses cell 5, then player $\mathcal{A}$ should choose to play on cell 9. This will leave exactly two valid cells for player $\mathcal{B}$: 3 and 7. No matter which choice player $\mathcal{B}$ makes, player $\mathcal{A}$ will choose the other corner cell and win as shown in Figure 7.

A_1		$\mathbf{B_2}$
	B_1	
A_3		A_2

Figure 7: Case 1 final board game state.

A_1	A_3	A_4
		A_2
B_1	$\mathbf{B_2}$	

Figure 8: Case 2 final board game state.

Case 2. If player $\mathcal{B}$ initially chooses cell 7, then player $\mathcal{A}$ should choose to play the second move on cell 6. The only remaining move for player $\mathcal{B}$ will be to select cell 8. Player $\mathcal{A}$ will eventually win (see Figure 8 for a complete board).

Case 3. If player $\mathcal{B}$ initially chooses to place a piece on cell 8, then player $\mathcal{A}$ should choose to play on cell 6. The only remaining move for player $\mathcal{B}$ will be to select cell 7. The complete board in this situation is similar to the complete board in Figure 8, but with pieces B_1 and B_2 switched; once again player $\mathcal{A}$ will win.

Case 4. If player $\mathcal{B}$ initially chooses cell 9, then player $\mathcal{A}$ should choose the center cell. By Lemma 3.1, player $\mathcal{A}$ will win.

4 Intelligent Strategies

In order to create agent-player functionality, it is important to consider what strategies students consider while playing the game. In this section we describe some possible strategies.

In each of the strategies we describe, the agent makes a choice depending on the current state of the board. That is, the agent does not consider past or future board states when making a decision. It is also true in each strategy that the agent considers all possible allowable moves at a given point. If a strategy identifies multiple possible best moves, a uniform random choice will be made. Each of our intelligent strategies view each open space on the board as the center of a 3×3 board consistent with Theorem 3.4. Our heuristic for winning a game on a board larger than 3×3 is for a player to win as many local 3×3 sub-boards as possible. We will use Figure 1 as an example when describing these strategies, and we will assume that it is player $\mathcal{B}$'s turn to place a piece. We use the term *adjacent* to refer to squares that share an edge.

The *Best Open* strategy selects an open space with the greatest number of open adjacent spaces around it, independent of whether either player can play on the surrounding squares. For instance, in Figure 1 the best open strategy would assess space $(0, 2)$ as having 3 open squares surrounding it, and it would say that space $(0, 4)$ would have two open spaces surrounding it. In this case, the best open strategy would select space $(1, 1)$ since it is the only space with 4 open spaces surrounding it. This is a weak, greedy strategy since it does not account for player interaction in its choices; however, it does prioritize non-border spaces.

The *Disruptor* strategy chooses a space that will disallow the maximum number of spaces for an opposing player (thus being disruptive). For example, for $\mathcal{B}$ in Figure 1, space $(4, 2)$ is disruptive to $\mathcal{A}$ because by playing there, $\mathcal{B}$ removes the possibility that $\mathcal{A}$ could play in the two spaces $(4, 1)$ and $(4, 2)$. We will describe this situation by saying that the space $(4, 2)$ is disruptive to $\mathcal{A}$ with rank 2. In Figure 1, both $(0, 1)$ and $(1, 0)$ are disruptive to $\mathcal{A}$ with rank 4, and so the disruptor strategy would choose randomly between these two spaces. Early in a game, the disruptor strategy tends to choose spaces in open areas on the board and may continue to do so by playing diagonally to itself (creating a self-checkerboard pattern). As the game progresses, this strategy moves toward ownership of spaces diagonal to the opponent creating a checkerboard-style parity.

Our last strategy uses the same approach as the disruptor, but prioritizes spaces that are diagonal to an opponent; we refer to it as a *Diagonal Disruptor*. For $\mathcal{B}$ in Figure 1, if given the choice between $(2, 0)$ and $(1, 1)$ each with rank 3, $\mathcal{B}$ would choose $(2, 0)$ since it is diagonal to one space with $\mathcal{A}$. This strategy results in close play and long-term checkerboard parity.

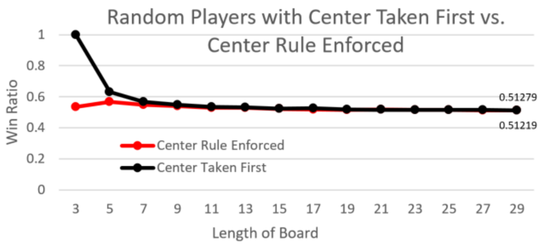


Figure 9: Player $\mathcal{A}$ win-ratios of center space taken first versus not.

5 Simulations

In this section we discuss several questions about the game of British Square and provide some empirical evidence. In much of what follows, we examine the questions for larger (and differently sized) boards as well. In all reported simulations, we executed a matrix-based Java solution 100000 times to minimize variance.

Question 1. *Does the center position give the first player "too much" of an advantage if permitted to play there on the opening move?*

To address this question we ran two simulations reported in Figure 9 with increasing odd length boards (even boards do not have a center space). In each simulation, both pairs of players chose their moves randomly; however, in the second simulation, player $\mathcal{A}$ will always take the center square on the first move. As shown in Figure 9, we confirm that the first player will always win the game on a 3×3 board and observe that as the size of the board increases, the center space becomes inconsequential.

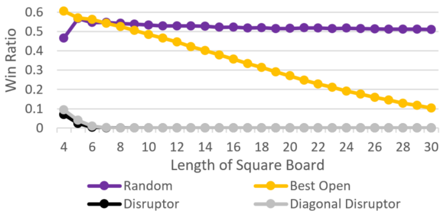


Figure 10: Random player $\mathcal{A}$ win ratios.

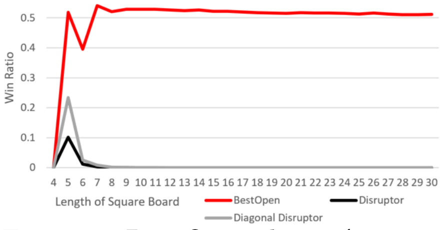


Figure 11: Best Open player $\mathcal{A}$ win ratios.

Question 2. *Which agent performed the best?*

Figure 10 depicts the win ratios of a first player making random choices and the second player using an agent method. For two random players, Figure 10 affirms the results of Figure 9 in that two random players will tend toward

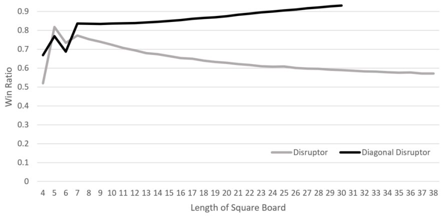


Figure 12: Win ratios of Disruptor first player.

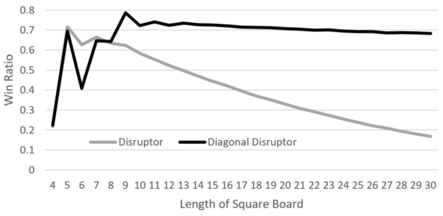


Figure 13: Win ratios of Diagonal Disruptor first player.

winning half the games as the board size increases. We also conclude from Figure 10 that the Disruptor agents are superior to the Best Open agent. In head-to-head competition, Best Open is weak compared to disruption as shown in Figure 11 and limits to 50% of wins against a self-Best Open opponent. What is interesting when comparing agents is that the Disruptor finds general success against all other agent types including against a Diagonal Disruptor (Figure 13) as well as a self-Disruptor match (Figure 12).

Generally, we conclude British Square is most interesting as a family game with a 5×5 board compared to other sizes since any of our strategies, including a random strategy, has an opportunity to win.

Table 1: Sample non-square board results (100000 iterations): Random vs. Random.

	3×4	4×5	5×6	4×6	5×8	2×4	3×9	1×10	27×39
Win Ratio	0.565	0.558	0.566	0.525	0.553	0.173	0.561	0.608	0.512
Tie Ratio	0.266	0.235	0.178	0.267	0.165	0.804	0.191	0.120	0.033
Avg. $\mathcal{A}$ Win Differential	1.696	2.090	2.366	2.313	2.6375	2.635	2.313	1.698	10.304
Avg. $\mathcal{B}$ Win Differential	1.223	1.618	1.913	1.874	2.159	2.0	1.873	1.308	9.690

Question 3. *Does the first player have an advantage?*

We first consider non-square boards. From Question 1, with increasing sized square boards, we know that the center position becomes less significant. However, with smaller boards, there is much variability. Therefore, we consider smaller, non-square boards since this type of board lacks a center position. Using random play for both agents, Table 1 indicates that the first player ($\mathcal{A}$) generally has the advantage over the second player ($\mathcal{B}$) when considering win ratio.

Question 4. *How many more squares does the winner control compared to the loser?*

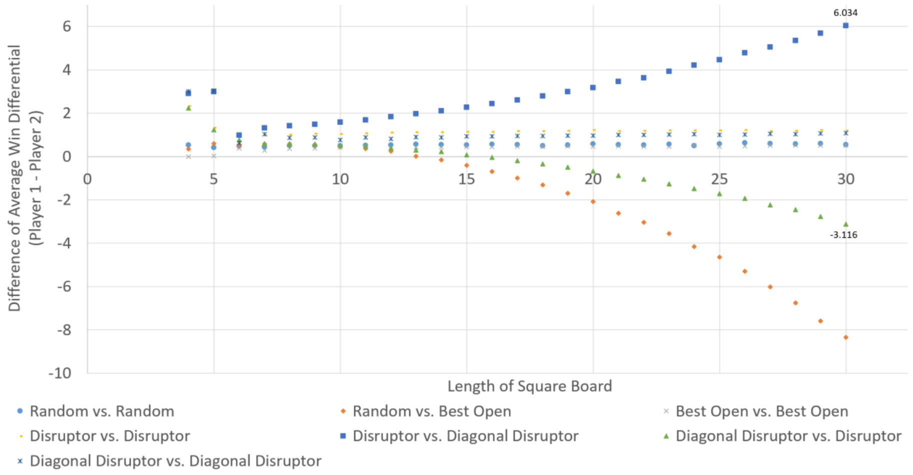


Figure 14: Differences between average maximum win differentials.

To examine this question, we introduce the average win differential: the difference between the number of spaces owned by the winner compared to the loser. In a fair game, we expect the differential between $\mathcal{A}$ and $\mathcal{B}$ to be roughly equivalent. We will see that when $\mathcal{A}$ wins, the average win differential is statistically significantly in favor of $\mathcal{A}$.

We also consider average win differential on square boards. Figure 14 shows a select subset of agent competitions within a meaningful range. Those games not depicted in Figure 14 such as a Random first player and any other agent resulted in significant long-term wins for the second player. For example, the average win differential between Random and Disruptor on a 30×30 board is 144.352, a considerable amount considering 900 available spaces in total. A more reasonable match between Random and Best Open in Figure 14 shows a gradual trend toward more significant wins for the Best Open player.

We make several observations about Figure 14. For more fair matches such as Best Open vs. Best Open and Disruptor vs. Disruptor, we see positive differences and thus conclude that the first player maintains an advantage even as the board size increases. The particular competitions with Disruptor and Diagonal Disruptor are of interest. We conclude that the Disruptor agent is superior with increasing board sizes noting the upward trend when Disruptor is first against the Diagonal Disruptor and the downward trend when the order is switched. However, we again see evidence of first player advantage because the average win differential is more significant when the Disruptor is the first player compared to being second (e.g., a difference of 6.034 vs. -3.116 on a

30×30 board) and the corresponding, respective monotonically increasing and decreasing sequences.

6 Related Works

Bezáková, Heliotis, and Strout [2, 3, 5] motivate the use of board games in the CS1 and CS2 classrooms to provide context for introductory computing concepts. The authors discuss several board-based games in the classroom and propose four criteria to selecting a game [2], including simple rules, non-violent, and gender neutral. We agree with the stated criteria, but argue that a "well-ranked game" (popular game) is not necessary. In the case of British Square, we have a game with simple rules that provides a fruitful environment for exploration and assessment with simulations as well as mathematical analysis. While games provide great motivation for CS1 and CS2, we argue that rigor and formalism should be instilled in students who plan to study computer science beyond CS2. That is, proof can motivate algorithm development and analysis (and vice versa). In fact, these same games can simultaneously motivate formal analysis by considering, for example, the size of a search space.

In [1], Alvin proposed an activity for CS1 students to analyze the game *Guess Who?* and develop an optimal decision procedure. *Guess Who?* is a game for children that is easily modeled whereas British Square is intended for an older audience with a larger search space. *Guess Who?* is not a traditional board-based game compared to British Square, but the author's approach to developing a winning strategy is related to our work. Our work does not attempt to elicit a machine learning-based algorithm, but instead focuses on strategy development through search and static board analysis.

7 Discussion and Conclusions

British Square is a game in which a player identifies which board spaces are possible, evaluates the fitness of taking a space, and potentially performs a search considering future piece placements. Thus, British Square naturally motivates concepts such as data structures, search algorithms, and eliciting simple fitness functions. In particular, a CS1 student with limited programming experience can implement a two-dimensional grid, a simple agent, and an interactive text-based interface. Such a domain allows for students to continue to explore and improve game strategies thus providing a fertile ground to explore for beginners and more advanced programmers. For example, a student can easily vary the playfield to be smaller, larger, non-square, non-rectangular, and more. A student might also implement more complex agents taking into account the entire board state or if a student is bold, implement a minimax algorithm for deep play and analysis.

On a 3×3 board we have rigorously shown that the first player may always win. Using our formal analyses of the smaller board, we proposed agent techniques for bot-play of the game on any size board that might be easily implemented and explored in a CS1 or CS2 classroom. We then performed a sequence of simulations in which agents would compete against one another. These simulations indicate that a 5×5 board is ideal family fun (using any strategy), but also that the Disruptor approach seems to be the most effective of the proposed strategies.

References

[1] Chris Alvin. Student generation of an optimal decision procedure using guess who? In *The Journal of Computing Sciences in Colleges*, volume 34(5), pages 26–34, 2019.

[2] Ivona Bezáková, James E. Heliotis, and Sean Strout. Board game strategies in introductory computer science. In *SIGCSE*, pages 17–22, 2013.

[3] Ivona Bezáková, James E. Heliotis, and Sean Strout. On the efficacy of board game strategy development as a first-year CS project. In *SIGCSE*, pages 283–288, 2014.

[4] Gabriel Games. British square, 1978.

[5] James E. Heliotis, Ivona Bezáková, and Sean Strout. Programming board game strategies in CS2. In *IEEE Frontiers in Education Conference*, pages 4–5, 2013.

[6] Valve Corporation. http://store.steampowered.com/ Accessed Nov-18-2019.

Experiential Learning Framework for Smaller Computer Science Programs[*]

Zachary Kissel[1], Christopher Stuetzle[1]
[1]Department of Computer Science
Merrimack College
North Andover, MA 01845
{kisselz, stuetzlec}@merrimack.edu

Abstract

Experiential learning (EL) permeates the Computer Science discipline. This work seeks to codify EL practices for computer science pedagogy into five key pillars. These pillars have been successfully applied at a small to mid-sized college within the heavily competitive Boston area. This paper further describes how a computer science department may effectively implement the pillars in their own curriculum.

1 Experiential Learning in Traditional Pedagogy

The benefits of EL practices in higher education are well established [4, 9, 10]. Research has shown that students learn most when they are more engaged in the experience rather than as passive participants [14]. The Association for Experiential Education (AEE), founded in 1977, regards experiential education as "a philosophy that informs many methodologies in which educators purposefully engage with learners in direct experience and focused reflection in order to increase knowledge, develop skills, clarify values, and develop people's capacity to contribute to their communities." [1] Traditionally, this has manifested in student internships and the use of case studies. While computer science students have traditionally been encouraged to seek internships and co-ops, and have been overall successful in attaining them, there are skill competencies

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

that these experiences are not guaranteed to provide, such as communication, team/self management, or service learning.

Computer science, by its nature, is rife with experiential opportunities inside the classroom. First and foremost, the hands-on aspects of courses in the discipline (project-based learning, in-course labs, internships-for-credit, and practical quizzes and tests) are common in university curricula and important for the overall growth of the students. These have also been universally-adopted by the community as a whole. We take these EL components as given for a computer science curriculum. However, from years of discussing program requirements with CIOs and hiring managers from several industries seeking recent graduates in computer science, as well as environmental scans of the local industry, this is no longer seen as sufficient by employers and graduate schools. Students need more well rounded experiential opportunities during their time as undergraduates.

This has been supported by the literature. Previous work has argued the importance of developing soft skills in computer science, through the use of service learning [16], modification of students' theories on self [2], and a form of gamification [17]. Previous work, except [2], placed soft skill development late in the curriculum, normally in a capstone experience. We assert this delay in soft skill development reduces the potential impact on student development.

There are several ways that schools introduce additional EL opportunities into their curricula. A near ubiquitous EL technique is the use of "hackathons," which have been adopted into formal instructional processes by authors such as Gama et al. [5]. Hackathons can increase student engagement and enhance team based learning outcomes. Hackathons are sometimes incorporated into curricula as either a prep course, or as part of a larger programming course. Along similar extracurricular lines are external experiences such as the NASA Robotic Mining Competition [7]. The authors of this work believe this is not sufficient for well-rounded graduates of computer science undergraduate programs.

1.1 Background of Institution and Department

Our institution is a Catholic college, in the Augustinian tradition, located outside of Boston, Massachusetts; an area of heavy competition in higher education. It currently enrolls 3500 undergraduate students and 700 graduate students. The computer science department graduates between 15 and 20 majors each year, and has roughly doubled in size since 2012.

Prior to 2012 the computer science department's use of EL was drastically different. While there was a capstone experience and some project oriented courses, there was a dearth of independent studies, public and in-class presentations, or formalized soft-skill development. Moreover, the curriculum lacked cohesion around the use of EL practices. Starting in 2012, the faculty initiated an organized effort to introduce consistent use of EL throughout the curriculum. The resulting curricular decisions are captured as a five-pillar framework.

1.2 Contribution

EL curricular frameworks have been presented in several disciplines, such as Marketing [13], Management [11], Medicine [12], and Foreign Language [8], for example. Additionally, authors have provided frameworks for particular aspects of EL curricula, such as scaffolded reflections [3] or working in teams [6]. To the best of this work's authors' knowledge, no framework for experiential learning curricular guidelines for small- and mid-sized computer science undergraduate programs has been explored in the literature.

This paper presents a formalized framework for EL in small- to mid-sized undergraduate computer science curricula that has been effectively deployed in the authors' computer science department. The framework is broken into five pillars which are emphasized throughout the curriculum. Although we recognize the fact that many of these curricular aspects are present and even ubiquitous throughout many institutions' undergraduate computer science curricula (specifically an emphasis on internship opportunities and a project-based curriculum), our work attempts to codify these notions into a sustainable framework that can be applied to institutions similar in size to our own.

2 Experiential Learning Framework

The following five pillars of our framework describe a formalized pedagogical approach to incorporating EL into undergraduate students' curricular and co-curricular experiences. In the following paragraphs, the incorporation and focus of each of these pillars will be discussed. Each discussion is followed by a description of how the pillar is implemented at our institution, though these are not the only ways in which they can be incorporated into the curriculum or department student outcomes. It's important to note that, while each pillar can be discussed separately, they all inform each other and as such create a web of EL opportunities.

- **Soft Skills -** Focus on soft skills through reinforcement and repetition
- **Real-World Focus -** Consistent and constant real-world curricular tie-ins
- **Group Work -** Group work with an emphasis on tools such as source control, test-driven development, and real-time collaborative development

- **Student Empowerment -** Student empowerment through choice and ownership of learning
- **Dissemination of Individual Work -** All novel or substantial work is disseminated to the broader community

2.1 Soft Skills Pillar

To incorporate soft skills into the curriculum our experiences have demonstrated that emphasis, at all levels, should be placed on both written and oral communication and self promotion. In both forms of communication, strong technical and non-technical communication should be fostered; paying special attention to communicating with audiences of various technical backgrounds. Oral communication should be honed through frequent projects incorporating presentations whose length is proportional to the scope of project. For theoretical courses and capstone experiences written skills should be fostered, in addition to oral skills. The key factor for building good communication skills is repetition. For this reason it is recommended that a concerted effort be made to include communications skills in a majority of the curriculum. Self promotion is most amenable to co-curricular activities, generally through the form of interview and internship preparation. We also recommend involving any career center on campus to participate in a portion of these co-curricular activities.

At our institution soft skills are present in almost every class and co-curricular opportunity. In our curriculum oral communication skills are practiced through course final projects in which presentations are required. Without exception, students are provided with a rubric to help focus the presentation and determine the instructors' weight on certain components (generally including clarity and professionalism). In independent studies and the capstone experience, communicating with varying audiences has been practiced through college wide poster sessions. The capstone experience additionally provides a communication opportunity that simulates direct interaction with a client, as well as a heavy seminar and discussion component where students can engage with each other and their faculty in debating current important topics, such as net neutrality or algorithmic transparency. Soft skills are incorporated into other coursework as well, such as public debates, the value of which has been previously demonstrated [15].

To ingrain in students the idea of self promotion the department runs seminars, in cooperation with the career center, in the last course of the introductory sequence. These seminars include writing a good resume, discussions of elevator pitches, and how to perform well in a code interview. These are often re-emphasized during informal meetings, such as office hours with either the faculty of the computer science department or advisers from the career center.

This pillar maps well to our student outcome goal of graduating students with *"an ability to communicate effectively with a range of audiences"*.

2.2 Real-World Focus Pillar

An emphasis should be placed on how skills and ideas learned through curricular and co-curricular means are necessary and valued by the industry and graduate school community. This emphasis takes many forms. Instructors should make an attempt to map in-class project topics to real-world applications. Where applicable, course content should encourage students to engage with important topics of the day and think about these topics in ways beyond a traditional computer science curriculum, such as from ethical or service perspectives. Faculty should not only allow this exploration, but encourage it through course content and informal discussion. Faculty should require a degree of professionalism from students when communicating through e-mail or in person.

Departments should encourage students to seek internships as soon as practically possible. Not only are internships arguably the most traditional of EL avenues, but they are an invaluable resource for students to experience first-hand application of the skills and ideas they acquire through course work. Departments should also be flexible in allowing students to engage with co-ops and study abroad opportunities, though this is a challenge as programs continue to grow. Additionally, departments should track internships taken on by their students, and encourage thoughtful formal reflection on their experiences.

Faculty should encourage and teach aspects of entrepreneurship. Students should understand intellectual property, knowing who owns the work they produce at school and in their internships. The environment they work in should foster creativity and encourage initiative, so that students believe what they create is valued and may be successful.

In our department we work closely with our institution's career services center in a variety of ways. First, all internships can be registered through the center and reflection papers are required for the institution to formally acknowledge the internship as part of the student's experience while in school. The department encourages students to register all internship and co-op experiences with the career center. Secondly, the career center regularly attends our classes and hosts workshops on resume building, interview tactics, and other career preparation topics.

Courses in the major present project opportunities that map directly to real-world applications. In some courses, such as Artificial Intelligence or Computer Graphics, this is straightforward. However, even in courses where the mapping might not be as obvious, such as Operating Systems, students write shells and implement and utilize thread pools to construct web servers and content filtering proxies, all tied back to the algorithms and concepts taught in the course.

This pillar maps to our student outcome goal of graduating students with *"an understanding of professional, ethical, legal, security and social issues and responsibilities"*.

2.3 Group Work Pillar

Whether in an industry setting or a graduate school setting, students will eventually spend significant portions of their lives working in groups toward a common goal. This clearly presents a large set of challenges. An EL-rich education should afford students as many opportunities as possible to work in groups. They will not always have a positive reaction to these opportunities, but they are important experiences. However, the curriculum should also strive to provide students with the tools to thrive in group settings, by providing the tools to enhance students' workflows and by providing the necessary emotional ground work for the challenges they may face when in group settings.

Faculty should encourage the use of group workflow tools for their group projects. This includes version control software (such as git), collaborative development tools (such as Google Docs), and project management tools (such as BitBucket's issue tracker). Embedding these ideas throughout the curriculum allows students to overcome their learning curve and get used to them, but also provides additional directly-applicable skills that can be added to resumes.

Arguably the most prolific complaint from students regarding group work is the unfair workload that emerges from working as a team. Faculty should put countermeasures in place to combat this workload imbalance. Allowing students to weight each members' contribution is a classic example of this, but there are others. Incorporating an issue tracker and weighting the projects based on issues closed is a more transparent method.

At our institution, group work is a focus of many upper level classes, and all include an emphasis on these team management tools. Groups present, produce deliverables, and disseminate together. Our capstone experience especially provides a heavy emphasis on group work, and includes units on group dynamics and inter-group communication. Groups are required to provide both individual and group-written deliverables and group contribution is weighted based on the number, weight, and priority of issues closed and bugs squashed.

This pillar maps directly to our department student outcome goal of graduating students with *"an ability to function effectively on teams to accomplish a common goal"*.

2.4 Student Empowerment Pillar

We, like others, have observed that students who feel empowered with choices in a class often achieve a deeper learning experience. It is recommended that, when possible, students should be provided choice in projects. Whether that choice is through a list of potential topics, a free form choice approved on a case-by-case basis, or a combination of the two.

It is important to foster a culture of independent studies within the department. Independent studies empower students to take control of their learning by adapting the curriculum to their aspirations. In order to build a successful culture of independent studies, the faculty must engage in practices that highlight the value of independent studies to the students. A substantial percentage (36.5%) of our student population engages in independent studies and thus mold the curriculum towards their interests. In our view, a culture of independent studies has formed over the past seven years, the time period where this framework was in place. This growth was energized by the faculty through announcement of potential topics in courses, where appropriate, as well as through curation of a student accessible list of potential topics of faculty interest. This culture has created a sense of a community of learners.

At our institution thirteen of the eighteen courses offered to computer science majors beyond their first year have presentations based on a project of the student's choosing. Some of these classes use group work while others utilize individual projects. Independent studies consist solely of student chosen work, in consultation with a supervising faculty member.

Finally, a component of course reflection by students is empowering. This may be achieved through the use of a post-course survey asking key questions such as: what they feel they learned, what changes to the course they would like to see implemented, what topics they feel they benefited from the most, etc. These questions may be used by the faculty to potentially modify courses, thus empowering students to enact changes in their learning environment. Thus the curriculum moves from prescriptive to adaptive. The faculty at our institution have implemented student course reflection using the questions described above, as well as additional questions. The feedback has been used to modify curriculum, which has not gone unnoticed by the students. The most change is normally enacted in the upper level electives as they are most malleable.

This pillar maps to our department student outcome goal of graduating students with *"recognition of the need for and an ability to engage in continuing professional development"*.

2.5 Dissemination of Individual Work Pillar

There are two ways in which dissemination should be formalized in the curriculum. First, all research projects, independent studies, and applicable final projects should be presented to the broader community. This can be accomplished through poster sessions, presentation symposia, or in-class project presentations. Second, faculty should emphasize and provide opportunities for students to present work at local and regional conferences. Even if student work is not accepted, the process and abstract writing that usually accompanies application procedures to conferences is valuable as it forces students to distill, summarize, and visualize their work for audiences with variable technical backgrounds.

Work dissemination ties together several of the other pillars while providing additional benefits for students. There are clear benefits with regard to soft skill development, as it forces students to communicate, through both written and oral means, with audiences of varying technical skills and backgrounds. Students who are given a choice as to what and where to disseminate are empowered to own their work, and this can often lead to requests to continue work on a project past the end of its allotted time. Additionally, work dissemination gives students a sense of belonging to a larger society of academics, both at their institution and beyond, boosting morale and improving retention.

At our institution, the computer science department hosts a symposium at the end of each semester. All students who take part in independent studies during that semester are required to present their work in formal 15-minute presentations to the department faculty and students, regardless of whether their semester included original research or not. Additionally, our institution hosts a campus-wide poster session each spring semester, and students present their work to the broader college community. Beginning with their sophomore fall semester, thirteen out of eighteen courses offered to computer science majors require a significant oral presentation component, and department faculty are invited to view these presentations. Students are regularly encouraged to present their work at local conferences by their faculty, and the department funds these presentations.

3 Conclusion

Informal reactions to several portions of this framework have been overwhelmingly positive. Student survey answers have praised aspects of the curriculum that focus on soft skills, such as: public debates, in-class seminars, and end-of-semester presentations. Student learning has tracked with student reactions. Faculty have observed both greater engagement and deeper understanding of course material. Independent studies have led directly to graduate school and career path choices, as well as an active culture of student work dissemination. Many of our students have received offers for full time post-graduation employment as a result of their internships.

Faculty and staff workload considerations are important as well. It has been our experience that the initial investment of time and energy is steep as course curricula are adjusted and faculty work to include students in multiple aspects of their reseach. However as with any cultural shift, this investment levels over time.

Almost all institutions face limited resources and must work to prepare students as best as they can for employment with vertical mobility in spite of those limitations. The presented framework has allowed our program to flourish within these limitations. In particular, evidence suggests that the framework, described above, has resulted in an increase in opportunities for graduates as well as greater satisfaction with their undergraduate experience.

The institution is working to establish a database of alumni positions and opportunities. Our immediate future work is to assess each pillar individually using feedback from the alumni and their employers and adjust our curriculum accordingly.

References

[1] What is experiential education. https://www.aee.org/what-is-ee. Accessed: 2019-07-25.

[2] Mikko-Ville Apiola and Mikko-Jussi Laakso. The impact of self-theories to academic achievement and soft skills in undergraduate cs studies: First findings. pages 16–22, 07 2019.

[3] Debra Coulson and Marina Harvey. Scaffolding student reflection for experience-based learning: a framework. *Teaching in Higher Education*, 18(4):401–413, 2013.

[4] J. Dewey. *Experience And Education.* Free Press, 1938.

[5] Kiev Gama, Breno Alencar Gonçalves, and Pedro Alessio. Hackathons in the formal learning process. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, pages 248–253, New York, NY, USA, 2018. ACM.

[6] Brenda S. Gardner and Sharon J. Korth. A framework for learning to work in teams. *Journal of Education for Business*, 74(1):28–33, 1998.

[7] A. Heiney. Nasa lunabotics engineering competition. www.nasa.gov/offices/education/centers/kennedy/technology/nasarmc.html. Accessed: 2019-07-28.

[8] Viljo Kohonen. *Experiential Learning in Foreign Language Education.* Routledge, 2000.

[9] David Kolb. *Experiential Learning: Experience As The Source Of Learning And Development*, volume 1. 01 1984.

[10] David Kolb. *Experiential Learning: Experience As The Source Of Learning And Development*, volume 1. Pearson FT Press, 01 2014.

[11] Makoto Matsuo. A framework for facilitating experiential learning. *Human Resource Development Review*, 14(4):442–461, 2015.

[12] Greg Ogrinc, Linda Headrick, Sunita Mutha, Mary Coleman, Joseph O'Donnell, and Paul Miles. A framework for teaching medical students and residents about practice-based learning and improvement, synthesized from a literature review. *Academic medicine : journal of the Association of American Medical Colleges*, 78:748–56, 08 2003.

[13] Ed Petkus. A theoretical and practical framework for service-learning in marketing: Kolb's experiential learning cycle. *Journal of Marketing Education - J Market Educ*, 22:64–70, 04 2000.

[14] Patricia Sendall, Kristin Stowe, Lisa Schwartz, and Jane Parent. High-Impact Practices: An Analysis Of Select University And Business School Programs. *Business Education and Accreditation*, 8(2):13–27, 2016.

[15] Christopher Stuetzle. Public debate format for the development of soft skill competency in computer science curricula. *J. Comput. Sci. Coll.*, 30(6):32–37, June 2015.

[16] Joo Tan and John Phillips. Incorporating service learning into computer science courses. *J. Comput. Sci. Coll.*, 20(4):57–62, April 2005.

[17] Bojan Tomić, Jelena Jovanovic, Nikola Milikic, Vladan Devedzic, Sonja Dimitrijevic, Dragan Đurić, and Zoran Sevarac. Grading students' programming and soft skills with open badges: A case study. *British Journal of Educational Technology*, 06 2017.

Workforce and Career Readiness for Computing and Technology Students*

Jean Chu[1], *Patricia Morreale*[1], *Michael Press*[2]
[1]*School of Computer Science and Technology*
Kean University, Union, NJ
{jchu@kean.edu, pmorreal}@kean.edu
[2]*Manager, Software Sales Engineering*
Tech Data Corporation, Plano, TX
mbpress@pressmail.org

Abstract

Colleges and universities provide students with both knowledge and skills to be successful in life. In parallel with academic achievement in higher education is the development of career skills for the challenges of professional life that are further developed through internships or co-op jobs. The role of the curriculum in this development is pivotal for the success of college students, especially first-generation college students with a non-professional family background. Career awareness and professional preparation should be an integral aspect of the university curriculum, as it helps prepare students for employment and guides them to develop the necessary career skills before they enter the workforce. Adoption of career development into a curriculum as one credit course plays an important role. The career development course curriculum presented here focuses on career education and the key components that play a crucial role in getting a job. Students must be aware of the challenges and understand the skill needed for post-graduate success in industry. Career education should be introduced as early as sophomore year with an emphasis on internship or co- operative experiences that allow students to utilize knowledge gained in the classroom. The career education course outlined here is part of a four-year program for both computer science and information technology majors at the university level. The outline of the course content is provided along with assessment from students who successfully completed the curriculum.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

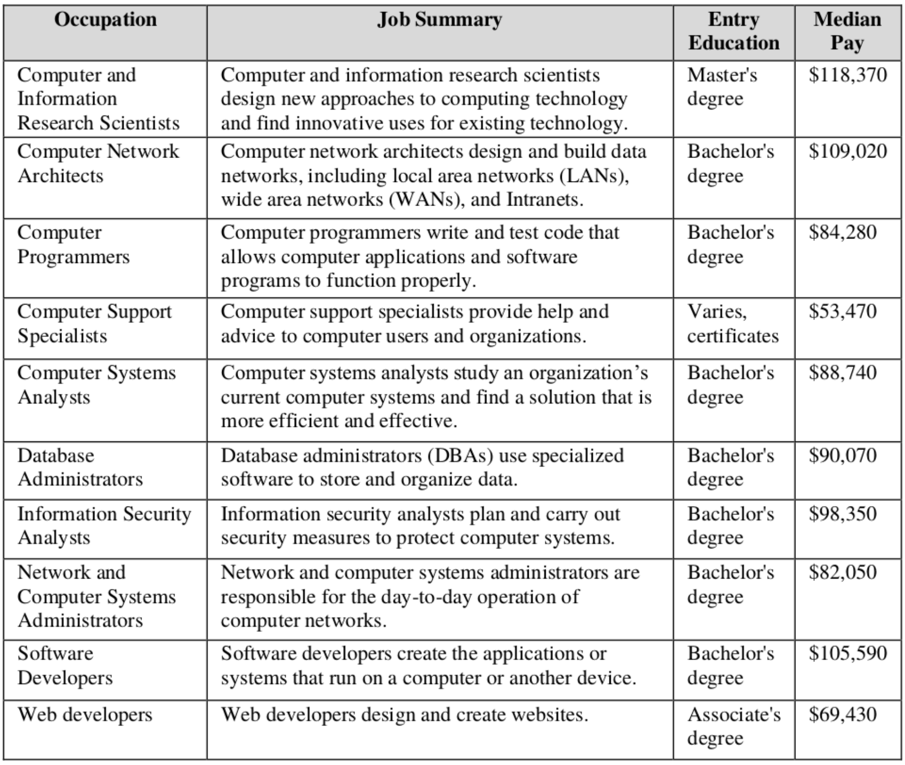
1 Introduction and Motivation

The practical application of learning is paramount for a student to ensure his/her success and attainment of his/her career goals. A Gallup survey [11] found that students

who reported having an internship or job that allowed them to apply what they were learning in the classroom during college were two times more likely to be engaged at work, but only 29% of students had that experience. Of the six high-impact experiences identified as contributing to work engagement, 25% of graduates participated in zero, and only 3% participated in all six [2]. Employment in computer science and information technology occupations is projected to grow 13 percent from 2016 to 2026, faster than the average for all occupations. These occupations are projected to add about 557,100 new jobs. Demand for these workers stems from greater emphasis on cloud computing, the collection and storage of big data, and information security. The median annual wage for computer and information technology occupations was $86,320 in May 2018, which was higher. than the median annual wage for all occupations of $38,640 [10]. Table 1 provides some details.

Low-income and first-generation students are less likely to be engaged in the academic and social experiences that foster success in college, such as studying in groups, interacting with faculty and other students, participating in extracurricular activities, and using support services [5]. Non-residential students (commuters) and students with jobs are similarly unlikely to participate in campus and pre-professional activities. Low-income and first-generation students often face barriers and challenges in understanding of their career options. With some experience, students can explore or narrow their interests from industry mentoring, coaching and job shadowing with an informed sense of their career direction when making decisions about what to study. Academic advising is complimentary to this process from the start when it includes purposeful career planning - start small think big.

Students at the university level generally have a goal or an idea about their future career choice. The university and academic departments should build on a student's career interests early and strive to forge a connection between the curriculum and the student's future career choices. Faculty can help by providing students with an opportunity to link their classroom learning with its workplace application. This will lead to positive learning outcomes such as motivation, grit, and career goal setting. "Guidance and information focused on careers should be included throughout one's undergraduate experience" [12]. Inviting guest speakers from various business sectors and industries can also prove beneficial in this regard. Integrating career-focused curricula into class discussions and projects can help students make a connection between the curriculum and practical education, an important fact to be noted. Students



Occupation	Job Summary	Entry Education	Median Pay
Computer and Information Research Scientists	Computer and information research scientists design new approaches to computing technology and find innovative uses for existing technology.	Master's degree	$118,370
Computer Network Architects	Computer network architects design and build data networks, including local area networks (LANs), wide area networks (WANs), and Intranets.	Bachelor's degree	$109,020
Computer Programmers	Computer programmers write and test code that allows computer applications and software programs to function properly.	Bachelor's degree	$84,280
Computer Support Specialists	Computer support specialists provide help and advice to computer users and organizations.	Varies, certificates	$53,470
Computer Systems Analysts	Computer systems analysts study an organization's current computer systems and find a solution that is more efficient and effective.	Bachelor's degree	$88,740
Database Administrators	Database administrators (DBAs) use specialized software to store and organize data.	Bachelor's degree	$90,070
Information Security Analysts	Information security analysts plan and carry out security measures to protect computer systems.	Bachelor's degree	$98,350
Network and Computer Systems Administrators	Network and computer systems administrators are responsible for the day-to-day operation of computer networks.	Bachelor's degree	$82,050
Software Developers	Software developers create the applications or systems that run on a computer or another device.	Bachelor's degree	$105,590
Web developers	Web developers design and create websites.	Associate's degree	$69,430

Table 1: Computer Science job descriptions and pay scale [10]

who do not have exposure to careers or career planning results in graduates who know very little about what they can do professionally post-graduation, which puts the students at risk for poor professional choices and limited success. In a career-focused curriculum, the projects and research papers submitted by students help to highlight those skills in the resume which are mostly sought by employers.

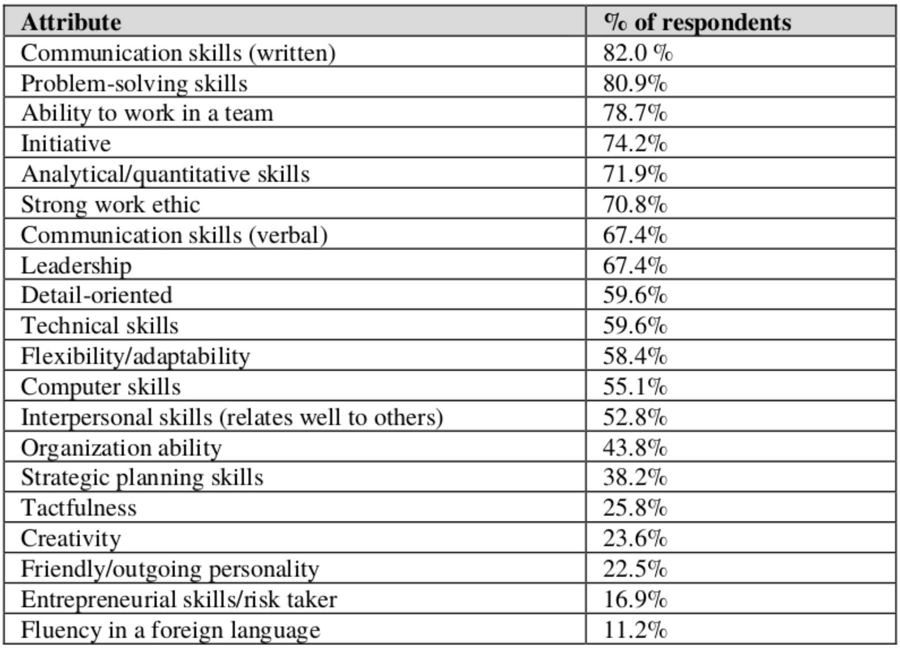
When examining the awareness and use of career services at a large Midwestern university, the research found that the career center resources were underutilized and that many students were having difficulty with career decisions [4]. Previous research regarding career services has identified that social stigmas and lack of awareness negatively impact students' usage of career service centers [11]. Kean University's Career Services Center helps all university students to find an internship or job to launch their career while Kean's School of Computer Science and Technology (SCS&T) focuses on computer science and information technology (CS/IT) students based upon each individual student's skills and interests, creating a culture of college and career readiness. The SCS&T works in collaboration with Career Services for student success. In SCS&T, career education is offered as part of the curriculum in the shape of a one-credit course. Other computer science programs elsewhere, such as Drexel, Carnegie Mellon, and Columbia, have integrated career education in the curriculum in a similar manner to prepare students for careers in a rapidly changing profession.

2 Gauging the importance of career education

Employers, regardless of industry, do have a preference for experience. Only 29% of college graduates had an internship or job during college [6]. It is evident from the figures that the number of students opting for internship is abysmally low while the preference of employers for students with internships over students without internships is overwhelming. The data only reinforces the fact that career readiness education must be an integral part of the university curriculum.

Recruiting has changed due to intense competition for graduates and technology. Organizations are utilizing the internet to attract, recruit, and select applicants more widely than ever before. There are many tools available to a hiring manager looking to fill an open role. From LinkedIn, Twitter, and Facebook networking groups, to the company's website and beyond, it has never been easier to access a deep pool of qualified candidates. Hiring managers are very focused on an individual's resume. For example, what sort of schooling did the applicant have? Had s/he won any awards? How much experience did a person have in the field? What were their technical qualifications? It is increasingly easy to create a workforce made up of a mix of permanent, freelance, outsourced and automated staffing options. All these options are the competition, so students must be even clearer and more compelling about their specific contributions [1].

The National Association of College and Employers (NACE) surveyed employers on what attributes they want to see on students' resumes. NACE [1] reported that the results of the Job Outlook survey (2019) showed that employers are looking for skills and qualities beyond a strong grade point average (GPA); more than four out of five employers indicated written communications skills (Table 2), problem-solving skills and an ability to work as part of the team are highly preferred. Another highly sought attribute in recent years is initiative and leadership. Other high value attributes include analytical and quantitative skills and a strong work ethic [8]. In addition, the Job Outlook survey (2019) by NACE stated employers look at the social media of applicants, such as LinkedIn, Twitter and Facebook.



Attribute	% of respondents
Communication skills (written)	82.0 %
Problem-solving skills	80.9%
Ability to work in a team	78.7%
Initiative	74.2%
Analytical/quantitative skills	71.9%
Strong work ethic	70.8%
Communication skills (verbal)	67.4%
Leadership	67.4%
Detail-oriented	59.6%
Technical skills	59.6%
Flexibility/adaptability	58.4%
Computer skills	55.1%
Interpersonal skills (relates well to others)	52.8%
Organization ability	43.8%
Strategic planning skills	38.2%
Tactfulness	25.8%
Creativity	23.6%
Friendly/outgoing personality	22.5%
Entrepreneurial skills/risk taker	16.9%
Fluency in a foreign language	11.2%

Table 2: Attributes Employers Seek on Candidate's Resume

Career education as defined by Kean University's School of Computer Science and Technology (SCS&T) is to prepare students who seek an internship or cooperative education during their sophomore year and is not limited to graduating and graduate students. The course curriculum is rendered in detail in section 3. In response to changes in employer recruiting, SCS&T adopted a more purposeful approach to promoting early career planning by students. This includes aligning academic and career counselling, making students aware of the relevant data e.g. current labor market and return on investment figures. Apart from that, alumni highlights, speeches, and knowledge exchange are good elements of career education and conversation between student-to-student and faculty-student interactions.

3 Curriculum for career education

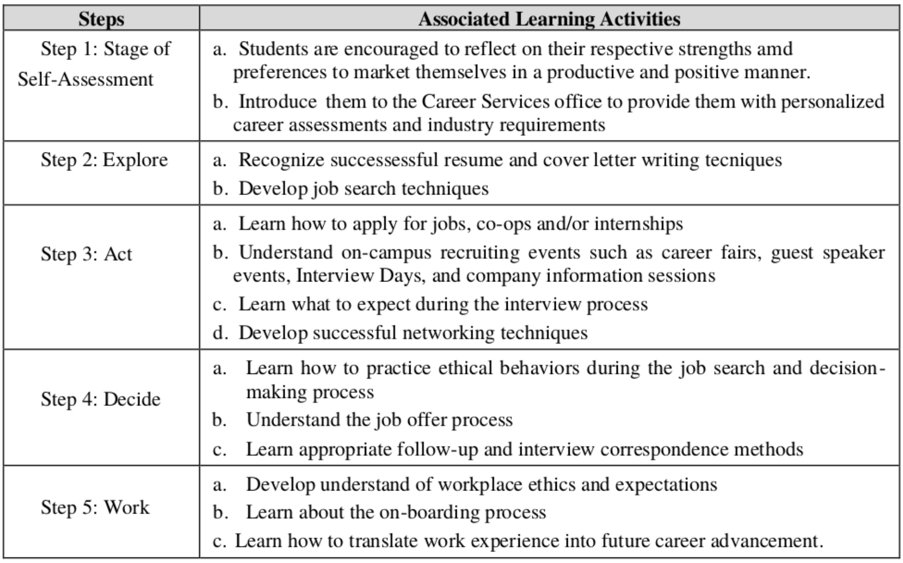
3.1 The Rationale for Developing a Curriculum

Kean University's School of Computer Science and Technology (SCS&T) provides pre-professional student chapters such as the ACM, ACM-W, and WiCyS with unique opportunities for networking, mentoring and bonding over common interests. The Women in Cybersecurity (WiCyS) group addresses the female workforce where women make up 11 percent of the global cybersecurity workforce [7]. Students interested in research can work with the faculty during one summer on campus and then apply for a National Science Foundation Research Experiences for Undergraduates (NSF REUs) opportunity to further develop their expertise as elaborated in Table 2. Other opportunities for participation include attending career fairs, being involved in a Hackathon, participating in ccoding challenges needed for interviews [9], and writing and solving algorithmic problems [3, 13].

Students are encouraged to become involved in the pre-professional chapters to develop their collaboration and leadership skills, as well as to learn more about their interests and meet a community of like-minded students. By infusing career development in early courses, with invitations to ACM, ACM-W, and WiCyS meetings, students are made aware of opportunities outside the classroom. Employers have stressed to faculty the importance of students being passionate about their future career, and these opportunities give students a chance to try out different areas within their major field. Guest or peer speakers talk about topics as varied as version control systems, new languages, or demonstrate robots that they have built. This allows students develop a self-identity within the profession. Early engagement in student groups also supports the development of a student culture that encourages undergraduate research or internships, which later moves students towards successful interviews and full-time professional positions or graduate school. By embedding career development in the curriculum and reinforcing it throughout the academic years, students develop confidence and expertise.

Student readiness is a shared responsibility and career readiness is a top priority in Kean's School of Computer Science and Technology where faculty, staff, and administrators support the growth of students by dedicating time and effort to student career readiness. Students are encouraged to remain flexible and adaptable in their career planning. If one approach is not working, students are encouraged to try a Plan B or alternative strategy and alternative vision. Therefore, in the development of a career education curriculum, both traditional and life-changing job searches are addressed [1].

A special emphasis on ensuring the career readiness of the students is used. For this, faculty and the contributing academic and administrative staff strive to contribute towards the professional nurturing of the students. The idea is to develop a curriculum that not only satisfies the educational needs but also contributes towards the aim of career readiness. A 5-step process has been developed, as listed in Table 3, which provides a learning spine for the curriculum.



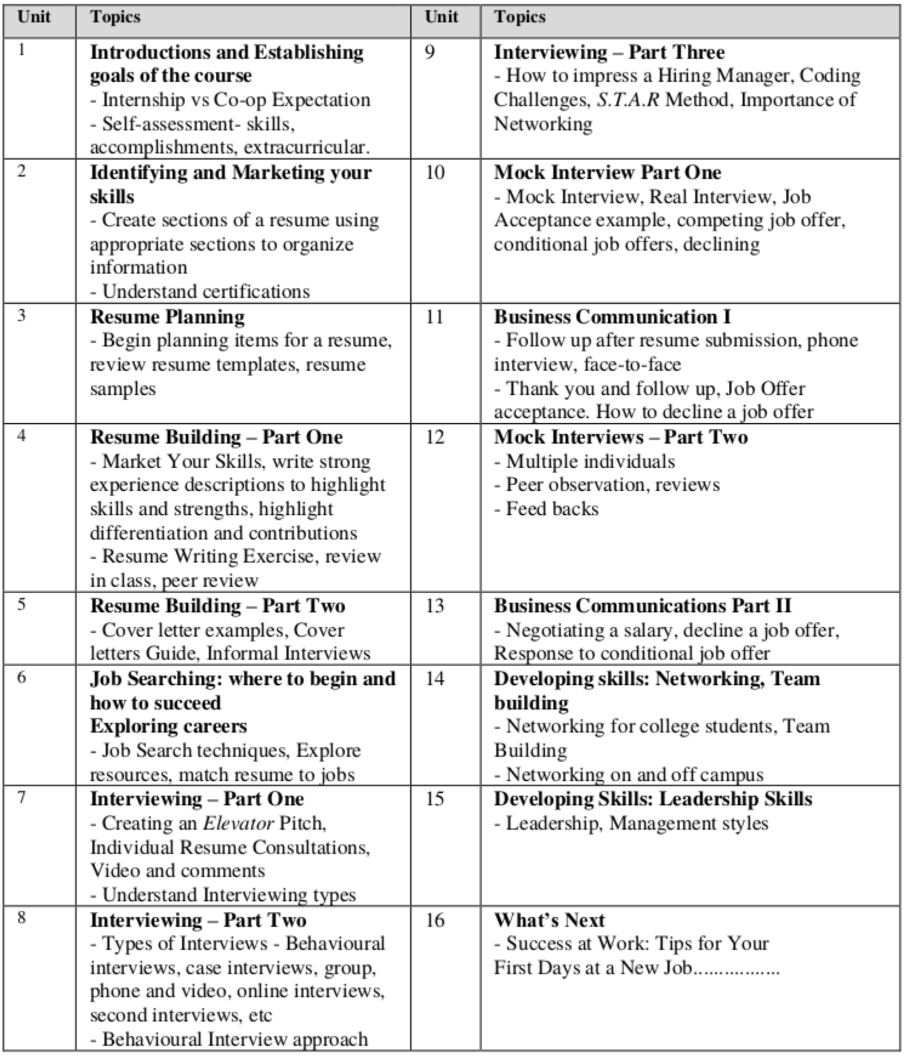
Steps	Associated Learning Activities
Step 1: Stage of Self-Assessment	a. Students are encouraged to reflect on their respective strengths amd preferences to market themselves in a productive and positive manner. b. Introduce them to the Career Services office to provide them with personalized career assessments and industry requirements
Step 2: Explore	a. Recognize successessful resume and cover letter writing tecniques b. Develop job search techniques
Step 3: Act	a. Learn how to apply for jobs, co-ops and/or internships b. Understand on-campus recruiting events such as career fairs, guest speaker events, Interview Days, and company information sessions c. Learn what to expect during the interview process d. Develop successful networking techniques
Step 4: Decide	a. Learn how to practice ethical behaviors during the job search and decision-making process b. Understand the job offer process c. Learn appropriate follow-up and interview correspondence methods
Step 5: Work	a. Develop understand of workplace ethics and expectations b. Learn about the on-boarding process c. Learn how to translate work experience into future career advancement.

Table 3: CS/IT Career Education Curriculum Steps and Learning Activities

3.2 Curriculum Design

The curriculum was designed after conversations with the program's Industrial Advisory Board, alumni, and current seniors going through the interview and hiring process. Table 4 details the specific units of the course.

Instruction is primarily through class lectures, student discussion and participation, and student research, using publicly available sources, such as the internet. Assessment includes projects and assignments, usually related to research specific firms or industry segments. Students are encouraged to select prospective firms they would like to work for and investigate those companies. Mock interviews are held, with students partnering with each other and alternating the roles of interviewer/interviewee. Finally, a career analysis report is expected from each student at the conclusion of the course, in which the student identifies roles they might like to have after graduation, firms where they might work, and outline the steps they will take in the coming semester to make their goals a reality. Learning outcomes include creating a professional resume, identifying career objectives, preparing a 30-second elevator pitch, practicing interviewing skills, employment negotiation, and how to accept a formal job offer. Table 4 details the specific units of the course.



Unit	Topics	Unit	Topics
1	**Introductions and Establishing goals of the course** - Internship vs Co-op Expectation - Self-assessment- skills, accomplishments, extracurricular.	9	**Interviewing – Part Three** - How to impress a Hiring Manager, Coding Challenges, *S.T.A.R* Method, Importance of Networking
2	**Identifying and Marketing your skills** - Create sections of a resume using appropriate sections to organize information - Understand certifications	10	**Mock Interview Part One** - Mock Interview, Real Interview, Job Acceptance example, competing job offer, conditional job offers, declining
3	**Resume Planning** - Begin planning items for a resume, review resume templates, resume samples	11	**Business Communication I** - Follow up after resume submission, phone interview, face-to-face - Thank you and follow up, Job Offer acceptance. How to decline a job offer
4	**Resume Building – Part One** - Market Your Skills, write strong experience descriptions to highlight skills and strengths, highlight differentiation and contributions - Resume Writing Exercise, review in class, peer review	12	**Mock Interviews – Part Two** - Multiple individuals - Peer observation, reviews - Feed backs
5	**Resume Building – Part Two** - Cover letter examples, Cover letters Guide, Informal Interviews	13	**Business Communications Part II** - Negotiating a salary, decline a job offer, Response to conditional job offer
6	**Job Searching: where to begin and how to succeed Exploring careers** - Job Search techniques, Explore resources, match resume to jobs	14	**Developing skills: Networking, Team building** - Networking for college students, Team Building - Networking on and off campus
7	**Interviewing – Part One** - Creating an *Elevator* Pitch, Individual Resume Consultations, Video and comments - Understand Interviewing types	15	**Developing Skills: Leadership Skills** - Leadership, Management styles
8	**Interviewing – Part Two** - Types of Interviews - Behavioural interviews, case interviews, group, phone and video, online interviews, second interviews, etc - Behavioural Interview approach	16	**What's Next** - Success at Work: Tips for Your First Days at a New Job.................

Table 4: CS/IT Career Education Curriculum Units

The class is taught by a faculty member and offered as a one semester credit for the satisfactory completion of a course that requires at least 15 hours (of 55 minutes each) of instruction and supplementary assignments.

4 Student Assessment and Feedback

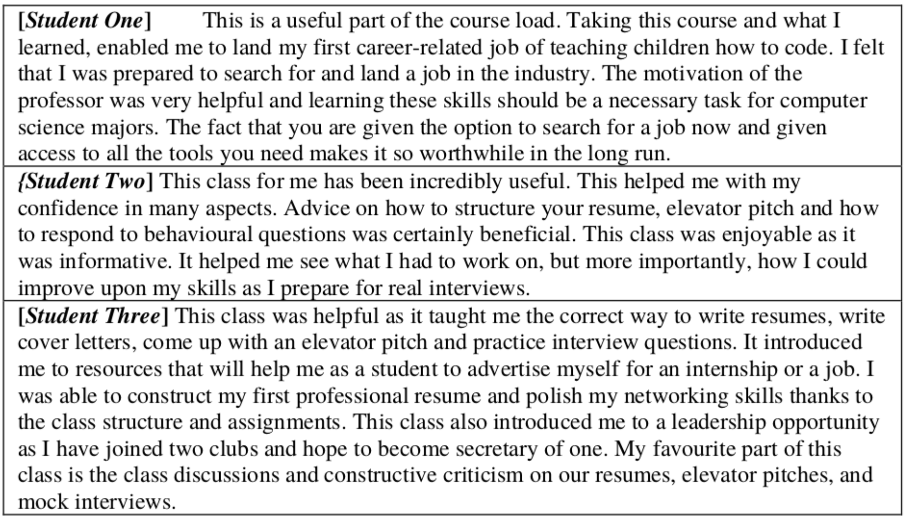
Career preparedness education should be an integral part of a student's education at higher levels. Every student, as in this case a Computer Science or Information Technology student, needs meaningful employment and career advancement. During college and even before, the student should think about giving an early start to his or her career, thus ensuring availability of a wide array of choices at the time of graduation. A late start, on the other hand, leads to a loss of many valuable opportunities and thus increases the challenges in finding employment even after getting good grades.

Internships are important, as they are investments in a person's future. The general trend is that, despite the faculty counselling, regardless of career education being a part of the course work, and irrespective of the requirement, an internship or related experience is required for graduation. It is unlikely that students think about this. Often, students are reluctant to pursue internships because they worry they'll be stuck doing menial tasks like getting coffee or making copies and certainly no one wants to work for free. The truth is, although not all internships pay in cash, they do pay in other ways. Table 5 provides student comments after taking a Career Education course.

5 Future Work

This study explores the experience at one public university in the School of Computer Science and Technology. The subject study is a semester where the student size of the class is smaller (approx. 15 students). As indicated from the student testimonials the course proved to be extremely beneficial in making them think about the career from an early stage. Twenty percent landed an internship because of this course, with data collection is still taking place. The SCS&T will continue to report data for upcoming semesters and gauge the outcomes in terms of student journeys toward success. Integrating career education content into class discussions provided an opportunity for students to share their questions about different jobs. Many students considered class discussions and practices to be the heart of the classroom learning. Integrating key career information in the course provided students with the opportunity to explore, connect, and apply career-focused questions and discussions.

Below are additional ways career content may be integrated into the classroom in the future:



[***Student One***] This is a useful part of the course load. Taking this course and what I learned, enabled me to land my first career-related job of teaching children how to code. I felt that I was prepared to search for and land a job in the industry. The motivation of the professor was very helpful and learning these skills should be a necessary task for computer science majors. The fact that you are given the option to search for a job now and given access to all the tools you need makes it so worthwhile in the long run.
*{**Student Two***] This class for me has been incredibly useful. This helped me with my confidence in many aspects. Advice on how to structure your resume, elevator pitch and how to respond to behavioural questions was certainly beneficial. This class was enjoyable as it was informative. It helped me see what I had to work on, but more importantly, how I could improve upon my skills as I prepare for real interviews.
[***Student Three***] This class was helpful as it taught me the correct way to write resumes, write cover letters, come up with an elevator pitch and practice interview questions. It introduced me to resources that will help me as a student to advertise myself for an internship or a job. I was able to construct my first professional resume and polish my networking skills thanks to the class structure and assignments. This class also introduced me to a leadership opportunity as I have joined two clubs and hope to become secretary of one. My favourite part of this class is the class discussions and constructive criticism on our resumes, elevator pitches, and mock interviews.

Table 5: CS/IT Student Comments after taking Career Education course

- Industry and site visits to observe company culture
- Ensuring employer engagement including workplace tours, job-shadowing experiences, career professional interviews and related opportunities
- Alumni mentorship programs
- Job site reviews and workshops
- Industry speaker series
- Student champions
- Provide the students with some data to prove return on investments (ROI) on college education to help understand their majors and credentials
- Provide industry certifications to make the students more marketable

A carefully designed and meticulously planned curriculum can ensure faster entry of the students into the workforce post-graduation and a higher likelihood of attaining a job in the desired field. As the program grows and extends to other departments, more opportunities will emerge to provide students with information about their potential and future outcomes attached to this integration of career education into the curriculum. If the future of a student can be shaped through such thoughtful guidance, it will encourage other students to seek career information in support of informed choices that will help their future success.

References

[1] National association of colleges and employers (nace).

[2] C. Billotte. Career services usage: An analysis of efficacy and contextual barriers' influence. ScholarWorks@BGSU 2019.

[3] Richard N Bolles. *What Color Is Your Parachute?* Ten Speed Press, 2017.

[4] B Busteed and S Seymour. Many college graduates not equipped for workplace success. *Gallup Business Journal*, 2015.

[5] J. Eagle and V. Tinto. The pell institute for the study of opportunity in higher education publications. Pellinstitute.org 2019.

[6] Nadya A Fouad, Amy Guillen, Elizabeth Harris-Hodge, Caroline Henry, Alexandra Novakovic, Sarah Terry, and Neeta Kantamneni. Need, awareness, and use of career services for college students. *Journal of career assessment*, 14(4):407–420, 2006.

[7] S Frost. The 2017 global information security workforce study: Women in cibersecurity. *Center for cybersecurity and education*, 2017.

[8] Lauren Levine. Then and now: How the hiring process has changed. https://hr.sparkhire.com/best-hiring-practices/now-hiring-process-changed 2017.

[9] Gayle Laakmann McDowell. *Cracking the coding interview.* 2015.

[10] Bureau of Labor Statistics. *Computer and Information Technology Occupations: Occupational outlook handbook.* https://www.bls.gov/ooh/computer-and-information-technology/mobile/home.htm, 2019.

[11] Purdue University. Gallup-purdue index report 2014. Gallup.com 2019.

[12] Beth M Schwartz, Virginia R Gregg, and Mark McKee. Conversations about careers. *Teaching of Psychology*, 45(1):50–59, 2017.

[13] SS Skiena. The algorithm design manual, 2008.

An Interdisciplinary Approach to Detecting Empathy Through Emotional Analytics and Eye Tracking*

Jami L. Cotler[1]*,Luis Villa*[1]*, Dmitry Burshteyn*[2]*, Zachary Bult*[3]
Garrison Grant[4]*, Michael Tanski*[5]*, Anthony Parente*[5]
[1]*Computer Science,* [2]*Psychology,* [3]*Sociology*
Siena College, Loudonville, NY 12211
{jcotler,lc19vill,dburshteyn,zb20bult}@siena.edu
[4]*Let's Chat About It*
garrison@letschataboutit.com
[5]*Dumbstruck*
{michael, a.parente}@dumbstruck.com

Abstract

The aim of this interdisciplinary study was to bring together different perspectives to discover if detecting empathetic emotional reactions is possible. This area of research has received recent attention from the computer science, human-computer interaction and psychological research communities. The research team consisted of three students; a computer science, sociology and marketing major. The team worked to understand the complexities of detecting emotions based on facial movement. The team collected time stamped facial emotional data from 210 participants as they watched a video clip from the popular movie depicting bullying behavior towards a disabled person. The results demonstrated significant before-and- after mean differences in emotions that are characterized as empathic towards the main character for the bullying events, which is a promising start to detecting empathic reactions. Each student brought a different perspective from their majors resulting in an educational experience that transcended learning about emotional analytics.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

The team started the project by agreeing on a common understanding and definition of empathy. The definition used for this study is sharing in another person's affective state or being able to take the perspective of and/or feel the emotions of another person, essentially to have the ability to put oneself in another's shoes, which is also called Affective Empathy [3, 12, 13, 1, 19]. Another type of empathy is cognitive empathy, which is the understanding of how another person feels, but without actually experience that feeling [14]. The team select affective empathy because the aim of the study was to see if the participants felt what the main character most likely felt. After becoming familiar with the concept of empathy the team learned how to read micro expressions by completing the Ekman library micro expression training tool [9]. Discussed more in the methods sections, exposure to working with human subjects, survey design and a technical understanding of the tools used were part of the study development process.

Detecting empathetic emotional reactions is a growing area of research in the field of Human-Computer Interaction [7, 14, 19]. Moreover, measuring empathy has also garnered interest in the psychological and neuroscientific research communities [2, 6]. Emotional detection has captured the attention of researchers in the area of Affective Computing [17, 15] Using technology to detect emotion has been realized in multiple applications including; text [4], speech or voice [20], facial expressions[11, 15], body gestures and movement [18], and emotion from physiological states [14]. State of the art facial recognition technology has been successful in predicting base emotions (joy, sadness, fear, anger, surprise, and disgust as shown in Figure 2) with high levels of accuracy [16]. This study extends facial expression detection to explore the detection of empathic states. In this study, we used the commercial program dumbstruck (dumbstruck.com) to collect facial expressions and attention (eye tracking) data. The database that Dumbstruck uses is based on the validated Facial Action Coding System (FACS) [5, 9, 10] and will be discussed in detail the methods section.

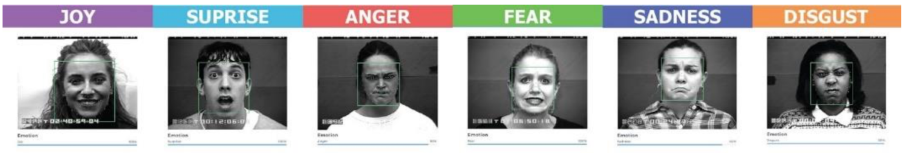


Figure 1: Primary emotions as detected from dumbstruck algorithms [8]

2 Methods

To understand the intricacies of working with human subjects, all team members went through the CITI human subject research program and helped develop the Institutional Review Board study application. The team picked the video clip and developed a short survey. The video clip from the movie Forrest Gump was selected for two primary reasons. First, the clip depicted bullying behavior based on a disability. Second, it clearly demonstrated both a bullying and a bullying averted event. The survey was limited to six questions that integrated different perspectives on the team and helped identify noteworthy questions. The survey responses and the corresponding relationships with the collected data will be investigated in our future studies. Dumbstruck facilitated the distribution of the video and survey questions through their participant network. Two hundred and ten participants watched the short video clip and completed the survey questions. The participants watched the movie clip while the camera on their computer recorded their responses. Dumbstruck technology recorded the facial emotional state and eye tracking data of each participant every millisecond. Our sample demographics are shown in Figure 2.

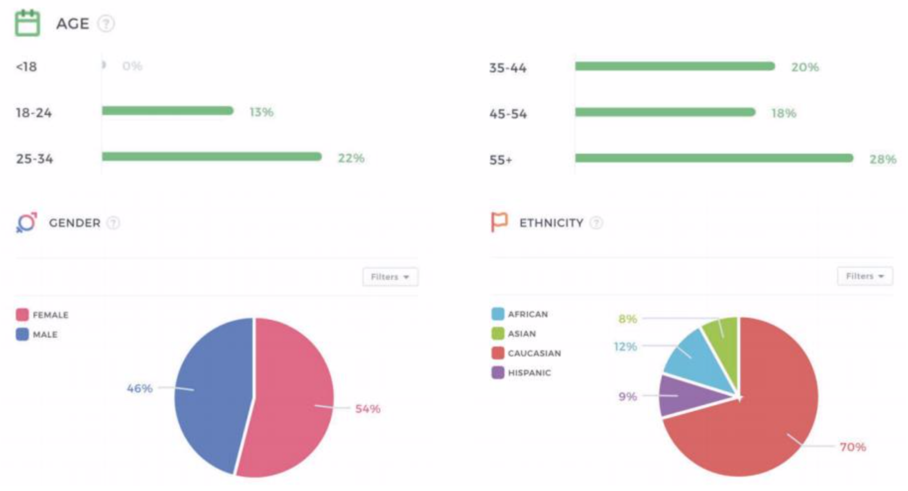


Figure 2: Study participants breakdown by age, gender and ethnicity

The “bullying event” occurs when the bullies hit the main character (Forrest Gump) in the head with a rock and Forrest subsequently ends up on the ground bleeding as a result of his injury. The “bullying averted event” commences when Forrest gets up to his feet, starts to run away and successfully avoids further bullying by breaking free of his leg braces and getting away. As a team we hypothesized that participants would exhibit differences in emotional responses to both bullying and bullying averted conditions in comparison to their baseline condition. Our hypothesis was to evaluate the differences in emotional states indicative of empathic behavior pre and post bullying events.

3 Results

An Analysis of Variance (ANOVA) was conducted to compare an equal number of epochs (time segments) in baseline condition (BASE) and conditions depicting bullying stimulus (BC), and bullying averting stimulus (BA). An ANOVA yielded statistically significant differences for the following variables:

Mood F (2, 5656)=15.494, p<.0001,
Joy F (2, 5656)=34.621, p<.001,
Disgust F (2, 5656)=15.303, p<.0001,
Anger F (2, 5656)=6.157, p<.002.

Of interest, no significant changes in participants' attentiveness measured through eye tracking were observed between baseline and experimental conditions.

Tukey HSD post hoc test was performed to compare mean differences between groups and indicated statistically significant differences between the following conditions:

Mood Baseline condition (Base) χ=-.85 and Bullying Averted (BA) χ=.2.18 , p<.002, and Bullying Averted (BA) condition χ=.2.18 and Bullying Conditions (BC)) χ=-2.50, p<.0001;
Joy BASE χ=2.91 and BA χ=6.05, p<.0001, and BA χ=6.05 and BC χ=3.25, p<.0001;
Disgust BASE χ=1.04 and BA χ=1.95, p<.0001, and BA χ=1.95 and BC χ=1.77, p <.0001, and BASE χ=1.04 and BC χ=1.77, p<.0001;
Anger BASE χ=3.09 and BA χ=3.91, p<.006, and Base χ=3.09 and BC χ=3.92, p<.006.

4 Discussion

The results presented above and in Figure 3 demonstrate statistically significant differences in key emotional responses between experimental and baseline conditions. These differences in participants' empathic responses point to the effectiveness of our experimental manipulation and the possible role of innovative technological tools in accurately detecting these types of responses.

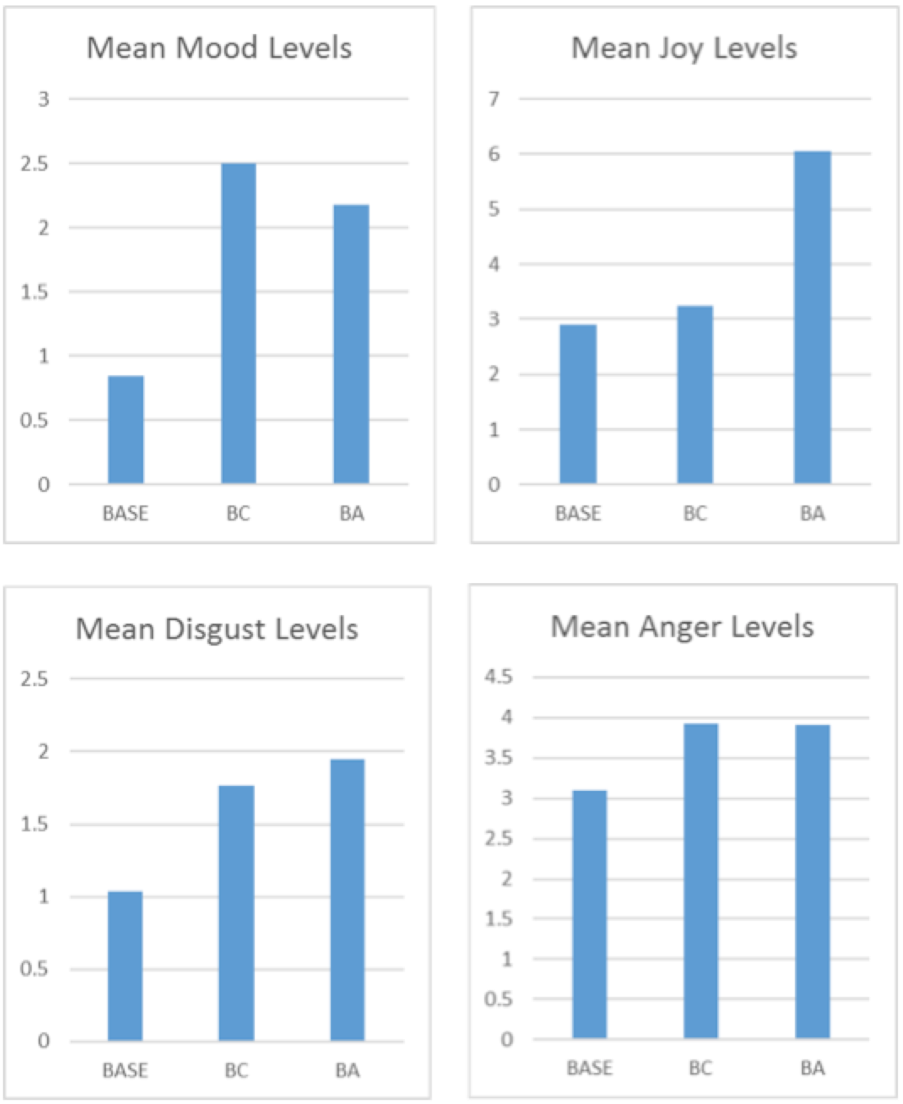


Figure 3: Mean comparisons between Baseline (Base), Bullying (BC) and Bullying Adverted (BA) Conditions

Some challenges and limitations we encountered include the type of stimulus we selected. This type of movie depicts fictional characters and is not necessarily representative of a realistic situation. Further, this is a popular movie, which could have had an impact on the study results. Designing future studies with better experimental conditions would be instrumental to the generalization of our results. However, despite the limitations of our stimulus, our findings are powerful and supported by the high level of statistical significance between the groups. At the same time, we would like to be cautious to not over generalize our findings until better control groups and more realistic experimental conditions are selected for future experiments.

5 Benefits of interdisciplinary approach

The dynamic of an interdisciplinary team with a range of academic focuses brought a unique team perspective and positively influenced our study. The sociological perspective helped in wording of the survey questions and resulted in participants achieving a high completion rate. In the analysis of the results, this perspective aided in the team's understanding and ability to differentiate and recognize individuals' emotions. The sociology major shared the perspective on human interactions as well as his expertise in research methodology that helped construct the survey used in this study.

The marketing perspective helped the team find new ways to visualize and interpret the data, and introduced an effective way to present the findings. Being able to produce a clear and concise summary of the data collected gave the project traction. The team delivered several poster presentations and elevator pitches for this project. The elevator pitches were constructed to describe the project in-depth without overwhelming the audience with excessive level of details. The visual aids used in the study including charts and graphics allowed for the audience to better understand the significance of the findings.

The computer science student was able to expound technology specifics and explain complex concepts to the other team members. The student's contribution led to the better understand of the role of camera angles, lighting, and the effect of the distance between the participant and the camera on the quality of data recorded. The computer science student learned how the Cohn- Kanade AU-Coded Facial Expression Database was used to validate the use of the dumbstruck coding algorithms to detect the primary emotions. The database that Dumbstruck uses is based on the validated Facial Action Coding System (FACS) [5, 10, 12]. The FACS-coded sequences are a snapshot of a collection of the muscles that are being used when an individual has an emotional reaction. Each part of the face has its own Action Unit or AU number [10]. The collection of certain muscles being used simultaneously is what gives the emotion detection. Active Appearance Models (AAM) take the face and plots points and then takes the measurements from each landmark and transfers that information over to the Support Vector Machine (SVM) to then predict the emotion that is being experienced by the participant [10]. The SVM is where the machine learning comes into play, the support vector machine makes decisions as to what emotion is being shown. The SVM classifies the emotion by compressing (regression) the information that is being given by the Active Appearance Models (AAM) [14]. Dumbstruck has thoroughly trained and tested their SVM and machine learning algorithms for commercial and research use.

Overall, the students reported very much enjoying this interdisciplinary experience and team dynamic and felt that they complemented each other in a way that increased the quality of the experience and study. They were each able to make contributions that suited their respective fields, and along with that came the general support and enthusiasm for the project.

6 Conclusion and future work

The goals of our future research are to analyze the responses of the six survey questions and how they relate to the data collected and conduct additional studies to discover potential cognitive-emotional substrate of empathy. In addition to using movie clips that introduce a potential confounding familiarity variable into the experimental setting, we intend to use live experimental stimuli while empathetic behavior is monitored and recorded.

The facial recognition and eye-tracking data collected from the 210 participants indicates significant differences between empathy induced conditions and baseline conditions for several combinations of emotional states. Future investigations will seek additional clarity to further the scientific understanding in identifying and recognizing such responses across multiple situations that may elicit empathetic responses. It is important to investigate if these multi-emotional responses remain stable or change as a result of a specific stimulus in both magnitude and duration. Our future studies will focus on validation of the empathy variable and expanding our knowledge of the effectiveness of facial recognition and eye tracking technology in the accurate detection of empathetic states.

We look forward to our future work in continuing the development of innovative methods for detecting empathetic responses. This research also has the potential to identify possible bullying prevention opportunities.

References

[1] Empathy. https://psychologydictionary.org/empathy/ accessed April 11, 2018.

[2] Zachary D Bloom and Glenn W Lambie. The adolescent measure of empathy and sympathy in a sample of emerging adults. *Measurement and Evaluation in Counseling and Development*, pages 1–15, 2019.

[3] Cambridge English Dictionary (2018). Definition of empathy. Retrieved from: https://dictionary.cambridge.org/us/dictionary/english/empathy.

[4] Ankush Chatterjee, Umang Gupta, Manoj Kumar Chinnakotla, Radhakrishnan Srikanth, Michel Galley, and Puneet Agrawal. Understanding emotions in text using deep learning and big data. *Computers in Human Behavior*, 93:309–317, 2019.

[5] Jeffrey F Cohn, Karen Schmidt, Ralph Gross, and Paul Ekman. Individual differences in facial expression: Stability over time, relation to self-reported emotion, and ability to inform person identification. In *Proceedings. Fourth IEEE International Conference on Multimodal Interfaces*, pages 491–496. IEEE, 2002.

[6] Michel-Pierre Coll, Essi Viding, Markus Rütgen, Giorgia Silani, Claus Lamm, Caroline Catmur, and Geoffrey Bird. Are we really measuring empathy? proposal for a new measurement framework. *Neuroscience & Biobehavioral Reviews*, 83:132–139, 2017.

[7] David Coyle, Gavin Doherty, Mark Matthews, and John Sharry. Computers in talk-based mental health interventions. *Interacting with computers*, 19(4):545–562, 2007.

[8] Dumbstruck (2018). Tutorial from company dashboard. Retrieved from: dumbstruck.com.

[9] P. Ekman. Micro expression training tools. Paul Ekman Group. https://www.paulekman.com/micro-expressions-training-tools/ 2019.

[10] Paul Ekman, Wallace V Friesen, Maureen O'sullivan, Anthony Chan, Irene Diacoyanni-Tarlatzis, Karl Heider, Rainer Krause, William Ayhan LeCompte, Tom Pitcairn, Pio E Ricci-Bitti, et al. Universals and cultural differences in the judgments of facial expressions of emotion. *Journal of personality and social psychology*, 53(4):712, 1987.

[11] Rosenberg Ekman. *What the face reveals: Basic and applied studies of spontaneous expression using the Facial Action Coding System (FACS).* Oxford University Press, USA, 1997.

[12] Robert Eres et al. Scanning for empathy. *Australasian Science*, 37(2):30, 2016.

[13] Xinbo Gao, Ya Su, Xuelong Li, and Dacheng Tao. A review of active appearance models. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(2):145–158, 2010.

[14] Jose Maria Garcia-Garcia, Victor MR Penichet, and Maria D Lozano. Emotion detection: a technology review. In *Proceedings of the XVIII International Conference on Human Computer Interaction*, pages 1–8, 2017.

[15] Maryam Hasan, Elke Rundensteiner, and Emmanuel Agu. Automatic emotion detection in text streams by analyzing twitter data. *International Journal of Data Science and Analytics*, 7(1):35–51, 2019.

[16] Patrick Lucey, Jeffrey F Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, and Iain Matthews. The extended Cohn-Kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression. In *2010 ieee computer society conference on computer vision and pattern recognition-workshops*, pages 94–101. IEEE, 2010.

[17] Adria Mallol-Ragolta, Maximilian Schmitt, Alice Baird, Nicholas Cummins, and Björn Schuller. Performance analysis of unimodal and multimodal models in valence-based empathy recognition. In *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*, pages 1–5. IEEE, 2019.

[18] R Santhoshkumar and M Kalaiselvi Geetha. Human emotion recognition using body expressive feature. In *Microservices in Big Data Analytics*, pages 141–149. Springer, 2020.

[19] Petr Slovak. Supporting teaching and learning of situational empathy by technology. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, pages 315–318. 2014.

[20] Adib Ashfaq A Zamil, Sajib Hasan, Showmik MD Jannatul Baki, Jawad MD Adam, and Isra Zaman. Emotion detection from speech signals using voting mechanism on classified frames. In *2019 International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, pages 281–285. IEEE, 2019.

Project-Based App Programming: Tools and Techniques for a Successful Novice-Focused App Development Course

Michael Makutonin[1] *and Samuel Chen*[2]
[1]*Computer Science Department*
St. Bonaventure University
St. Bonaventure, NY 14778
michael.makutonin@gmail.com
[2]*College of Medicine*
SUNY Upstate Medical University
Syracuse, NY 13210

Abstract

This paper presents an introductory app development course taught over two semester-long sections to a mix computing majors and novice programmers as a general elective. This project-based course aimed to teach the requisite skills for application development in JavaScript in a manner equally engaging to computing majors and novices. Students were expected to complete practical exercises and functional projects following short daily lectures written in a reference-style format. A complex final application was built by individuals or groups utilizing Scrum, and showed the efficacy of tools and instruction pedagogy employed. Students were able to look up syntax and reference external sources when programming and could apply their understanding to learn other concepts. There was no significant difference between novice and computing major performance, as both groups could reference and integrate multiple programming concepts to solve problems posed without an over-reliance on specific syntax they were taught.

1 Introduction

Due to the increasing implementation of tools and practices traditionally confined to CS careers in business and related fields, there has been increasing demand for coding skills in traditionally non-technical disciplines [11]. In fact, students graduating with degrees as disparate as philosophy and business are

all somewhat likely require skills like excel queries and analytics mastery [11]. Extensive research in novice computer science (CS) education has found common difficulties faced by students starting to learn programming in the university setting [1, 10, 12]. These techniques are not universally implemented in CS curricula, and CS remains an inaccessible field in the public view.
This paper describes an introductory programming course taught in the fall and spring semesters of 2019 to a mixture of computing and non-computing majors. This course assumed no prior programming experience, and employed innovations recommended in prior research [1, 3, 4, 10, 12, 14, 15], implemented current business practices [13], and adapted novel technologies [6, 8] to teach students JavaScript (JS). Students were then meant to apply this knowledge by programming an complex application after 5-7 weeks of instruction.

1.1 Grounding in Prior Research

Over the last couple of decades, researchers have consistently investigated the challenges faced by novice programmers. Students tend to have an adequate understanding of syntax but cannot apply multiple syntactic elements to solve larger problems [7, 12]. Despite this, research reviewed by Robins et al. suggested that many textbooks and computing courses place a heavy emphasis on learning the syntax of a specific language [10].
Multiple surveys have attempted to pinpoint pain points of students learning introductory computing [1, 7]. Many students significantly prefer to learn using hands-on coding examples [7] and debugging code [1].
Studies have further shown that student engagement is an important indicator of educational outcomes [5, 15]. Engagement can be increased by giving students a stake in what they learn and allowing them to direct the work that they complete. Students lose engagement when they are given inadequate resources or forced to produce work that does not enhance their learning [13].
Literature has also contrasted novice and expert programming behaviors [3, 4, 10]. Novices tend to spend more time searching for information when solving problems, while experts spend more time defining problems and narrowing search topics before search and implementation of solutions [4]. Novice programmers in the workforce also tend to have problems communicating solutions and interacting with colleagues in a productive manner, suggesting a lack of experience applying social skills and collaborating on long-term projects [3].

1.2 Grounding in Industry Practices

Industry environments tend to focus on process efficiency and reliability. In software development, practices are designed to increase efficiency of programmer communication, work, and research/learning. In a project-centric course,

it is important to teach students these priorities, since inefficiency results in time wastage on tasks unrelated to development. This wasted time would not contribute to student learning and could result in increased student frustration and less tangible progress.
Scrum, a development practice used by over 50% of programmers at a variety of established software companies [2, 9] increases the efficiency of programming teams and individuals by increasing communication and transparency of the development process. Scrum iteratively focuses developer efforts on prioritized list of tasks to be completed in a set period, which is then reprioritized based on business needs. Scrum also increases development efficiency through continuous improvement tasks that are built into every development period.

2 Instruction Methodology

2.1 Course Meeting Structure

Course meetings occurred twice or thrice for five cumulative hours a week, and usually involved a 20-minute lecture and an exercise work portion. Lectures would introduce new material and review prior concepts, while exercises would require students to reference and synthesize new concepts in order to solve programming challenges. Students could collaborate on exercises if desired, though each student had to submit an individual solution a week after the exercise was assigned. The final 3-5 weeks of the course were devoted to work on final projects of interest to teams of students.

2.2 Implementation of Best Practices

Function over Style

Due to difficulties novices have understanding programming concepts rather than syntax [7, 12], this course emphasized the functionality of programs over syntax and style. Most exercises were unit tested, and students were given full credit if exercises passed unit tests. For projects other than the final, students were given clear minimal acceptance criteria, and earned full marks upon meeting said criteria. Only when students mastered programming concepts were they encouraged to follow better syntactical practices. To assess that students mastered best practices during the course, their final projects were assessed for best practices as part of the project grade.

Familiarization via Debugging

JS involves concepts that are difficult for students to grasp [7] and implement immediately. To help students become familiar with difficult concepts with less frustration, some exercises involved debugging existing code rather than implementing new functions. Debugging was meant to cause students to reference syntax and concepts frequently while building mental models of how the intended code was supposed to work.

Student Engagement

Students were encouraged to create projects whose content and purpose was meaningful to the student. Efforts of students who developed application beyond the minimal requirements were then validated by classmates' praise during project presentations. In order to decrease student frustration, students were made aware of resources available to them prior to the start of every project. Students were graded exclusively on performance of their code – lecture attendance was not mandatory (except during sprint demos for the final project). Scrum practices were taught to reduce student inefficiency and frustration on large projects, and were implemented by all teams during final project development.

Project Focus

A project-focused course has multiple benefits. It increases student engagement by offering maximal flexibility and a feeling of tangible accomplishment to student work. Project work also has a didactic benefit, since it allows students to learn and be assessed via implementation rather than proxies like exams or attendance. Group projects can build professional communication skills between team members, while individual projects encourage students to become self-driven and creative with their ideas.

2.3 Adaptation of Novel Tools

The adoption and adaptation of the following tools was critical to the implementation of the best practices described above. Tools were meant to enhance engagement, learning, and communication throughout the course.

Lecture Tools

Observable Notebooks [8] were used to create referenceable and interactive lectures. Lectures were written in a textbook format: concepts were explained using text and graphics so that students could read lectures independently as instructors emphasized important points. Students could view the notebooks on their personal computers during the lecture, which allowed them to follow along and code inside the notebook during lecture. Exercises were included in the notebook, and forked by students when complete. These exercises came

bundled with pre-written unit tests that allowed students to see if their code met functional acceptance criteria before submission.
Because Observable Notebooks could not effectively display React, Codesandbox [6] was used to teach React. Codesandbox projects could be embedded into Observable Notebooks for easy reference and manipulation during the lecture.

Language

JS was taught because its primary applications are visual and tangible. This allowed students to create code that produced interactive results, which could increase engagement. JS allowed for exposure to facets of imperative, scripting, and object-orientated paradigms, which may help them when learning other languages in the future. When utilizing the React library, JS could be used to create mobile, web, or desktop apps, increasing student flexibility in implementing the final project.

3 Results and Discussion

This course was offered in the spring and fall of 2019, with section sizes of 7 and 5 students, respectively. The spring section was composed entirely of computing majors, while the fall section contained two computing and three non-computing majors. Performance between sections was compared to control for variation between offerings and evaluate accessibility to novice programmers. All data discussed below were collected with IRB approval.

3.1 Assessment Performance

Exercise completion was assessed as a measure of student understanding of concepts taught in lecture. Though some modifications were made to exercises assigned in the fall semester, they remained comparable across both semesters. Notably, the fall cohort had better completion rates for exercises, especially as concepts became more complex (Figure 1).

3.2 Project Performance

Several projects were assigned throughout both spring and fall semesters. Student performance in each project is discussed below.

Control Project: HTML Assignment

Students in both cohorts were asked to build an HTML webpage (with little instruction in HTML) at the beginning of the course. This project measured

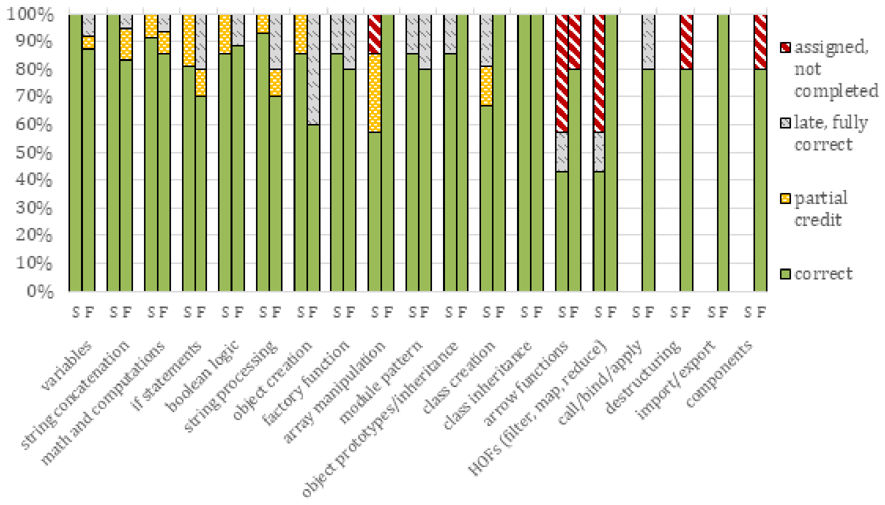


Figure 1: A comparison of exercise completion rates in spring (S) and fall (F) cohorts. Concepts missing S bars were not tested via exercises for that cohort, but were implemented in final projects.

baseline programming skill of students and their ability to learn a simple new language. Students were given online resources to reference and minimal out-of-class help upon request.

In the spring semester, all students turned in websites on-time and were able to incorporate multiple HTML constructs into creative, individual projects. Students in the fall semester were asked to work on the HTML project in self-selected pairs: the first pair, made up of two novices, delivered the project two weeks late, after one of them dropped the course due to difficulty balancing it with other commitments. The other two pairs, made up of one computing major and one novice, were able to complete their projects on time. However, when feedback was collected from those groups, it was found that the projects had been coded mainly by the computing major.

Additional Projects in Fall Cohort

Students in the fall cohort completed two additional projects. First, students had to extend their websites to call publicly available APIs. For this project, students separated themselves into groups of two and three students, containing one computing major each. Here, a similar pattern was observed in student feedback, with most of the functionality being implemented by the computing major when the groups were working together. However, in one group, when the two non-computing group members were asked to work on the project without the aid of the computing major for an hour, they were able to implement most of the project alone. This incident showed that the non-computing majors were able to implement projects without assistance from computing majors but were more comfortable relying on a computing major's expertise when available. Keeping this event in mind, students were asked to complete the final in-lecture project independently. This project involved the creation of a random Pokémon generator using a React template and assessed students' synthesis of several concepts (data structures and manipulation, higher order functions, API calls, promises, and React/React Components) from the lecture. Because this project was assigned during the second-to-last lecture, students expressed a lack of motivation to complete the project – two projects were turned in on time, two were late, and one was not completed. However, all but one student were able to complete the project, showing an ability to apply their understanding of complex programming concepts to an implementation.

Final Project

In the final three weeks of Fall and five weeks of Spring, students self-organized into teams and worked on full-scale applications as a practical learning assessment. This project was worth 50% of the student grade for the course.
In the spring cohort, two groups completed and presented projects to a panel of external faculty. The first presented a Yelp-like web application. The second completed a mobile quiz game meant to familiarize incoming university freshmen with campus landmarks by having them identify photos to earn points.
Both projects functioned during the demonstration and received positive feedback. When projects were assessed for their adherence to best practices, code was readable and logically structured overall, which showed that teams were able to follow some best practices in their implementations.
Group members contributed to development equally, with one notable exception: a student in the quiz app team did not communicate well with his team and would not complete any programming portion of the project.
Students who developed mobile apps selected to use React Native and were able to learn the library with minimal input and aid from instructors. Though JS concepts translate to the construction of apps using React Native, the library is different enough from the React library taught in lecture that it should have taken significant effort to learn.
Due to the deadline for this paper submission, the outcomes of the final projects for the fall course have yet to be assessed. However, at the equivalent milestones, the fall cohort's results have surpassed those of the spring cohort consistently. Due to their use of an iterative approach, all three teams in the fall cohort

currently have working applications: a 2D web-based adventure game, a web-based “adulting” application, and a mobile educational game for children. The games were developed by individual computing majors, while the web app was developed by the three novices in the cohort.

3.3 Efficacy of Tools

Student success could not be quantitatively attributed to a particular tool or technique. Nevertheless, there were strong trends observed in both cohorts which recommended several tools and frameworks for use in an introductory programming offering.

Observable Notebooks

Observable notebooks were essential for instruction of this course. They allowed students to execute and modify code in real time and made programming a primarily interactive and visual experience. In addition, since the notebooks were written in a reference format, and since exercises were included directly in the notebooks, they allowed students to develop skills in reading documentation and looking up information. Students stated that they enjoyed learning using notebooks, with multiple comments attributing learning outcomes to the format of the lectures. When implementing exercises and assignments, students identified the programming constructs necessary to solve a problem and referenced notebooks/search engines to identify appropriate syntax.

Scrum

Students adopted Scrum principles in order to develop their final projects, including sprint demos and other Scrum ceremonies that helped organize their iterative development process. Students stated that Scrum made it easier to develop their apps and allowed them to spread their work more evenly and sustainably throughout the project work period. In addition, students stated that iterative development allowed them to include useful features in their apps that they had not thought of at the beginning of their development process.
Though many of the graduates of this course will not develop projects in teams, Scrum allows them to organize their own work on individual projects and finds application outside of a coding environment in other project management applications. Some students have reported using Scrum significantly on passion programming projects and projects unrelated to computer science.

Daily Exercises/Projects

We saw the impact of exercises and implementation of concepts on learning when reviewing the case of an auditor in the spring semester. The auditor attended each lecture and paid attention during lecture but refused to do any of the daily exercises. The auditor stated to instructors around the middle of the semester that he did not understand material being covered, despite help from instructors and review of old notebooks. This auditor was not able to program any portion of his team's project at the end of the semester and served only in an organizational capacity for his team.

4 Conclusion

The aim of the course presented in this paper was to teach programmers working knowledge in a new language that they could use to develop complex applications. Using techniques found in industry and recommended by prior research, as well as tools applied in a novel fashion to undergraduate CS education, students were taught JS in 5-7 weeks of lecture. Regardless of prior programming background, students could create small-scale (but complex) applications after this lecture period.

5 Acknowledgements

The authors would like to thank the St. Bonaventure School of Arts and Sciences for running this course, as well as individuals who helped in this paper's conceptualization (including Ashley Delvento, Dr. Remah Alshinina, Dr. David Levine, and Dr. Xiao-Ning Zhang, among others).

6 Additional Resources

All materials for this course (including final project links, lectures, and syllabi, can be found at http://bit.ly/ccscne_paper_materials.

References

[1] Marzieh Ahmadzadeh, Dave Elliman, and Colin Higgins. An analysis of patterns of debugging among novice computer science students. *SIGCSE Bull.*, 37(3):84–88, June 2005.

[2] Scrum Alliance. The state of scrum report 2017 edition. In *Technical Report. Scrum Alliance*, page 31. 2017.

[3] Andrew Begel and Beth Simon. Novice software developers, all over again. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, pages 3–14, New York, NY, USA, 2008. ACM.

[4] Saskia Brand-Gruwel, Iwan Wopereis, and Yvonne Vermetten. Information problem solving by experts and novices: Analysis of a complex cognitive skill. *Computers in Human Behavior*, 21(3):487–508, 2005.

[5] Robert M Carini, George D Kuh, and Stephen P Klein. Student engagement and student learning: Testing the linkages. *Research in higher education*, 47(1):1–32, 2006.

[6] Codesandbox. https://Codesandbox.io/.

[7] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *SIGCSE Bull.*, 37(3):14–18, June 2005.

[8] Observable notebooks. https://observablehq.com/.

[9] Mark C Paulk. A scrum adoption survey. *Software Quality Professionals, ASQ*, 15(2):27–34, 2013.

[10] Anthony Robins, Janet Rountree, and Nathan Rountree. Learning and teaching programming: A review and discussion. *Computer science education*, 13(2):137–172, 2003.

[11] Matthew Sigelman, Scott Bittle, W Markow, and B Francis. The hybrid job economy: How new skills are rewriting the dna of the job market. *Burning Glass Technologies*, 2018.

[12] James C. Spohrer and Elliot Soloway. Novice mistakes: Are the folk wisdoms correct? *Commun. ACM*, 29(7):624–632, July 1986.

[13] Jeff Sutherland and JJ Sutherland. *Scrum: the art of doing twice the work in half the time.* Currency, 2014.

[14] Leon E. Winslow. Programming pedagogy- a psychological overview. *SIGCSE Bull.*, 28(3):17–22, September 1996.

[15] Linda G Wyatt. Nontraditional student engagement: Increasing adult student success and retention. *The Journal of Continuing Higher Education*, 59(1):10–20, 2011.

Using Gamification to Encourage Student Success[*]

Patricia Morreale and Nohelia Diplan
School of Computer Science and Technology
Kean University
Union, NJ 07083
{pmorreal, diplann}@kean.edu

Abstract

Increasing computer science program enrollments have impacted college and university programs in many ways, usually resulting in additional curriculum offerings and larger courses enrollments. Student support, such as faculty mentoring, previously offered one-on-one or in small group settings, has been impacted. First generation college students and students without knowledge of professional careers are adversely impacted, as faculty mentoring can be vital to their academic and professional success.

Using gamification, a pathway has been designed to identify and encourage student success. The pathway offers concrete steps for students to take in each of their four undergraduate years. Incentives are offered to students as they complete each step, culminating in badges. The purpose of the experience report presented here is to determine if gamification can motivate students to take steps towards academic success and career readiness. Lessons learned from this effort include the importance of early introduction of gamification to the students, reinforcement in the classroom and with faculty, and the use of experience rewards over material rewards.

Student success has been outstanding, with a majority of eligible students using the gamification pathways in the first semester. Completion of tasks in support of professional success is high, and student internships and first destination job placements reported in greater numbers and earlier than ever before. Class attendance, extracurricular engagement and advisement have also been improved, leading to academic success.

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Surging enrollments in computer science and information technology programs have necessitated the scaling-up of many programs, resulting in more curriculum offerings and larger courses enrollments to meet increasing student demand. For both research-oriented and teaching colleges and universities, this increase in enrollment has the potential to adversely impact student outcomes, as students may be unable or unwilling to interact with faculty in the larger environment. Faculty and programs which had previously offered high-touch interactions to students are now faced with maintaining the quality of their teaching and research engagement with students, and student outcomes, in a larger environment which does not allow as much student interaction as was possible earlier.

This has the potential to impact student outcomes, particularly students who are first generation college students and students who do not come from professional family backgrounds. These students, previously able to rely on small class sizes and faculty interaction as they developed their skills needed to successful navigate a college degree in computer science and information technology, as well as internships, and professional job interviews, are now larger in number in many programs, a result of the enrollment increases, but harder to reach, and effectively prepare for both engagement in undergraduate program activities and professional careers.

In response to the need to clearly define the steps to be taken for academic and professional success in computing careers, a pathway for success has been defined by one program at a teaching university. This program design incentivizes the students, using gamification techniques, to engage in a sequence of activities which will develop academic and professional skills needed for success. By listing the steps and options available to students and personalizing the engagement of the students on each individual pathway, the faculty mentoring role, once done one-on-one is now extended to a larger group of students, allowing for consistent, greater impact overall.

In addition, the interview process for both internships and first jobs after graduation has changed, with resumes, elevator speeches, video interviews and coding assessments all part of the screening process, leading to team interviews and site visits. This lengthy path is unfamiliar to many students pursing degrees in computer science and information technology. The insight available from computer science faculty members and alumni is vital for student success along this route. In order to capture the steps which students should take in order to be successful, a guided pathway has been illustrated and publicized, providing consistent, equitable information about student activities and preparation which should occur prior to graduation for both academic and professional success.

2 Prior Work

Inspired by the work of [17] and pathways identified in other, non-computer science programs, a survey of the literature highlighted that gamification has been effectively used for student engagement [4]. Gamification is defined as the use of game design in non-game contents [16], and gamification when used with teaching practices has been found to have a positive impact on student achievement and student attitudes towards academic lessons.

Much of the prior work has been on the use of gamification in specific classroom settings [2, 1, 6, 13]. The gamification of specific classes, with points, badges, and experience levels, showed improvement in student participation, pro-activity, superior performance, motivation resulting in higher grades, and overall encouragement resulting from a motivated student mindset. While faculty are looking at educational games to determine if they can increase student engagement and learning [11], it is most often in the context of a specific course, not program wide. The use of a mobile application in a classroom was found to improve student retention rates and academic performance [14]. Other web-based gamification in the classroom also fared well [7].

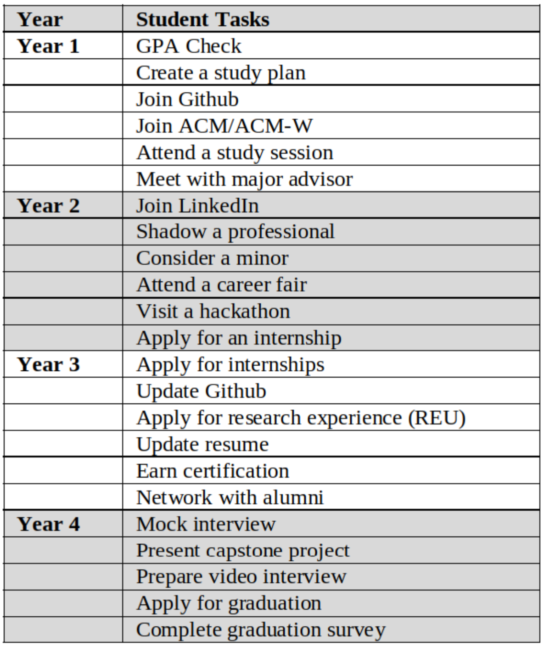
The purpose of the designed pathway was motivation, and gamification is recognized as being a strong student motivator [5]. Badges, or waypoint indicators within a game, illustrating accomplishments, have been found to increase student activity [8]. Gamification has been shown to improve awareness of CS-related careers [15] as well. A gamification approach had been taken for one aspect of new student orientation [10], which was shown to improve student engagement and completion of tasks in support of badges and mastery of material. The alignment of gamification with practice systems, such as those found in CS1 and CS2 level courses show that gamification increased student attendance, improved retention rates, and assisted students at all levels [9].

Academic work and career readiness have been explored previously, in isolation. Earlier work on gamification in a course [12] demonstrated that students did respond positively to gamification, when used to encourage development of "soft skills". Academic success, particularly attendance, has also been measured [3], and gamified awards did support improved completion of tasks and class attendance. The work presented here integrates both academic work and career readiness, outside individual classroom experience, measuring student engagement and satisfaction.

3 Methodology

The purpose of this experience report is to determine if gamification could motivate students to take steps towards academic success and career readiness. A gameboard was developed through several iterations. The result was a gameboard which depictstasks to be accomplished each year, moving students towards their goals of academic success, development of a professional portfolio, landing a paid internship and, after graduation, beginning a graduate or professional career. The pathway for student success is visually appealing, with roughly ten discreet activities students can accomplish during each academic year. Depicted students were representative of the diverse student population. Table 1 provides a sample of activities aligned by years. Figure 1 shows the pathway game board.

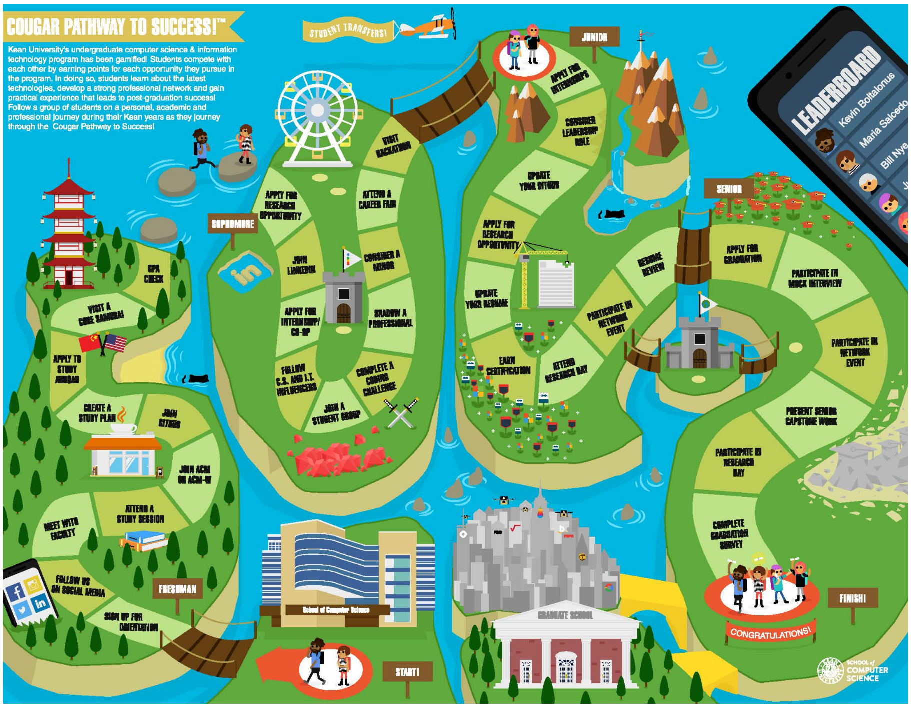
Table 1: Tasks, by year, from the pathway for student success



Year	Student Tasks
Year 1	GPA Check
	Create a study plan
	Join Github
	Join ACM/ACM-W
	Attend a study session
	Meet with major advisor
Year 2	Join LinkedIn
	Shadow a professional
	Consider a minor
	Attend a career fair
	Visit a hackathon
	Apply for an internship
Year 3	Apply for internships
	Update Github
	Apply for research experience (REU)
	Update resume
	Earn certification
	Network with alumni
Year 4	Mock interview
	Present capstone project
	Prepare video interview
	Apply for graduation
	Complete graduation survey

While these activities are basic, to many students who are attending school and unfamiliar with the effort which must be put into a professional job search, the pathway for success has been very helpful. Many students work, in addition to attending school, and professional preparation, particularly for internships, which are very important in computer science and information technology, or full-time jobs after graduation, is often deferred until graduation. However, student success and program outcomes are greater if students are prepared throughout their program, understanding why they are working on the academic courses and how they will contribute to their future professional success.

The gameboard and tasks have been reinforced with Quick Response (QR) codes, available at events, during meeting with faculty advisors, resume review


COUGAR PATHWAY TO SUCCESS!™

Kean University's undergraduate computer science & information technology program has been gamified! Students compete with each other by earning points for each opportunity they pursue in the program. In doing so, students learn about the latest technologies, develop a strong professional network and gain practical experience that leads to post-graduation success! Follow a group of students on a personal, academic and professional journey during their Kean years as they journey through the Cougar Pathway to Success!

Figure 1: Pathway to success gameboard

sessions, or any other tasks which will allow students to gain points in the pathway for success game. The tasks correspond to three badges which were developed, to provide intermediate feedback to students as they moved along towards their goal. The three badges are the Ready to Succeed badge, Road to Graduate School badge, and Academic Merit badge. Each badge can be earned by completing tasks on the pathway to success. Table 2 outlines the badges and subordinate tasks.

Additional events which occur during the semester, such as guest speakers or corporate information sessions, can be easily added as events, allowing the pathway for student success to be dynamically updated to support current activities.

Badges correspond to professional, academic and networking activities and encourage participation and engagement, as shown in Table 3.

A variety and range of activities which students can participate in is provided. Students in the department are frequently commuters, with jobs off-campus, whichencouraged the faculty and staff developing the game to include

Table 2: Badges and tasks in the pathway to success game

Badges	Subordinate Tasks
Ready to Succeed	Complete an internship Attend a career fair Develop an elevator pitch Complete a LinkedIn profile Three optional tasks
Road to Graduate School	Report on a graduate program Conduct research with a faculty member Three optional tasks.
Academic Merit	Complete a career education elective course Visit your major advisor Complete an internship Three optional tasks.

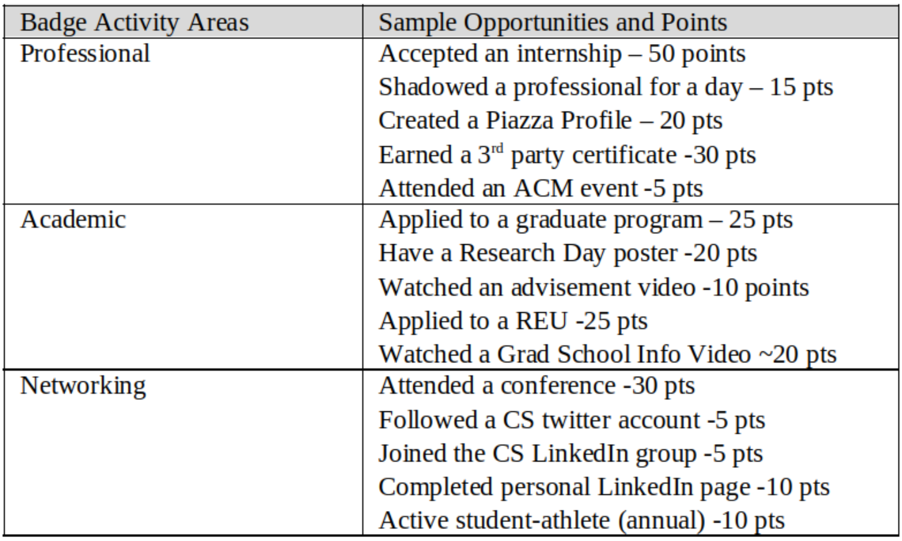
a mix of both in- person on-campus activities, as well as off-campus activities, such as videos, webinars, programming skill development activities and certifications. These activities are reported through reflections or the submission of evidence of completion or demonstration of new skills. Students are not disadvantaged by their scheduling and the importance of participation and engagement is supported by the adaptability of the activities and availability of the platform.

Once designed, the game was deployed on a mobile application, allowing students to download the app from the app store for both iOS and Android phones, and play the game anywhere. This supports the 'meeting students where they are' engagement allowing all students to participate, regardless of location and time of day.

4 Evaluation

Student success has been outstanding, with almost 50% of the eligible students registering for the pathway for student success in the first semester it was available. Registration in the following semester increased to 66%. Completion of tasks in support of professional success is high, and student internships and first destination job placements reported in greater numbers and earlier than ever before. For the first year of use, reported internships and full-time job placements doubled, with students eager to report their success and receive points. This has provided the faculty with insight into student outcomes much earlier than before. Class attendance, extracurricular engagement and advisement

Table 3: Professional/Academic/Networking Opportunities



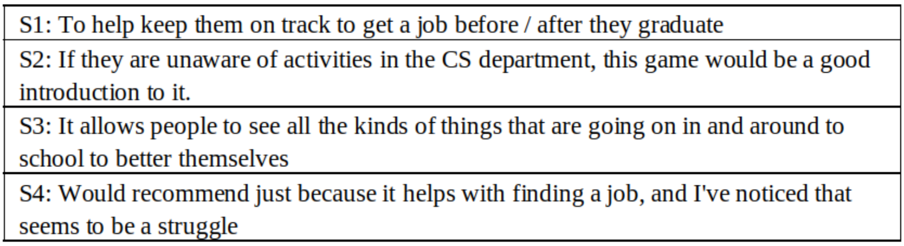
Badge Activity Areas	Sample Opportunities and Points
Professional	Accepted an internship – 50 points Shadowed a professional for a day – 15 pts Created a Piazza Profile – 20 pts Earned a 3rd party certificate -30 pts Attended an ACM event -5 pts
Academic	Applied to a graduate program – 25 pts Have a Research Day poster -20 pts Watched an advisement video -10 points Applied to a REU -25 pts Watched a Grad School Info Video ~20 pts
Networking	Attended a conference -30 pts Followed a CS twitter account -5 pts Joined the CS LinkedIn group -5 pts Completed personal LinkedIn page -10 pts Active student-athlete (annual) -10 pts

have also been improved, leading to academic success. Students understand the expectations of their program.

Students are enjoying having specific activities outlined as moving them towards their goals. Student understanding of the importance of internships and where internships fit into their overall college career is much higher than it was before. This is apparent from the most recent career fair which was well attended by juniors and seniors, as well as a few sophomores. Their engagement in the pathway for student success was high, they had prepared resumes, LinkedIn profiles and elevator pitches, and they understood the importance of an internship or full-time job. Attending the career fair was the logical next step and they were fully prepared.

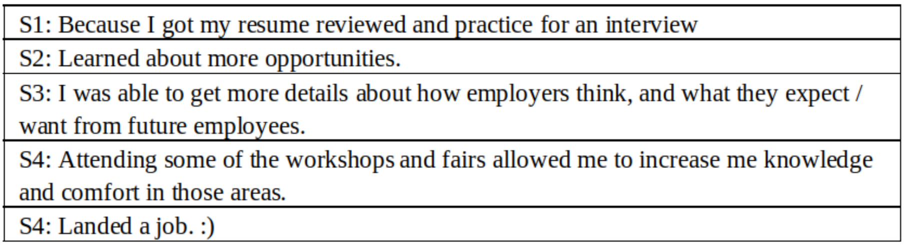
The tasks and badges map back into competency areas, with personal and professional development being the largest area of activity, followed closely by academic achievement, and networking skills. Students can see their peers that are on a 'leaderboard' and enjoy seeing who is doing well. The department, in return, can see which students are most engaged in the process, and faculty can reach out to students who may not be engaged to encourage participation. Experience awards are given to the top students by academic year. For example, early access to the career fair is given to the top students in the pathway for student success, as they have shown a strong effort in preparing them for the event. Student feedback has been positive, with students reporting that they would encourage friends to play the game and stating how they felt for interviews. Selected student responses are listed in Tables 4 and 5.

Table 4: Would you encourage your friends to play? Why? (selected responses)



S1: To help keep them on track to get a job before / after they graduate
S2: If they are unaware of activities in the CS department, this game would be a good introduction to it.
S3: It allows people to see all the kinds of things that are going on in and around to school to better themselves
S4: Would recommend just because it helps with finding a job, and I've noticed that seems to be a struggle

Table 5: Were you better prepared for interviews? Why? (selected responses)



S1: Because I got my resume reviewed and practice for an interview
S2: Learned about more opportunities.
S3: I was able to get more details about how employers think, and what they expect / want from future employees.
S4: Attending some of the workshops and fairs allowed me to increase me knowledge and comfort in those areas.
S4: Landed a job. :)

5 Conclusions and Future Work

Students have been encouraged to participate in the pathway for success through their classes. During the first week of each semester, a department representative visits freshmen, sophomore, junior and senior classes to describe the game and encourage students to sign up. With a large percentage of transfer students and many transfers into the major from other majors in the university, this regular reminder is necessary and effective. Lessons learned include:

1. Freshmen and sophomore students are most receptive to this defined process. If gamification is used on a program-wide scale, it should be introduced early to students.
2. Students should be required to sign up for the pathway for success. The sign-up, reinforced through the classroom visit, is also listed on faculty syllabi. Reinforcement through the learning management system (LMS), also is important. Many faculty placed the sign-up link in their class LMS banner, which encourages adoption. Sign-up is now included in the freshman and transfer introduction course and reinforced through the department's career education course which prepares students for internships, graduate applications, and professional jobs.

3. Faculty must be onboard with the pathway for student success as well. Reinforcement in the classroom, and faculty participation, encourages student engagement. Several faculty are also ‘playing’ the game and can be seen on the leaderboard at times.

Plans for future work include providing more rewards, such as inviting students to alumni events, where they can interact with recent graduates and learn more about the necessary professional preparation. The gamification provided by the pathway for success has greatly increased both student understanding of what is needed for academic and professional success and provided them with measurable steps and feedback as they advanced in the program. This allows increasing numbers of students to receive specific, detailed information about their progress in and out of the classroom towards their professional goals.

The goal of providing information to students in an engaging format has improved both student and faculty awareness of the steps needed to move towards professional and graduate goals in computer science and information technology fields .

References

[1] Paul E Anderson, Thomas Nash, and Renée McCauley. Facilitating programming success in data science courses through gamified scaffolding and Learn2Mine. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 99–104, 2015.

[2] Gabriel Barata, Sandra Gama, Joaquim Jorge, and Daniel Gonçalves. Improving participation and learning with gamification. In *Proceedings of the First International Conference on gameful design, research, and applications*, pages 10–17, 2013.

[3] Hope Caton and Darrel Greenhill. The effects of gamification on student attendance and team performance in a third-year undergraduate game production module. In *European conference on games based learning*, page 88. Academic Conferences International Limited, 2013.

[4] Luma da Rocha Seixas, Alex Sandro Gomes, and Ivanildo José de Melo Filho. Effectiveness of gamification in the engagement of students. *Computers in Human Behavior*, 58:48–63, 2016.

[5] Sebastian Deterding. Gamification: designing for motivation. *interactions*, 19(4):14–17, 2012.

[6] Kiko Fernandez-Reyes, Dave Clarke, and Janina Hornbach. The impact of opt-in gamification on students’ grades in a software design course. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 90–97, 2018.

[7] Panagiotis Fotaris, Theodoros Mastoras, Richard Leinfellner, and Yasmine Rosunally. Climbing up the leaderboard: An empirical study of applying gamification techniques to a computer programming class. *Electronic Journal of e-learning*, 14(2):94–110, 2016.

[8] Juho Hamari. Do badges increase user activity? a field experiment on the effects of gamification. *Computers in human behavior*, 71:469–478, 2017.

[9] Brian Harrington and Ayaan Chaudhry. TrAcademic: improving participation and engagement in CS1/CS2 with gamified practicals. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, pages 347–352, 2017.

[10] Andrew Januszak and Cristina Koorie. Designing and deploying a virtual IT services orientation for first-year undergraduate students in Moodle. In *Proceedings of the 2018 ACM SIGUCCS Annual Conference*, pages 87–89, 2018.

[11] Trisha Litz and Kevin Pyatt. Using an educational game to increase student engagement & learning. *Journal of Computing Sciences in Colleges*, 33(2):129–133, 2017.

[12] Bruce R Maxim, Stein Brunvand, and Adrienne Decker. Use of role-play and gamification in a software project course. In *2017 IEEE frontiers in education conference (FIE)*, pages 1–5. IEEE, 2017.

[13] Olena Pastushenko, Tomáš Hruška, and Jaroslav Zendulka. Increasing students' motivation by using virtual learning environments based on gamification mechanics: Implementation and evaluation of gamified assignments for students. In *Proceedings of the Sixth International Conference on Technological Ecosystems for Enhancing Multiculturality*, pages 755–760, 2018.

[14] Ekaterina Pechenkina, Daniel Laurence, Grainne Oates, Daniel Eldridge, and Dan Hunter. Using a gamified mobile app to increase student engagement, retention and academic achievement. *International Journal of Educational Technology in Higher Education*, 14(1):31, 2017.

[15] Brenda Scholtz, Larissa Raga, and Gavin Baxter. Design and evaluation of a "gamified" system for improving career knowledge in computing sciences. *The African Journal of Information and Communication*, 2016(18):7–32, 2016.

[16] Ibrahim Yildirim. The effects of gamification-based teaching practices on student achievement and students' attitudes toward lessons. *The Internet and Higher Education*, 33:86–92, 2017.

[17] Dustin York. Non-verbal immediacy's role in student learning. *Journal of media and Communication Studies*, 7(1):1–7, 2015.

Automatic Programming Assignment Assessment beyond Black-box Testing[*]

Karen H. Jin
Department of Applied Engineering and Sciences
University of New Hampshire
Manchester, NH 03101
Karen.Jin@unh.edu
Michel Charpentier
Department of Computer Science
University of New Hampshire
Durham, NH 03824
Michel.Charpentier@cs.unh.edu

Abstract

Automatic programming assignment assessment is often premised on black-box testing. Grading of student submissions typically relies on functional specifications expressed in terms of expected outputs for given test inputs. However, when students are required to use certain programming language constructs, algorithms, or design strategies as they implement their programs, a different approach to automated assessment is needed. Our strategy is centered on programming assignments designed in such a way that the internals of the assignment implementation can be evaluated through automated testing. Compared to plain functional specifications and black-box tests, auto-graded "cutout-box" assignments require that both the specifications and the grading tests have a much higher level of complexity. In this paper, we discuss the benefits and difficulties of using such an approach through case studies in designing data structure assignments with a functional programming language.

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Courses that rely heavily on programming assignments for pedagogical and grading purposes are common in computing curricula. Many of these programming courses are dependent on automated black-box testing of assignments for grading and evaluation. The programs to implement are thus specified functionally in terms of expected outputs for given inputs. The benefits, drawbacks, and limitations of automated evaluation based on black-box testing have been documented [1, 4, 8]. It is usually agreed that this strategy tends to work best in introductory courses, in which students have limited freedom on how to implement programs and where a correct output is indicative of a suitable implementation. Black-box assignments can also be used in some upper-level courses [3, 6]. A course on algorithms, for instance, can often prepare large enough inputs that sub-optimal implementations cannot solve in a reasonable time—it is relatively easy to generate *knapsack* instances that can be solved using dynamic programming but on which a naive recursive solution would take an inordinate amount of time. Tools are being developed that can produce estimates of the complexity of an implementation, using black-box testing [2].

Many courses, however, are centered on concepts that relate to how programs are implemented. In a course that teaches functional programming, for instance, the assignments should require that students use functional programming techniques, even if imperative solutions are supported by the language. Similarly, a data structure assignment may require students to implement a tree search that follows optimized paths rather than simply producing the target. In these cases, black-box assignments are inadequate. They are not well-suited to evaluate the proper use of recursion or the application of lazy evaluation to reduce CPU and memory usage. Solutions that rely on iteration (instead of recursion) or eager evaluation (instead of lazy evaluation) could possibly satisfy all functional requirements but be unacceptable for the purpose of a course.

An alternative to running tests for evaluation and grading is to rely instead on static analysis tools that operate at the source code level [5]. Such tools have been very successful at checking the coding style of beginning programmers or at looking for so-called "code smells" [7]. There are, for instance, excellent tools that can help enforce good coding practice (formatting, naming, size, cyclomatic complexity, etc.) in Java and other languages. Static analysis can be used beyond style checking but remains limited when dealing with upper-level courses. Moreover, these tools tend to suffer from the presence of *false positives.* While false positives do not prevent a static checking tool from being useful in guiding students, it usually precludes using it for grading purposes.

1.1 Auto-grading with Cutout-box Assignment Design

Our proposed approach to improve the automated assessment of programming assignments is based on a hybrid form of black-box and white-box testing. Our strategy is centered on the idea of designing programming assignments in such a way that their internals (the *how* of their implementation) can be evaluated through automated testing. We refer to such assignments as *cutout-box assignments* (Fig. 1). Their specification needs to expose more implementation details than is strictly necessary for a functional specification, in order to enable tests to evaluate assignments beyond their performance and functional behavior.

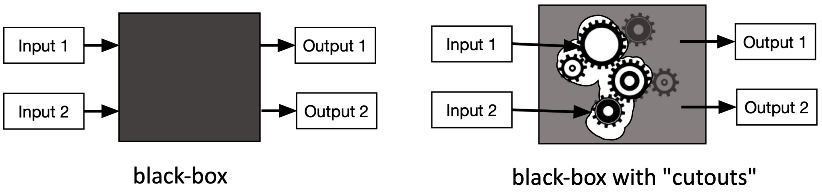


Figure 1: Testing of black-box vs cutout-box programming assignments.

The primary motivation for using cutout-box assignments is to be able to require that students rely on certain programming language constructs, algorithms or design strategies as they implement their programs. In many courses, the goal of programming assignments is for students to gain practice and experience with the specific concepts covered in the course; it is not to have them create a program that accomplishes task X by any means necessary. Meanwhile, implementation requirements that are too specific might stifle student creativity. Thus, it is crucial that the specifications and grading tests of cutout-box assignments are designed hand in hand.

Grading tests should also be able to detect forbidden algorithms or language features. Even if a specification is clear about what *not* to use in an implementation, it can be hard to design tests that will check that students obeyed such constraints. Most importantly, because the assignment specification will need to include some level of implementation details, it is critical not to rely on unspecified behavior in the design of grading tests. That is, a test should not inadvertently rely on implementation details that were *not* specified. In this paper, we illustrate the benefits of our strategy with two case studies in which our strategy is applied to design programming assignments for teaching functional programming in a course using the Scala programming language.

2 Case Study - List Recursion

2.1 Initial Assignment Design

This assignment aims to teach the use of recursion on the standard *head/tail* functional list. Students were asked to implement a list-based representation of a set of numbers and to practice recursive programming by implementing standard set operations. Sets of integers are represented as lists of ranges, e.g., the set $\{3, 4, 5, 8\}$ is represented as the list $([3, 5], [8, 8])$. This representation is potentially more compact than expanded lists when sets are dense. Implementing typical set operations on such lists makes a good exercise in recursive programming. For instance, a `size` function can be implemented in Scala using tail recursion:

```
def size(ranges: List[(Int, Int)]): Int = {
  @tailrec
  def calcSize(ranges: List[(Int, Int)], size: Int): Int =
    ranges match {
      case Nil           => size
      case (a, b) :: r => calcSize(r, size + b - a + 1)
    }
  calcSize(ranges, 0)
}
```

Unfortunately, our initial design was not careful enough to prevent students from implementing it with unacceptable strategies. The original assignment included `toList` and `fromList` functions (good exercises on recursion in themselves), to convert sets to and from fully expanded lists. These functions were only intended as a bridge between two models of the sets, but once implemented, students could use them as the basis of all the other methods, e.g.:

```
def size(ranges: List[(Int, Int)]): Int = toList(ranges).length
def contains(ranges: List[(Int, Int)], x: Int): Boolean =
  toList(ranges).contains(x)
def remove(ranges: List[(Int, Int)], x: Int): List[(Int, Int)] =
  fromList(toList(ranges).filterNot(_ == x))
```

Such an implementation defeats the purpose of the assignment, which is to practice recursion on lists. The only way for a black-box test to catch the problem would be to use extremely large dense sets that cannot be fully expanded in memory. This tends to be impractical for reasons of timing and fragility.

2.2 Improved Assignment Design

A revised assignment uses instead a notion of *marked lists* and exposes the constructors of such a list. Marked lists are very much like functional lists, with a head and a tail, but the head can optionally be *marked*, allowing some elements of the list to be singled out. A type `MarkedList` is defined using three constructors, `Empty`, `Cons` and `MCons`:

```
case object Empty ...
case class Cons[A](head: A, tail: MarkedList[A]) ...
case class MCons[A](head: A, tail: MarkedList[A]) ...
```

For instance, `Cons(1, Cons(2, Cons(3, Empty)))` is the list $(1, 2, 3)$ while `Cons(1, MCons(2, Cons(3, Empty)))` is the list $(1, \underline{2}, 3)$, where element "2" is marked. Recursive code on marked lists is very similar to code on regular lists, the only difference being two forms of "cons" instead of the `::` method of Scala.

The assignment still includes a `toList` and a `fromList` functions, but `toList` loses all marks and `fromList` produces unmarked lists. In particular, `fromList(toList(m))` is *not* the same marked list as `m`, which makes it impossible to implement marked list functions directly in terms of corresponding functions on regular lists. Instead, students end up writing recursive code directly on the three constructors. For instance, a function that counts the number of marked elements can be written using tail-recursion:

```
def countMarks: Int = {
  @tailrec
  def count(m: MarkedList[A], c: Int): Int = m match {
    case Empty        => c
    case Cons(_, t)   => count(t, c)
    case MCons(_, t)  => count(t, c + 1)
  }
  count(this, 0)
}
```

This function is almost identical to a `length` function, but it cannot be implemented by converting the marked list into a regular list. As an added benefit, other mark-specific functions can be used instead of the more complex higher-order functions on lists, especially in an early assignment. For instance, a function `firstMark` can look for the first marked element, if any:

```
@tailrec
def firstMark[A](m: MarkedList[A]): Option[A] = m match {
  case Empty        => None
  case Cons(_, t)   => firstMark(t)
  case MCons(h, _)  => Some(h)
}
```

This function is as good a practice on recursion as the standard higher-order *find* function, but it is conceptually simpler.

By using a custom-made marked list structure that exposes constructors similar to that of a regular list, the revised assignment allows for automated testing of correct internal implementation, i.e., the proper use of recursion.

3 Case Study - Quadtree

3.1 Design Challenge

A quadtree is a hierarchical data structure that splits a two-dimensional space into four quadrants (top-left, top-right, bottom-left, and bottom-right) and recursively splits each subspace in the same manner. A classic implementation of a quadtree consists of internal nodes with exactly four children representing four quadrants of the space, and leaf nodes that contain the actual data, i.e., a collection of points in the space. Such a tree uses a notion of *threshold*: Leaf nodes have at most the threshold number of elements and internal nodes are only used for subtrees with more than the threshold. Trees that satisfy this property are said to be in *canonical form*.

It is impossible for a purely functional, black-box test to check that a tree is in canonical form, or, for all practical purposes, to check that a tree operation is implemented with the correct algorithm. For example, searching for the nearest point to a target in a quadtree can be sped up by relying on the tree's spatial properties. That is, certain quadrants—and thus the subtrees that represent them— can be ignored during a search because they cannot possibly contain a point closer to the target than a point already found. Automated testing needs to check that the search has correctly traversed the internal nodes by not visiting any unnecessary nodes. An implementation that systematically searches all four quadrants of each node is considered incorrect even though it produces the correct target. (This would be like visiting both children of a node in a binary search tree.) Unfortunately, such an error cannot be detected by traditional black-box testing. It is also infeasible to use a white-box testing approach that would build explicit search paths (e.g., by requiring print statements to be inserted in the search code) because such paths are not fully specified in complex search operations and because this approach would preclude testing large trees with thousand of nodes.

3.2 Verify Canonical Form with Exposed Constructors

Our *quadtree* assignment is designed with the following constructors:

```
class QuadTree[K <: Location, A] (...) {...}
case class QuadNode[K <: Location, A](...) {...}
case class QuadLeaf[K <: Location, A](...) {...}
```

The class is parameterized by two types: A subtype `K` of `Location`, which represents 2D coordinates and is used as keys, and a type A, which represents the tree's content. All classes are left public so tests can check the internal structure of the tree. We represent the expected structure of quadtrees as objects of type `TreeStruct`, which can be given to students as part of a test. Simple code is used to check that quadtrees generated by students match the expected structure:

```
// a (content-free) representation of the structure of a quadtree
trait TreeStruct ...
// a function to extract the structure of a quadtree
def q2Struct(q: QuadTree[_, _]): TreeStruct ...
```

Exposing the quadtree constructors makes it possible to check that a tree is in canonical form without including its exact value as part of the test, e.g., `q2Struct(q) == expectedStruct` where `expectedStruct` represents the expected structure of quadtree `q`.

3.3 Test Search Path with Customized Keys

Customized key implementations can be used to ensure that a search for the nearest neighboring node in a quadtree only visits a subset of all nodes in the tree. In our tests, a `Point` class is defined that keeps track of how many times its coordinates are queried. A method `untouched` is written to check that points have not been accessed during a search in a given quadtree:

```
test("getNearest"){
   val pt = quad.getNearest(Point(5, 4))
   assert(pt === C)
   assert(untouched(A, B, D, E, F))
}
```

The first assertion (`pt == C`) is the functional specification that the search produced the correct output. The second assertion checks that a set of points—belonging to quadrants that should not be explored—were not queried during the search.

Note that there exist multiple valid search paths for the `getNearest` method, and students are not required to produce a specific search path as long as their search follows the optimized algorithm of a quadtree. With our approach, a large number of automated tests can be written for various search scenarios—all points within a distance, neighboring points that satisfy a filter, etc.—while still giving students the freedom to implement the details of the searching methods.

4 Lessons Learned

Over the years, we have learned several valuable lessons in the art of designing good assignments specifications.

Firstly, good software engineering practices like modularity are essential. Most assignments see a profusion of interfaces (Java) and traits (Scala) in addition to code-containing modules like classes and functors. These make it easier to write tests by introducing specialized variants of modules (so-called *mock* objects). There have been many cases of grading tests that were impossible to write until an assignment was re-engineered to introduce intermediate types through interfaces and signatures. Hooks can also be used to measure intended side-effects (e.g., count how many times a method is entered).

Secondly, writing extensive collections of grading tests makes it possible to provide students with a set of sample tests as a way to clarify or emphasize certain aspects of a specification. No matter how careful we are, specifications tend to be ambiguous, as cutout-box assignments need to remain somewhat open-ended when specifying a desired implementation without giving it away. Sample tests can help alleviate this difficulty, but need to be designed carefully. Good sample tests can help emphasize subtle aspects of a specification, steer students clear from common mistakes and give them some idea of what performance is expected of their code. Bad sample tests are those that require an implementation to exhibit specific behaviors not specified as part of the assignment. Here also, we have learned from past mistakes. For instance, a Java assignment was designed using generic types, but all the sample tests used strings for convenience. Actual grading code used numbers instead of strings in order to generate large test instances more easily, but some of the code submitted by students relied on string typecasts and could not pass any of the grading tests. In another case, sample tests used only small numbers. Incorrect Java code that compared `Integer` instances using == could successfully pass these tests because `Integer` instances under 127 are usually implemented as singletons in Java.

Thirdly, good cutout-box assignments are often the result of multiple iterations. Ambiguities, inconsistencies and other mistakes can be detected one semester and resolved before the assignment is reused. Our first case study showed how a flawed assignment was reworked into a better one over time. Since it is such a demanding task to design cutout-box programming assignments with an adequate structure, a precise specification, helpful sample tests and accurate grading tests, courses offered on a regular basis can only introduce a few new assignments each time. Most assignments have to be reused multiple times. Furthermore, variations of an existing assignment are hard to produce. The constraint of writing specifications that expose internal aspects of programs tends to make cutout-box assignments fragile. It is often difficult to modify specifications, even in a minor way, without introducing inconsistencies or resulting in semantics at odds with a large set of existing tests.

Finally, after years of writing in-house tools, we came to the conclusion that it is preferable to rely on standard software development packages. Using them provides students with an opportunity to be (re-)exposed to some concepts, like version control or unit testing, that they will later use professionally. Standard unit-testing tools like JUnit, TestNG, ScalaTest, etc., have improved over the years and now support features that used to be lacking. Most of these tools can be turned into grading systems by using customized observers and reporters that keep track of passed and failed tests. Nevertheless, these tools were not designed for teaching purposes and have their own limitations, and overall support for testing more advanced features such as multi-threaded programs remains limited.

5 Conclusion

The most frequent form of grading based on tests follows a black-box approach, in which student programs are evaluated in terms of their functional behavior. However, many advanced courses are centered on concepts that relate to *how* programs are implemented, e.g., what algorithms are followed, what programming language constructs are used. The black-box testing approach becomes insufficient for these courses. The alternative that we have been using in our teaching relies on a strategy that allows precise and thorough automatic grading of complex assignments. The strategy is based on specifying and evaluating the details of a program implementation instead of solely on performance and observable functional behavior. Exposed structure and hook methods are frequent patterns in designing assignments that allow for automated testing of their internal structure. The case studies of this article illustrate this approach can be applied in designing data structure and functional programming assignments. We believe that the design of such assignments is an important topic, and we feel that there is a need for discussion, collaboration, code sharing and common tool building, which we hope to foster with this work.

References

[1] Kirsti M. Ala-Mutka. Automatic test-based assessment of programming: A review. *J. Computer Science Education*, 15(2):83–102, Jun 2005.

[2] Clara Benac Earle, Lars-Ake Fredlund, and John Hughes. Automatic grading of programming exercises using property-based testing. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, pages 47–52, New York, NY, USA, 2016. ACM.

[3] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121–131, 2003.

[4] Christopher Douce, David Livingstone, and James Orwell. Automatic test-based assessment of programming: A review. *J. Educ. Resour. Comput.*, 5(3), September 2005.

[5] Stephen H. Edwards, Nischel Kandru, and Mukund B.M. Rajagopal. Investigating static analysis errors in student java programs. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*, ICER '17, pages 65–73, New York, NY, USA, 2017. ACM.

[6] E. Enström, G. Kreitz, F. Niemelä, P. Söderman, and V. Kann. Five years with kattis - using an automated assessment system in teaching. In *2011 Frontiers in Education Conference (FIE)*, pages T3J–1–T3J–6, Oct 2011.

[7] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, pages 110–115, New York, NY, USA, 2017. ACM.

[8] Vreda Pieterse. Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, CSERC '13, pages 4:45–4:56, Open Univ., Heerlen, The Netherlands, The Netherlands, 2013. Open Universiteit, Heerlen.

Oh the Robots that You can Choose: A Technical Review of Mobile Robot Platforms*

Benjamin T. Fine[1] *and Jory Denny*[2] *and*
Nate Dix[2] *and Ashley Frazier*[2]
[1]*School of Theoretical and Applied Sciences*
Ramapo College of New Jersey
Mahwah, NJ 07480
bfine@ramapo.edu
[2]*Department of Mathematics and Computer Science*
University of Richmond
Richmond, VA 23173
jdenny@richmond.edu

Abstract

In recent years, the number of available robotic platforms on the market has not only increased, but also the choices in terms of sensors, actuation, and language compatibility have diversified. While there is a large body of literature that discusses the integration and use of robots in Computer Science, many of the works do not adequately treat the selection of the robot platform. In this study, we review the technical aspects of 17 mobile robot platforms for integration into a Computer Science curriculum. This work presents a discussion on how one should evaluate and compare various mobile robot platforms with respect to how the robot will be integrated and utilized. Additionally, new metrics for cross comparing the available robots on the market are presented and used in exemplar case studies.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Educational robotics is a field that has been around since the early 1980's but is still rapidly evolving and expanding. There are many studies that show the pedagogical benefits of using robots in the classroom, including experiential learning [3, 13, 19], student engagement [9, 6], and improving transfer skills [1]. The field of educational robotics explores questions of diversity and broadening participation [1, 14, 11, 10], effectiveness of robotics in education [7, 2, 8], creativity and student engagement [9, 6, 16], and professional development [15]; ranging from K-12 to graduate institutions. While many investigations have been conducted in this field related to the pedagogy, few studies have focused on the the robot platform chosen. With the advancements of technology and educational robotics, an increasing number of educators are introducing robots into their courses [20, 4, 17, 5]; however, many of these still neglect the selection of the robot.

Choosing an appropriate platform for use in a given Computer Science curriculum should take in to account a number of factors, including, but not limited to (1) the cost, (2) variety of support languages, (3) assignment versatility, (4) ease-of-use for both the faculty and students, (5) and if the robot platform will be used in a single course or across the curriculum. There are only a few published reviews that cover the various robotic platforms available on the market [12, 18]. These reviews cover a larger variety of robot styles, such as articulated arms and aerial vehicles, where the review presented here will narrow the focus to wheeled[1] robots. Another key difference between prior studies and the work presented here is that a series of metrics are presented as a tool for faculty outside of the discipline of Robotics to use in the selection of a platform.

In both of these studies the authors present short prose based descriptions of the capabilities and sensor profile of each of the selected platforms. Each of the robots were primarily evaluated on four factors: (1) modularity, (2) reusability, (3) versatility, and (4) affordability. While these factors are captured in the work presented here, our review introduces new metrics which limit subjectivity when evaluating a platform that we see in the previous works. In this work, we will present

- a technical summary of 17 mobile robot platforms for use in a Computer Science curriculum[2],

[1]For this study, "wheeled" includes both wheeled and tracked actuation mechanisms. We did not consider legged robots as the actuation mechanics are much different than wheeled and tracked robots. Future work should include these platforms.

[2]We do not cite the robot platforms due to space limitations, but we note that they are easily located with an internet search.

- a series of new metrics that compare these 17 platforms against the other possible platforms,
- and recommendations for how to choose a mobile robot platform in comparison to other candidate platforms.

In addition to the above contributions, this study will present case studies for how one would select a robot platform for integration into a Computer Science curriculum. It is also important to note that the authors have hands-on classroom/laboratory experiences with 10 of the 17 robot platforms reviewed in this study, and we have integrated a number of these platforms into two curricula at two different institutions. From these experiences, a number of side considerations will be discussed throughout this work.

2 Robot Platform Review

2.1 Robot Selection

Over the past few decades the variety and style of robot platforms have steadily increased. The variety of platforms range from articulated arms, aerial vehicles, ground vehicles, marine vehicles, humanoids, and simulated robots. While each of these platform types have a place in a Computer Science curriculum, this study is going to only consider wheeled robots. We have chosen this classification of platform as it has the lowest entry cost both monetarily and in integrating the platform into a given course or curriculum for educators not directly in the field of Robotics.

This review consists of 17 wheeled robots that are available on the market at the writing of this work. These robots range from platforms specifically designed for the classroom to robots that are more geared towards higher level research. We selected robots that have been used in a classroom setting in at least one known instance. We do not include usage counts in our review as this number could be skewed in a number of ways.

2.2 Metrics

The review of each platform is presented in Table 1, which is divided into three sections: (1) the metrics for cross platform comparisons, (2) the supported languages, and (3) the equipped sensors. Below are the definitions of the metrics used in this study. We only explicitly cover Cost Ratio, Sensor Variety Count, Sensor Diversity, Language Variety Count, Language Diversity, Computation, IDE Restricted, and Expandability as the other metrics are standard in the other platform reviews and are self-explanatory. Moreover, all of these metrics are used in the metrics section of Table 1, which is the primary section we will use for selecting appropriate platforms for a Computer Science course or curriculum.

Cost Ratio This metric is the cost of the robot over the average textbook cost ($153.00 USD [21]) students spend per course nationwide. This ratio is useful in determining if it would be reasonable to have the students purchase the robot platform or as justification for internal funding opportunities. For example, if the cost ratio is 2.00 and the robot will be used in two separate courses (assuming no textbooks are required), then the cost to the student stays in line with national averages.

Sensor Variety Count and Diversity The sensor variety count metric is the number of different types of sensors that the robot has equipped. Sensor diversity is the sensor variety count over the average variety count for all 17 robots in this study. The sensor diversity tells us how diverse the sensor layout is in respect to the other options on the market.

Language Variety Count and Diversity The language variety count metric is the number of different languages natively supported by the platform. We did not count languages that required firmware updates or extensive serial port communication programming. Language diversity is the language count over the average language count. Again, the diversity metric allows for comparisons across available platforms.

Computation This metric refers to the level and style of computation supported by the robot. The three classifications used are On Board (OB), Micro Processor (MP), and Tethered (T). On board computation means that the robot is controlled by a full computer (*e.g.*, Raspberry Pi or laptop), Micro Processor means the robot is controlled by a micro processor such as an Arduino, and Tethered means that the robot must be connected to another device in order to operate.

IDE Restricted This metric is "Yes" if you must use a platform specific interface to communicate, upload code, or execute programs. Here, the restriction is not on the software libraries associated with the robot, but on the programming interface. For example, in order to upload the code to the Sparki platform, you must go through the SparkiDuino interface, thus it is considered to be IDE Restricted.

Expandability This metric encapsulates the flexibility and modularity of a given platform. To avoid subjectivity in this study, we only consider this metric to be binary. A value of "No" signifies that there is no reasonable way to modify or extend the robot. An exemplar of this would be the

Elisa-3 platform. The sensors and actuators are permanently attached to the processing chip.

It is important to note that the data in the sensor section of Table 1 is open to interpretation. With many of these platforms, there are a number of configurations within the base kits and between available expansions. For this work we tried to *n*ormalize the kits across the 17 platforms. For example, for the GoPiGo3 table entry, this includes the basic kit and the sensor expansion pack.

Also, we note in regards to the language section of Table 1 that all block based languages such as Scratch, mBlock, and Snap are simply considered "Block Based". Additionally, we have chosen to merge NodeJS with JavaScript and to exclude entries for the VEXCode and PBASIC languages; as these languages are rarely used in a college setting. You will see that the entry for the Boe-Bot and Sumo-Bot platforms are missing a language since they are only programmed with PBASIC.

3 Discussion

With the large variety of available robots on the market, it can be difficult to select an appropriate robot for a course or for an entire curriculum. Sometimes robots are chosen because another educator from a different institution has recommended the platform, the available language support, or the price point of the platform. While these approaches are common, this work aims to present a more systematic approach to the selection of a platform.

There are four key aspects one must consider when selecting a platform: (1) the computational style of the platform, (2) the sensor profile, (3) the supported languages, and (4) the cost. These aspects should be the driving force behind any platform selection. We present two case studies below that consider these aspects with the metrics presented earlier in this work.

3.1 Case Study: Selecting for a Curriculum

When considering a robot for use in multiple courses in a curriculum, the computation style should be a complete computer (On Board), as to have the computational power for a high level course such as Data Structures, Robotics, or AI. As it is difficult to predict the various assignments that the students will have throughout the curriculum, both the language diversity and the sensor diversity should be high. This will allow for the most flexibility through the entire curriculum. The cost of the platform is less of a factor in this case as it is easier to justify (for internal funding), as the robot will be used across multiple

	Metrics								Language						Sensor Type									
Robot	Cost Ratio	Sensor Variety Count	Sensor Diversity	Language Count	Language Diversity	Computation	IDE Restricted	Expandability	C/C++	Python	Java	Block Based	Matlab	JavaScript	LiDAR/Sonar	Audio	Image	IR Proximity/Downward IR	Light	Color	Tactile	Accelerometer/Gyroscope	Magnetometer	Temperature
AlphaBot	0.69	4	0.94	2	0.87	OB	No	Yes	X	X					1	0	1	2/4	0	0	0	0	0	0
Boe-Bot	1.43	4	0.94	1	0.44	MP	Yes	Yes							1	0	0	2/0	2	0	2	0	0	0
Create	1.31	3	0.71	4	1.74	OB	No	Yes	X	X		X	X		0	0	0	1/4	0	0	2	0	0	0
Elisa-3	2.55	3	0.71	1	0.44	MP	No	No	X						0	0	0	8/4	0	0	0	1/0	0	0
E-Puck	6.67	5	1.18	3	1.31	MP	No	Yes	X	X		X			1	4	1	8/4	0	0	0	1/0	0	0
Finch 1.0	0.65	5	1.18	3	1.31	T	No	No		X	X	X			0	0	0	2/2	2	0	0	1/0	0	1
GoPiGo3	1.90	6	1.42	3	1.31	OB	No	Yes		X		X		X	1	1	1	0/2	0	0	0	0	1	1
Hemisson	1.80	5	1.18	2	0.87	MP	No	Yes	X				X		1	0	1	6/2	8	0	0	0	0	0
Khepera	20.78	7	1.65	2	0.87	OB	No	Yes	X	X					5	1	1	8/4	0	0	0	1/1	0	0
mBot	0.33	3	0.71	1	0.44	MP	Yes	Yes				X			1	0	0	0/2	1	0	0	0	0	0
Mindstorm	2.29	5	1.18	2	0.87	MP	Yes	Yes	X			X			1	0	0	1/0	0	1	2	1/0	0	0
Sparki	0.98	5	1.18	2	0.87	MP	Yes	No	X			X			1	0	0	0/5	3	0	0	1/0	1	0
Sphero Bolt	0.98	5	1.18	2	0.87	T	Yes	No				X		X	0	0	0	4/0	1	0	0	1/1	1	0
Sumo-Bot	1.05	2	0.47	1	0.44	MP	Yes	No							0	0	0	2/2	0	0	0	0	0	0
Turtlebot3 Burger	3.59	4	0.94	4	1.74	OB	No	Yes	X	X	X	X			1	0	0	0	0	0	0	1/1	1	0
Turtlebot3 Waffle	9.15	5	1.18	4	1.74	OB	No	Yes	X	X	X	X			1	0	1	0	0	0	0	1/1	1	0
VEX EDR	4.25	1	0.24	2	0.87	MP	Yes	Yes				X			0	0	0	0	0	0	2	0	0	0

Table 1: This Table is broken down into three main sections; (1) the metrics used for comparing robot platforms, (2) the languages natively supported by the platforms, and (3) the sensors that are equipped on the selected robots.

courses. However, if the students are required to purchase the platform this changes the consideration.

On one hand, the more courses the students takes with the robot, the more justified the cost is, but the upfront cost of the robot may be too high for some students. One consideration that is institution specific, is to work with an on-campus book store to purchase a set of robots. This has two benefits, (1) the bookstore can gain bulk discount prices and (2) the students can use financial aid to purchase the robot. This option also has the added benefit that the institution does not need to manage or maintain a large collection of robots over the years and the robots for the curriculum can be updated with little resource impacts.

A useful exercise is to determine what an ideal robot would be given the presented metrics. For this case study, we would consider robots that have On Board computation and diversity scores above 1.0 (*i.e.*, above average in terms of diversity). Therefore, using our notion of ideal and the results presented in Table 1, we would consider platforms similar to the E-Puck or the Turtlebot3 platforms due to their high diversity and expandability.

Although the Finch robot has high sensor and language diversity, its computation is tethered and its expandability is static; for this reason, it would not be considered. However, one may argue that the Finch could be appropriate for multiple lower-level courses such as CS0 and CS1 courses. Another reason why the Finch may not be a reasonable choice for a curriculum is due to the complexity and robustness of the equipped sensors.

There is a fundamental difference in information gathered from RGB-D cameras, 360° LIDAR, distance sonar, and IR proximity sensors with respect to robotic algorithms. That is to say, some common algorithms for sensing an environment (*e.g.*, SLAM) cannot be performed with all types of sensory information. For more advanced courses such as AI or Robotics, this would be a critical aspect of the selection. Regardless, the Finch may still be a plausible selection if the robot is mainly used in lower-level courses where the specific type of sensory data is not as vital. This information-centric view of sensory data is not addressed in this review and is left for future work.

3.2 Case Study: Selecting for an Individual Course

When choosing a platform for a single course, each of the aspects have a different focus and priority compared with selecting for an entire curriculum. A good place to start, again, is with determining the ideal robot for a particular course. In this case, language diversity is not as important as most courses revolve around a single language in any given semester. However, the metrics of cost, sensor diversity, and computation are probably of high importance.

Let us consider the selection of a robot for a CS1 course in a C++ heavy curriculum. The platform that might warrant consideration would be the Sparki. This platform is under the average cost that students spend on a given course and it has good value in terms of sensor diversity; while having native support for C/C++. The other robot one may consider would be the GoPiGo3. While this robot is almost twice the cost of what a student may spend on a course, it offers a number of benefits in terms of diversity and expandability.

4 Conclusion

This work presented a review of wheeled robots for integration and utilization in a Computer Science curriculum. The review of the robots was centered around the technical specifications of the robot and how they related to Computer Science pedagogy. From the review of the platforms, new metrics were introduced to review the platforms and allow for cross comparisons. Lastly, this work presented two case studies that walked through the selection of a robot platform using the review and metrics presented.

Future work should apply this technical review approach presented here to the other robot platform types (*e*.g., aerial, simulated, and articulated arms). More investigations should study whether or not different types of robots serve different pedagogical needs or more effectively addresses student engagement and retention. With the rapid growth of this field, we must be mindful that our studies should be cognisant that scholars and educators outside of the field of Robotics will be utilizing our studies.

Another aspect of a platform not covered here is the reliability and robustness of the robot; take for example the Sparki and Finch platforms. The Sparki platform is equipped with stepper motors where the Finch is not; thus the motions of the Sparki robot are more precise and reliable. This may not be a limitation in CS0 or CS1 courses but poses a strong limitation in upper-level courses. This was not included due to the potential for subjectivity. Future work will explore better methods for objectively quantifying this aspect.

Although not directly related to the capabilities of the platform, consideration should be given to the support structure and documentation of the robot. For example, the AlphaBot is mainly supported in the Chinese language and we found difficulties in using the current documentation with students that do not know the language. Similarly, if there are issues with the hardware or the supporting libraries, it would be important to know if there are developers working on known issues, or if the educator would be responsible for maintaining the library. Lastly, future works should strive to frame Educational Robotics work from a Computer Science perspective whenever appropriate.

Acknowledgements

We would like to thank the lab members of the Ramapo Robotics Lab at Ramapo College of New Jersey and the members of the SPIder RObotics Lab (SPIROL) at the University of Richmond for their efforts and input in reviewing these robot platforms.

References

[1] Saira Anwar, Nicholas Alexander Bascou, Muhsin Menekse, and Asefeh Kardgar. A systematic review of studies on educational robotics. *Journal of Pre-College Engineering Education Research (J-PEER)*, 9(2):2, 2019.

[2] Tucker Balch, Jay Summet, Doug Blank, Deepak Kumar, Mark Guzdial, Keith O'hara, Daniel Walker, Monica Sweat, Gaurav Gupta, Stewart Tansley, et al. Designing personal robots for education: Hardware, software, and curriculum. *IEEE Pervasive Computing*, 7(2):5–9, 2008.

[3] Bradley S Barker and John Ansorge. Robotics as means to increase achievement scores in an informal learning environment. *Journal of research on technology in education*, 39(3):229–243, 2007.

[4] Amy Delman, Lawrence Goetz, Yedidyah Langsam, and Theodore Raphan. Development of a system for teaching c/c++ using robots and open source software in a cs1 course. In *FECS*, pages 141–146, 2009.

[5] Zachary Dodds, Lloyd Greenwald, Ayanna Howard, Sheila Tejada, and Jerry Weinberg. Components, curriculum, and community: Robots and robotics in undergraduate ai education. *AI magazine*, 27(1):11–11, 2006.

[6] Martina Doolan and Michael Walters. Repurposing the learning environment: using robots to engage and support students in collaborative learning through assessment design. In *European Conference on e-Learning*, page 166. Academic Conferences International Limited, 2016.

[7] Barry Fagin and Laurence Merkle. Measuring the effectiveness of robots in teaching computer science. In *Acm sigcse bulletin*, volume 35, pages 307–311. ACM, 2003.

[8] Barry S Fagin and Laurence Merkle. Quantitative analysis of the effects of robots on introductory computer science education. *Journal on Educational Resources in Computing (JERIC)*, 2(4):2, 2002.

[9] Laura M Grabowski and Pearl Brazier. Robots, recruitment, and retention: Broadening participation through cs0. In *2011 Frontiers in Education Conference (FIE)*, pages F4H–1. IEEE, 2011.

[10] Emily Hamner and Jennifer Cross. Arts & bots: Techniques for distributing a steam robotics program through k-12 classrooms. In *2013 IEEE Integrated STEM Education Conference (ISEC)*, pages 1–5. IEEE, 2013.

[11] Emily Hamner, Tom Lauwers, Debra Bernstein, Illah R Nourbakhsh, and Carl F DiSalvo. Robot diaries: Broadening participation in the computer science pipeline through social technical exploration. In *AAAI spring symposium: using AI to motivate greater participation in computer science*, pages 38–43. Palo Alto, CA, 2008.

[12] Allaa R Hilal, Khaled M Wagdy, and Alaa M Khamis. A survey on commercial starter kits for building real robots. In *Proceedings of the International Conference on Electrical Engineering*, 2007.

[13] Rosalyn S Hobson. The changing face of classroom instructional methods: service learning and design in a robotics course. In *30th Annual Frontiers in Education Conference. Building on A Century of Progress in Engineering Education. Conference Proceedings (IEEE Cat. No. 00CH37135)*, volume 2, pages F3C–20. IEEE, 2000.

[14] Stephanie Ludi. Educational robotics and broadening participation in stem for underrepresented student groups. In *Robots in K-12 education: A new technology for learning*, pages 343–361. IGI Global, 2012.

[15] Muhsin Menekse. Computer science teacher professional development in the united states: a review of studies published between 2004 and 2014. *Computer Science Education*, 25(4):325–350, 2015.

[16] Innwoo Park, Donjeong Kim, Junghyuk Oh, Yoonho Jang, and Keol Lim. Learning effects of pedagogical robots with programming in elementary school environments in korea. *Indian Journal of Science and Technology*, 8(26):1–5, 2015.

[17] Andrew Ray. Evolving the usage of lego robots in cs1 to facilitate highlevel problem solving. In *Proceedings of the IASTED International Conference on Computers and Advanced Technology in Education (CATE'12). ICTA Press*, pages 91–98, 2012.

[18] Marco Ruzzenente, Moreno Koo, Katherine Nielsen, Lorenzo Grespan, and Paolo Fiorini. A review of robotics kits for tertiary education. In *Proceedings of International Workshop Teaching Robotics Teaching with Robotics: Integrating Robotics in School Curriculum*, pages 153–162, 2012.

[19] Newton Spolaôr and Fabiane B Vavassori Benitti. Robotics applications grounded in learning theories on tertiary education: A systematic review. *Computers & Education*, 112:97–107, 2017.

[20] Jay Summet, Deepak Kumar, Keith O'Hara, Daniel Walker, Lijun Ni, Doug Blank, and Tucker Balch. Personalizing cs1 with robots. In *ACM SIGCSE Bulletin*, volume 41, pages 433–437. ACM, 2009.

[21] Kaitlyn Vitez. An Action Plan for Affordable Textbooks. Technical report, 2018.

Animated Hints Help Novices Complete More Levels in an Educational Programming Game*

Michael J. Lee and Joseph Chiou
Department of Informatics
New Jersey Institute of Technology
Newark, NJ 07102
{mjlee,jcc45}@njit.edu

Abstract

Many people are learning programming on their own using various online resources. Unfortunately, learners using these resources often become disengaged or even quit when encountering an obstacle they cannot overcome without additional help. Teachers in a classroom can provide this type of help, but this may be impractical or impossible to implement in online educational settings. To address this issue, we added a visually-oriented hint system into an existing online educational game designed to teach novices introductory programming concepts. We implemented three versions of the hint system, providing equivalent information for each level of the game, adjusting the amount of interactivity between versions. The first version consisted of a static image with text showing how to solve a level in a single panel. The second version included a series of images that allowing users to scroll through hints step-by-step. The final version showed a short video allowing users to play, pause, and seek through animated hint(s). In total, we had 150 people play the game, randomly assigned to one of these three versions of the hint system. We found that users had a strong preference for the video version of the hint system, completing more levels. Based on these findings, we propose suggestions for designers of online educational tools to better support their users.

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

In recent years, there has been an increasing interest in enhancing computing literacy. Rising availability of online learning resources, such as tutorial sites (e.g., Codecademy.org, Kahn Academy), block programming environments (e.g., Scratch), and educational games (e.g., Swift Playgrounds), are popular choices for people to acquire programming knowledge [21]. However, despite these numerous online educational resources for learners to choose from, there continues to be high dropout/attrition rates. Researchers and educators attribute this to a lack of support [17, 18, 21], user frustration, a lack of motivation to continue studying, and no direct interaction with an instructor [23].

One potential way to address high dropout rates is to provide users with additional help, or hints [22]. Textual hints are often used to provide important information to users [5, 13]. Studies have shown that textual hints can positively influence users' in-system behavior and time spent on tasks [5, 27]. In addition to textual hints, many non-educationally focused systems, such as games, use visually-oriented hints to help their users overcome obstacles [2]. To the best of our knowledge, we have not found any research examining the effectiveness of visually-oriented hints in educational programming games.

To address the challenges of high dropout/attrition rates in online programming resources, we explore the use of interactive graphics as an alternative form to present hints. In this study, we examine how three different visualized hints—a static image, a carousel (series) of images, and a video clip—affects user retention in an educational game. Based on the success of certain types of text-based [5, 13] and visually-oriented hints [2] we hypothesized that a hybrid approach—the carousel hints—would provide a good combination of interactivity, text, and visual aids to assist users overcome obstacles and ultimately complete more levels than the other two conditions.

2 Related Work

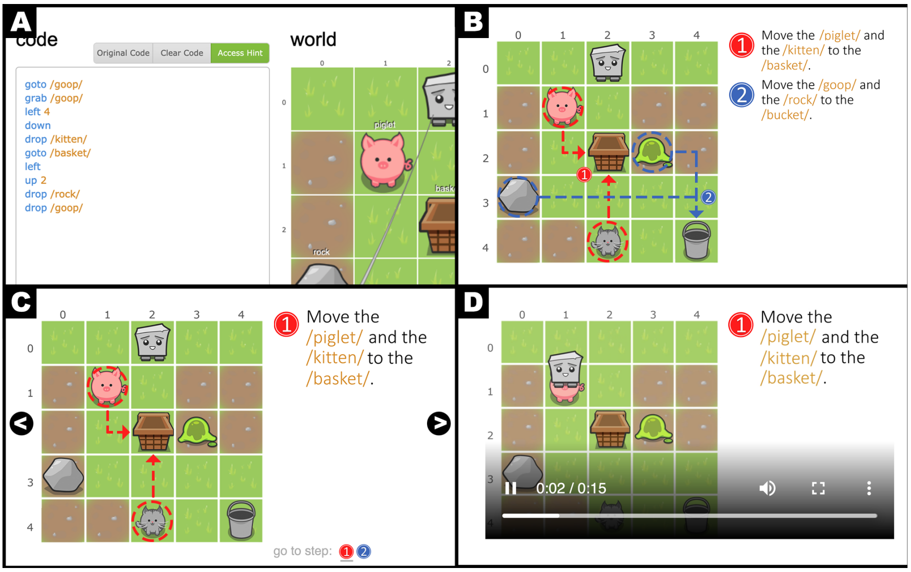
Engagement, and how it affects learning, has been widely studied in educational contexts. Student engagement has been shown to be key for student success at all grade levels [6, 9, 19]. As students become more engaged in learning, they improve learning outcomes and academic achievements [14, 19]. Engagement is essential for learning challenging topics such as computer programming [4], and educational games for teaching introductory programming concepts have shown to be successful at attracting a wide range of learners [8, 16, 22]. However, even with the success of these resources, users of these systems who encounter difficulties and not do not receive the support they need to overcome difficulties may become frustrated and quit the activity/topic.

To address these types of obstacles and frustrations, teachers often provide personalized, directed feedback to their students in classrooms. Online learning contexts—where teachers are unavailable or at a premium—such as Massive Open Online Courses (MOOCs) and tutorial websites often utilize help and/or hint systems of varying sophistication to help their users. Research shows that students perform better in learning environments when hints are provided [3]. Moreover, studies have found that the content of hints may have different effects, where high level hints tend to lead to long term positive effects, and detailed hints tend to be more useful immediately [5, 22].

The visual and interactive aspects of hints may also be important factors to consider when evaluating the usefulness of hints for online learners to overcome obstacles. According to Presmeg, visualization aids one in understanding a problem or a concept in a different modality and perspective, enabling them to better seek solutions [24]. Similarly, Gangwer suggests that students combine visualizations with active learning strategies to develop better mental models of problems and actively work on different approaches to solve them [7]. Finally, using visualization/graphics has shown to promote student learning and create opportunities for them to apply what is taught [12, 15, 20]. However, there is little consensus of how interactive graphics (e.g., static vs. animated images) compare in their utility for helping learners overcome obstacles [1, 11, 25, 26], especially in different online learning environments. In this study, we aim to explore this space, specifically examining how different types of hint visualizations, ranging from static images to animations, may affect learners' motivation to continue playing an educational programming game.

3 Method

The goal of our study was to determine how different types of visual hints in an educational computing game affects engagement and task completion rates in self-directed learners, and to identify the extent of these effects. To do this, we modified *Gidget* (see Figure 1-A; www.helpgidget.org), a freely available online game, adding new types of hints. The game has a total of 37 levels, where each level teaches a new programming concept (e.g., variable assignment, conditionals, loops, functions, objects) using a Python-like language [16, 18]. The goal of each level is to debug existing code to pass 1-4 test cases (i.e., statements that evaluate to 'true') upon running the code. After code execution, the game displays which test cases were successful and which ones failed. The game already includes a set of help features to assist players overcome obstacles while coding on their own [17]. These include popup bubbles explaining different code components, a dictionary explaining keywords [17], and a context-aware, implementation of the Idea Garden [13] to assist with programming anti-patterns.



```
goto /goop/
grab /goop/
left 4
down
drop /kitten/
goto /basket/
left
up 2
drop /rock/
drop /goop/
```

1 Move the /piglet/ and the /kitten/ to the /basket/.

2 Move the /goop/ and the /rock/ to the /bucket/.

Figure 1: *A: the "Access Hint" button, displayed above the main code pane in green; B: the static image hint condition; C: the carousel hint condition; and D: the video hint condition.*

Users could access the new hint system by clicking on a button labeled *Access Hint*, which was prominently displayed above the game's coding pane (see Figure 1-A). This button opened one of three different kinds of visual hints, which served as the independent variable we manipulated in our experiment: (1) a ***static*** image with the entire hint for the level displayed in one panel (see Figure 1-B); (2) a ***carousel***, or sequence of images showing one step of a hint at a time, allowing the player to scroll left or right through each hint panel (see Figure 1-C), or (3) an ***animation*** hint in the form of a short, 5-50 second video clip showing one step of a hint at a time, with controls allowing users to pause, play, and scroll through the clip (see Figure 1-D). We created customized hints for each of the 37 levels in the game, ensuring that each level's three visual hint systems conveyed equivalent information so that we could make a fair comparison among them. We used a "divide and conquer" approach to breaking down each level's hints into 1-5 smaller tasks (depending on the complexity of the level) which were organized into a numbered list on the right side of the hint, along with a graphical representation of the state of the system on the left. We also did not want players to exploit the hint system to get the exact answer(s) to complete the level [13], so we did not provide actual code. Instead, the hint system presented one possible path and actions that the character could take through the level to complete it successfully.

3.1 Participants

We recruited our participants on Mechanical Turk (MTurk), specifically sampling adults who self-reported that they had no experience with programming—those who responded "never" to all of the following statements: 1) "taken a programming course," 2) "written a computer program," and 3) "contributed code towards the development of a computer program." We also required participants to be U.S. residents to minimize English language barriers with the instructions and activities. We followed our previous work's pricing model from a similarly scoped study [16], adjusting the payment to US$5 to better reflect the task difficulty and other similar HITs and prices on MTurk at the time. To help participants make an informed decision about the time commitment required to participate in our study, we told them that they would be playing a puzzle game for as long (or as short) as they wanted, over a maximum of seven days so they could have flexibility in their play time(s). Our HIT was labeled as "5 hours" for the task time, but emphasized that this was an estimate, and that they could quit the task at any time without negative repercussions.

Once an MTurker accepted the HIT, they were required to fill out the form certifying they were a novice programmer, and to read and digitally sign the informed consent form agreeing to participate in the experiment. Once they did so, they were redirected to the game website to make an account (requiring an e-mail address, password, gender, and age). Each participant was randomly assigned to one of the hint conditions, and this information was saved so that they would always see their assigned type of hint whenever they played the game. The introductory tutorial for the game (shown automatically the first time someone logs in) highlighted the *Access Hint* button and included text encouraging players to use it when they needed help. For the purposes of this study, we logged the total number of levels the participant completed, how many times they pressed the *Access Hint* button per level, and the cumulative time they had the hint window open per level.

4 Results

We provide quantitative results comparing the outcomes from our three groups using nonparametric Chi-Squared and Wilcoxon rank sums tests with $\alpha = 0.05$ confidence, as our data were not normally distributed. For post-hoc analyses, we use the Bonferroni correction for three comparisons: ($\alpha/3 = 0.01\overline{6}$).

Our study was a between-subjects design, with an even split of 50 people each among the three conditions. Demographic data revealed that there were no significant differences between groups by age (range 18-54 years old; median 22) or gender (88 females and 62 males). The key dependent variable in our study was engagement, which we operationalized as the number of levels completed.

We also examine the participants' use of the hint system (number of times accessed per level and total time open per level).

4.1 Animation Condition Participants Complete More Levels

All participants completed at least seven levels. The range of levels completed in the static, carousel, and animation conditions were 7-33 (median 10), 7-37 (median 13), and 7-37 (median 13), respectively. There was a significant difference in the number of levels participants completed between the three conditions ($\chi^2(2, N = 150) = 7.0276, p < .05$). Further post-hoc analysis with a Bonferroni correction shows that the significantly different pair was the static vs. animation conditions ($W = 14.64, Z = 2.541, p < .01\overline{6}$), with the animation group completing more levels. The static vs. carousel ($W = 11.52.5, Z = 1.996, p < .05$) comparison trended towards significance (p-value was less than .05, but not less than the correction threshold of $0.01\overline{6}$) with the carousel group completing more levels. Finally, comparing the carousel vs. animation conditions showed no significant difference ($W = 1.58, Z = 0.274, n.s.$).

Since all participants were novice programmers with no statistical difference in demographics, these results suggest that something about interacting with the animated hints (and to a lesser degree, the carousel hints), had a significant positive effect on participants' engagement and ability to complete more levels in the game compared to the other condition(s). This was surprising, as we had hypothesized that the carousel hints would help learners the most because it would allow quick, directed access to different parts of the hints (instead of having to look through a long list of hints, or seek through a video).

4.2 No Significant Differences in Accessing the Hint System

All participants used their respective hint system at some point during their gameplay. The range of access to the hint system in the static, carousel, and animation conditions per level were 0-11, 0-11, and 0-10, respectively. Because everyone completed a different number of levels, we calculated the average number of times each participant accessed the hint system per level (sum of number of times they pressed the hint button throughout their entire gameplay record, divided by the farthest level number they reached) for our analysis. There was no significant difference in the number of times participants accessed the hint system among the three conditions ($\chi^2(2, N = 150) = 0.036, n.s.$).

We originally expected to see a different number of hint access across the conditions because the information from each was conveyed so differently. For example, we thought that the static image would be accessed the least, since it showed the entirety of a hint in one image, and that the users of the carousel and animated hints would have to jump back and forth between their code and hints more often since the hints were broken down into smaller sections. However, this was not the case, with participants accessing their respective hints a similar number of times on average per level. Combined with the previous result, this suggests that animation participants made better use of their time when accessing a hint, as they clicked on their hint button a similar number of times as their counterparts, but ended up completing more levels.

4.3 Static Hint Participants Spend Less Time Looking at Hints

To further examine our last result, we examined how long participants looked at their respective hints (i.e., time the hint window was open). The range of looking at the hint system in the static, carousel, and animation conditions were 0-87 seconds (median 46), 0-81 seconds (median 62), and 0-96 seconds (median 65), respectively. Because everyone completed a different number of levels, we calculated the average time each participant spent looking at a hint per level (sum of the cumulative time they looked at a hint within each level throughout their gameplay record, divided by the number of the farthest level they reached) for our analysis. There was a significant difference in the time participants looked at their respective hints between the three conditions ($\chi^2(2, N = 150) = 8.335, p < .05$). Further post-hoc analysis with a Bonferroni correction shows that the significantly different pair was the static vs. carousel conditions ($W = 14.28, Z = 2.462, p < .01\overline{6}$) and the static vs. animation conditions ($W = 14.40, Z = 2.482, p < .01\overline{6}$) with the static group spending less time on the hints. Comparing the carousel vs. animation conditions showed no statistically significant difference between the two ($W = 3.140, Z = 0.541, n.s.$).

Similarly to the reasoning we described above, we expected (and found) that participants spent the least amount of time looking at the static hints (since everything was displayed in one image), and more time looking at the videos (since each video hint had a specific pace and run time), with the carousel being somewhere in the middle (since the user could skip to specific, labeled parts of the hints on demand). Combining the last two results with this shows that animated hint users (and to a certain degree, the carousel hint users) spent a little more time looking at the same number of hints, but were more successful in completing levels than the other condition(s), suggesting something about the animated hints helped users better apply the information to complete levels.

5 Discussion & Conclusion

Our findings show that animated hints (and to a certain degree, carousel hints) can significantly improve users' performance in an educational game. The animated hint condition group participants completed significantly more levels than their static hint counterparts, while looking at their respective hints a similar number of times. Our results have several potential interpretations for better understanding hints in the context of educational games.

We tried to keep the information content of the three conditions equivalent—mainly manipulating the visual density and interactivity of each condition—but found significant differences in outcomes. A possible interpretation of our results is that showing users different visual states of a program (i.e., a screenshot of beginning state, some middle states, and an end state) can help users better understand the goals of a level. Both the animation and carousel hints showed the users what their program should look like in at least three different stages. On the other hand, the static hints presented everything in one picture, which may not have significant impact on helping understand the goal of a level and cause information overload. Information overload can be a factor that negatively affect users' information acquiring process [10]. To further this case, we observed that our static hint users spent less time with their hint windows open compared to the other two conditions, even though users from all three conditions opened their respective hint windows the same number of times. This may mean that users were able to use the time they had their hint window open more efficiently when the hints were broken down into smaller, more digestible chunks. Our results show that hints in educational games can be represented and interacted with in different ways, and that small changes can have a significant impact on user engagement.

We have several limitations to our study. First, we recruited participants on MTurk, which might not be representative of the larger population. However, our groups were similar to each other, with no significant differences by age or gender. Second, we provided an economic incentive for people to participate in our study, which may have affected their engagement and sense of obligation to complete levels. To counteract this, our payment was low compared to the estimated time to complete the task, and allowed participants to quit at any time. Despite this, we found that participants were engaged with the task, spending hours playing the game and everyone completing a minimum of 7 levels, suggesting that they were entertained and not playing the game for the monetary compensation. Third, all participants completed a different number of levels, making it difficult for us to get a consistent count of overall times people accessed and looked at hints across the same levels. In our analyses, we calculated the average count and time each participant took through their play of the game, which may have introduced some level of inaccuracy. However, this was intentional as we did not want to force all our players to complete the entire game, which might be difficult and/or unreasonable for some participants, and also because our main goal was to measure engagement as a function of how many levels users in each condition completed. Future studies may ask participants to complete all levels and/or collect qualitative measures from users, asking them how they felt about the hints they used. Finally, we plan to measure learning outcomes (using pre-post tests) to determine users' knowledge before and after playing the game using these different hint systems.

In conclusion, our study examined how different visualized hints affect users' engagement with an online educational game. We found that participants using animated hints—video clips that allowed users to pause, play, and seek through numbered hints—were more engaged with the game, completing more levels. Our findings suggest that interactive, visual hints that are subdivided into smaller parts showing different states of a program during execution assist learners in understand programming task goals. Researchers, educators, and designers of these online learning systems may benefit by utilizing these types of hints. Future work will explore these findings further, especially with different types of online resources to explore potential differences and similarities.

6 Acknowledgements

This work was supported in part by the National Science Foundation (NSF) under grants DRL-1837489 and IIS-1657160. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of the NSF or other parties.

References

[1] Erik Andersen, Yun-En Liu, Rich Snider, Roy Szeto, and Zoran Popović. Placing a value on aesthetics in online casual games. In *ACM CHI*, 2011.

[2] Erik Andersen, Eleanor O'Rourke, Yun-En Liu, Rich Snider, et al. The impact of tutorials on games of varying complexity. In *ACM CHI*. ACM, 2012.

[3] John R Anderson, Albert T Corbett, Kenneth R Koedinger, and Ray Pelletier. Cognitive tutors: Lessons learned. *Journal of the Learning Sciences*, 4(2), 1995.

[4] Lori Carter. Why students with an apparent aptitude for computer science don't choose to major in computer science. In *ACM SIGCSE Bulletin*, volume 38, 2006.

[5] Christa Cody, Behrooz Mostafavi, and Tiffany Barnes. Investigation of the influence of hint type on problem solving behavior in a logic proof tutor. In *International Conference on AI in Education*, pages 58–62. Springer, 2018.

[6] Lyn Corno and Ellen B Mandinach. What we have learned about student engagement in the past twenty years. *Big theories revisited*, 4:299–328, 2004.

[7] Timothy Gangwer. *Visual Impact, Visual Teaching: Using Images to Strengthen Learning.* Simon and Schuster, 2015.

[8] Nan Gao, Tao Xie, and Geping Liu. A learning engagement model of educational games based on virtual reality. In *IEEE ICIME*, pages 1–5, 2018.

[9] Rosemary Garris, Robert Ahlers, and James E Driskell. Games, motivation, & learning: A research & practice model. *Simulation & Gaming*, 33(4), 2002.

[10] Kyle J Harms. Applying cognitive load theory to generate effective programming tutorials. In *IEEE VL/HCC*, pages 179–180, 2013.

[11] Tim N Höffler and Detlev Leutner. Instructional animation versus static pictures: A meta-analysis. *Learning and Instruction*, 17(6):722–738, 2007.

[12] Jozef Janitor, František Jakab, and Karol Kniewald. Visual learning tools for teaching/learning computer networks: Cisco networking academy and packet tracer. In *IEEE ICNS*, pages 351–355, 2010.

[13] Will Jernigan, Amber Horvath, Michael Lee, Margaret Burnett, et al. A principled evaluation for a principled idea garden. In *IEEE VL/HCC*, 2015.

[14] Greg Kearsley and Ben Shneiderman. Engagement theory: Framework for technology-based teaching and learning. *Educational Technology*, 38(5), 1998.

[15] Daesang Kim and David A Gilman. Effects of text, audio, and graphic aids in multimedia instruction for vocabulary learning. *Journal of Educational Technology & Society*, 11(3):114–126, 2008.

[16] Michael J Lee. Teaching and engaging with debugging puzzles. *University of Washington, Seattle, WA*, 2015.

[17] Michael J Lee, Faezeh Bahmani, Irwin Kwan, et al. Principles of a debugging-first puzzle game for computing education. In *IEEE VL/HCC*, 2014.

[18] Michael J Lee, Amy J Ko, and Irwin Kwan. In-game assessments increase novice programmers' engagement and level completion speed. In *ACM ICER*, 2013.

[19] Helen M Marks. Student engagement in instructional activity: Patterns in the elementary, middle, and high school years. *American educational research journal*, 37(1):153–184, 2000.

[20] Richard E Mayer and Roxana Moreno. Aids to computer-based multimedia learning. *Learning and Instruction*, 12(1):107–119, 2002.

[21] Daniel Fo Onah, Jane Sinclair, and Russell Boyatt. Dropout rates of massive open online courses: behavioural patterns. *EDULEARN*, 1:5825–5834, 2014.

[22] Eleanor O'Rourke, Christy Ballweber, and Zoran Popovií. Hint systems may negatively impact performance in educational games. In *ACM LS*, 2014.

[23] Angie Parker. Identifying predictors of academic persistence in distance education. *Usdla Journal*, 17(1):55–62, 2003.

[24] Norma C Presmeg. Prototypes, metaphors, metonymies and imaginative rationality in high school mathematics. *Educational Studies in Mathematics*, 23(6):595–610, 1992.

[25] Fanny Ståhl and Hanna Holmgren. How does animated and static graphics affect the user experience in a game?, 2016.

[26] Barbara Tversky, Julie Bauer Morrison, and Mireille Betrancourt. Animation: Can it facilitate? *International J. of Human-Computer Studies*, 57(4), 2002.

[27] Kurt Vanlehn. The behavior of tutoring systems. *International Journal of AI in Education*, 16(3):227–265, 2006.

(Re)Engaging Novice Online Learners in an Educational Programming Game[*]

Michael J. Lee
Department of Informatics
New Jersey Institute of Technology
Newark, NJ 07102
`mjlee@njit.edu`

Abstract

Many people are learning programming on their own using various online resources such as educational games. Unfortunately, little is known about how to keep online educational game learners motivated throughout their game play, especially if they become disengaged or frustrated with their task. Keeping online learners engaged is essential for learning programming, as it may have lasting effects on their views and self-efficacy towards computer science. To address this issue, we created a coarse-grained frustration detector that provided users with customized, adaptive feedback to help (re)engage them with the game content. We ran a controlled experiment with 400 participants over the course of 1.5 months, with half of the players playing the original game, and the other half playing the game with the frustration detection and adaptive feedback. We found that the users who received the adaptive feedback when frustrated completed more levels than their counterparts who did not receive this customized feedback. Based on these findings, we believe that adaptive feedback is essential in keeping educational game learners engaged, and propose future work for researchers and designers of online educational games to better support their users.

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Online learning has expanded in popularity with the continued growth of the Internet. Online learning's popularity can be attributed in part to a number of advantages including flexibility, convenience, access, and low cost [9]. Online courses enable students to access materials anytime and anywhere, allowing them to work in an environment of their choice and at their own pace.

However, with the lack of in-person, face-to-face communications in online courses, keeping students motivated to learn can be challenging. Critics have long claimed that online learning is not as effective as traditional classroom learning because of the absence of face-to-face interactions [4]. In a classroom, teachers can gauge students' reactions, body language, and behaviors to determine if and when students are engaged with the course content. Teachers can use these cues to determine the best course of action to re-engage their students. Unfortunately, many of these opportunities to encourage learners go unfulfilled in online contexts (which are essential for any learning setting [1]), negatively affecting learning outcomes [13]. In fact, lack of motivation has been identified as a major cause of the high dropout rates in many online courses [19].

This project explores how feedback, based on users' actions in an online, educational programming game, might affect their motivation to complete more levels. We tested our game with and without a new feedback system with 400 new users, tracking their progress through the game for one week each (approximately 1.5 months total). Our goal was to test if we could successfully detect learners' frustration using models of past users, and how intervening with adaptive feedback might help them re-engage with the content/levels.

2 Related Work

2.1 Dropouts in Introductory Computing Courses

It is well established that introductory programming (CS1) courses in higher education have high dropout rates [11, 15]. A worldwide survey on completion rates reported that only an average of 67% students complete their CS1 course [3]. Further meta-analysis synthesizing 15 years of research on CS1 literature found that the mean pass rate of CS1 courses is 67.7% and that pass rates have not improved over time [3, 23]. Online teaching resources, such as Massive Open Online Courses (MOOCs) fare even worse, with recent numbers reporting fewer than 5% of users completing the curricula they sign up for [10]. Although there are fewer statistics of dropout rates for discretionary online learning settings such as educational games, it is reasonable to presume that rates would be similar, or possibly worse since these online settings lack the mechanisms that compulsory learning resources have to retain their students.

Much of the recent work on engagement in educational programming games has been conducted by our research lab using *Gidget* [16] (www.helpgidget.org). We investigated several strategies for preventing dropout (or *abandonment*), including more personalized error feedback [17] and the inclusion of in-game assessments [18], finding that features that anthropomorphized characters in the game or confirmed understanding could significantly increase engagement [17]. Outside the domain of coding, some researchers have successfully built predictive models of learners' motivational states in similar interactive learning environments [7, 21]. These systems have found predictive success using features related to help seeking, particularly the use of manuals and tooltips.

These and other efforts from prior work have several implications for coding tutorial abandonment prediction. First, many of the most predictive features in prior work have concerned social, instructional, and motivational factors, all of which are difficult to detect using a coding tutorial, especially if the users are using it anonymously. Moreover, the majority of studies have considered dropout at the end of a course of learning, leaving open the possibility that early detection of dropout is not feasible. That said, prior work suggests that some behavioral features, particularly indicators of frustration, may be strong predictors of either engagement or disengagement.

2.2 Detecting Frustration and Providing Feedback

There has been much work in modeling users and providing feedback to change their behavior in the fields of learning science and instructional design. Baker et al.'s work suggests the use of educational data mining and prediction modeling to have educational systems display messages to encourage positive behavior [2]. Rodrigo & Baker identified several coarse-grained measures (e.g., consecutive compilations with the same edit location, the number of pairs of consecutive compilations with the same error, the average time between compilations and the total number of errors) to detect frustration by observing students and analyzing their coding assignment logs [22]. Hattie & Timperley's survey of different kinds of feedback found that the most effective at engaging learners were not those that were related to praise, rewards, or punishment, but rather informative messages relating to feedback about a task and how to do it more effectively [12]. Similarly, Kickmeier-Rust et al. found that adaptive feedback (those relating to a user's current context) helped facilitate users' learning and educational game immersion. However, some researchers report the opposite effects, such as Conati & Manske, who found that adaptive feedback based on learners' actions in an educational game did not lead to learning gains [8]. Our study builds on these previous works, using models of past users' behavior data to detect learners' frustration as a basis to provide an intervention to re-engage them with the content and complete more levels.

3 Method

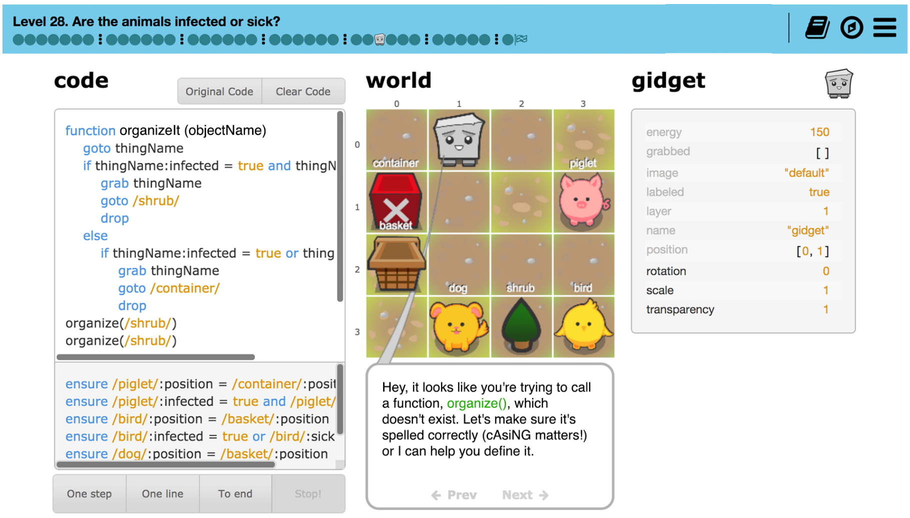
We modified our free, introductory coding game, *Gidget* (see Figure 1), adding a coarse-grained frustration detector that provided adaptive feedback. The game has a total of 37 levels, where each level teaches a new programming concept (e.g., variable assignment, conditionals, loops, functions, objects) using a Python-like language [16, 18]. For each level, a player has to debug existing code so that the protagonist character can complete its missions. The goal of each level is to pass 1-4 test cases (i.e., statements that evaluate to 'true') upon running the code. After code execution, the game displays which test cases were successful and which ones failed. Each level introduces at least one new programming concept, becoming progressively more difficult as players reach later levels. Therefore, completing more levels means that users are exposed to more programming concepts. Finally, the game also includes a set of help features to help players overcome obstacles while coding on their own [16].

3.1 Modeling Frustration

Based on our literature review and our own prior work using machine learning techniques to detect factors leading to game abandonment [24], we decided to focus on frustration as a primary predictor for addressing disengagement and game abandonment. We were inspired by prior work that found that coarse-grained predictors performed better than fine-grained predictors at detecting frustration [22] and used that model for our frustration detector. As a proof-of-concept, we also decided to to limit the number of factors our disengagement detector distinguished to reduce resource overhead (i.e., client/server processing requirements). We selected a total of five signs of frustration (loosely defined), with the first two adapted from Rodrigo & Baker's work [22] and the latter three adapted from our past work [24]:

1. deviations from the average number of consecutive code executions with the same edit location
2. deviations from the average time between code executions
3. deviations from the average number of code executions
4. deviations from the average time spent on a level
5. deviations from the average time without any activity (idle time)

We defined deviation as values exceeding two standard deviations from the calculated mean of any measure. This value was chosen because two standard deviations away from the mean can be considered "unusual," and we did not want to provide feedback too often (which could result in the *Clippy* effect, where users find the intervention bothersome rather than helpful [20]), or too



```
function organizeIt (objectName)
    goto thingName
    if thingName:infected = true and thingN
        grab thingName
        goto /shrub/
        drop
    else
        if thingName:infected = true or thing
            grab thingName
            goto /container/
            drop
organize(/shrub/)
organize(/shrub/)
```

```
ensure /piglet/:position = /container/:posit
ensure /piglet/:infected = true and /piglet/
ensure /bird/:position = /basket/:position
ensure /bird/:infected = true or /bird/:sick
ensure /dog/:position = /basket/:position
```

Hey, it looks like you're trying to call a function, organize(), which doesn't exist. Let's make sure it's spelled correctly (cAsiNG matters!) or I can help you define it.

energy	150
grabbed	[ ]
image	"default"
labeled	true
layer	1
name	"gidget"
position	[0, 1]
rotation	0
scale	1
transparency	1

Figure 1: *The experimental condition, displaying an adaptive message (bottom-center speech bubble) that is helping with a function call after detecting consecutive code executions with the same edit location.*

rarely (in which case we miss opportunities to re-engage users). We calculated all of the means and standard deviations for each of the frustration measures above using a data set of 15,448 past users' game logs. These game logs were detailed, including individual players' time spent on level, idle time, all of their code edits, clicks, keystrokes, and execution button usage.

3.2 Adaptive Feedback

We took Hattie & Timperley's [12] and Kickmeier-Rust et al.'s [14] approach in providing users with customized, adaptive feedback relating to their current context (as described in Section 2.2). Our objective was to provide users with contextually relevant information to help (re)engage them with their current task, without giving them exact solutions. To do so, we adapted the design of the Idea Garden, a help system that examines and transforms users' code to provide relevant, but not exact, code examples [5, 6]. For all five cases described above, we used the Idea Garden analysis methodology (described in [5]) with the most recently executed or current version of the users' code to generate a customized, adaptive message for the user (see Figure 1). Longer messages were split into multiple panels that the user could click through (forward and backwards). Finally, because we did not want to directly interrupt the user and allow them to ignore the message if they wanted to, we had the message generator fade text into the protagonist character's permanent speech bubble at the bottom of the screen. Examples include:

- Hey, it looks like you're trying to call a function, `checkBaskets()`, which doesn't exist. Let's make sure it's spelled correctly (cAsiNG matters!) or I can help you define it.
- You're almost there! The last thing you were working on was on *Line 5*, which seems to be inside a `for` loop block. Remember, `for` loops are written in the following way to iterate through each item in the list:
  ```
  for myPiglet in /piglets/s
      goto myPiglet
  ```
  Don't forget to add tabs for code belonging in the `for` code block!

3.3 Participant Recruitment

We tested our system with a group of 400 online users who were randomly assigned to the regular version of the game (the control condition participants) or a version of the game with the new feedback system (the experimental condition participants) during account creation. The game logged each user's condition so that they would only see the game version they were assigned to, even when coming back to play at a later time. The sign-up page required users to enter their age, gender, e-mail address, state whether they have prior programming experience, and agree/disagree to participating in a research experiment. For the purposes of this study, we only selected users that indicated they were at least 18 years old, had no prior programming experience, and were willing to participate in a research experiment. We set the observation window to 7 days (168 hours) per user to have a consistent timeframe for all users. To expedite the account creation procedure, we did not collect other demographic information such as ethnicity, geographical location, or education level. Participants were required to read and digitally sign an online consent form that briefly described the study. We were intentionally vague in our description of the messages participants might see, stating that we were "testing various types of messages to see how they might help players" to minimize any potential leading or biasing of participants focusing on specific types of messages. However, we e-mailed all participants a copy of the study procedures 7 days after the end of their individual observation window to debrief them, regardless of the condition they were assigned to. Prior to the debrief message (one day after their observation window ended) we sent an e-mail with a link to an optional online questionnaire that asked participants to rate their agreement to the following three statements about their experience with the game on a scale from 1 ('strongly disagree') to 7 ('strongly agree'):

1. The messages that Gidget provided helped me with my goals.

2. The messages that Gidget provided came up too often.
3. The messages that Gidget provided were distracting.

We intentionally under-specified *messages* (and their contents), so that participants from both conditions could interpret what *messages* were on their own. Our system e-mailed two different URLs containing the same questions to their respective participants to distinguish responses between the conditions.

4 Results & Discussion

We provide quantitative results comparing the outcomes from our three groups using nonparametric Chi-Squared and Wilcoxon rank sums tests with $\alpha = 0.05$ confidence, as our data were not normally distributed. Our study was a between-subjects design, with an even split of 200 participants in the control condition group (aged 18-54; median 20), and 200 participants in the experimental condition group (aged 18-55; median 20). Comparisons of demographic data revealed that there were no significant differences between the control and experimental conditions by age or gender (107 males, 88 females, and 5 other or decline to state; and 102 males, 90 females, and 8 other or decline to state, respectively). The key dependent variable in our study was engagement, which we operationalized as the number of levels completed. We also examine the participants' responses to the optional questionnaire.

4.1 Experimental Condition Participants Complete More Levels

All participants completed at least four levels. The range of levels completed in the control and experimental conditions were 4-37 (median 10) and 4-37 (median 13), respectively. We verified that all participants in the experimental condition saw messages from the new feedback system throughout their time playing the game (with more occurring in later, more difficult stages). There was a significant difference in the number of levels participants completed between the two conditions ($W = 42385, Z = 1.986, p < .05$), with the experimental group participants completing more levels.

Since all participants were novice programmers, these results suggest that something about interacting with the new feedback system (frustration detector and adaptive message generator), had a significant positive effect on the experimental condition participants' engagement and ability to complete more levels in the game compared to the control condition participants.

4.2 Unable to Compare Differences in Play Times

Next, we had planned to measure the differences in how long participants took to complete the levels they passed. However, because everyone completed a different number of levels and the range of completion times for each level were vastly different, we would only able to compare the levels that all 400 participants completed (i.e., Levels 1-4) to see if there were any differences in play times. However, we found that only a few (11 out of 200) participants in the experimental condition received at least one of the new feedback system messages during the first four levels. Therefore, we were unable to compare the differences between the control and experimental group play times since the majority of the experimental group (189/200, or 94.5%) did not experience anything differently from the control group for these common completed levels.

4.3 Experimental Group Agrees Messages Helped with Goals

Finally, we compared our optional questionnaire responses, which had a total response rate of 10.25%, (19 control, 22 experimental). For our analyses, we flipped the scales for Questions 2 and 3 since the statements were negative.

For Questions 1 and 2, our median scores for both conditions were 6 (range 4-7) and 4 (range 2-6), respectively. For Question 3, the control and experimental conditions were 3 and 4 (range 2-6), respectively. Additionally, we found a significant difference between the control and experimental groups agreement to Question 1 ($\chi^2(3, N = 41) = 8.299, p < .05$). However, we did not find significant differences between conditions for Question 2 ($\chi^2(4, N = 41) = 2.410, n.s.$) or Question 3 ($\chi^2(4, N = 41) = 1.385, n.s.$)

We had not expected to find significant differences in our questions because of the low response rate and were excited to find that the experimental group users reported that their messages helped them with their goals—which was the aim of this study. However, we need to explore this result further in future work, as we did not specify exactly which messages in our questions.

5 Conclusion

Our findings show that adaptive feedback messages, triggered by a frustration detector using coarse measures, can significantly improve users' performance in an educational game. In our study, our experimental group participants (those with the frustration detector and adaptive feedback messages) completed significantly more levels than their control group counterparts (who played the game without these additional features). Researchers and educators for online resources for teaching programming may benefit from adding these types of frustration detection and adaptive feedback to their systems.

We have several limitations to our study. First, we recruited participants who opted into a research study. These types of participants may already have high motivation, and therefore may not be completely representative of the larger population. However, we found that the participants in our two groups were similar to each other, with no significant differences by age or gender. Second, participants from both groups completed a different number of levels, making it impossible for us to compare their usage of the new feedback system (especially because the frustration detector was triggered more often in later levels, which many participants did not reach). Third, we had a low questionnaire response rate in comparison to our full participant pool, which may limit the generalizability of the findings. Future studies may ask participants to complete all levels and everyone to fill out the questionnaire. Finally, we also plan to measure learning outcomes (using pre-post tests) to determine how this feedback system affects learners' knowledge.

Our results from this proof-of-concept study shows that adaptive feedback, triggered by coarse measures to detect frustration, are sufficient in increasing online learners' performance. Our future work will examine these outcomes in more detail to determine what exactly is causing these effects, along with additional coarse frustration predictors, and possibly some fine-grained predictors.

6 Acknowledgements

This work was supported in part by the National Science Foundation (NSF) under grants DRL-1837489 and IIS-1657160. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of the NSF or other parties.

References

[1] Susan A Ambrose, Michael W Bridges, Michele DiPietro, Marsha C Lovett, and Marie K Norman. *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons, 2010.

[2] Ryan Shaun Baker and Paul Salvador Inventado. Educational data mining and learning analytics. In *Learning Analytics*, pages 61–75. Springer, 2014.

[3] Jens Bennedsen and Michael E Caspersen. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2):32–36, 2007.

[4] Mark Bullen. Participation and critical thinking in online university distance education. *Int. Journal of E-Learning & Distance Education*, 13(2):1–32, 2007.

[5] Jill C Cao. Helping end-user programmers help themselves: the idea garden approach. *Oregon State University, Corvallis, OR*, 2013.

[6] Jill C Cao, Scott D Fleming, Margaret Burnett, and Christopher Scaffidi. Idea garden: Situated support for problem solving by end-user programmers. *Interacting with Computers*, 27(6):640–660, 2014.

[7] Mihaela Cocea and Stephan Weibelzahl. Eliciting motivation knowledge from log files towards motivation diagnosis for adaptive systems. In *International Conference on User Modeling*, pages 197–206. Springer, 2007.

[8] Cristina Conati and Micheline Manske. Evaluating adaptive feedback in an educational computer game. In *International Workshop on Intelligent Virtual Agents*, pages 146–158. Springer, 2009.

[9] Simone CO Conceição. Faculty lived experiences in the online environment. *Adult Education Quarterly*, 57(1):26–45, 2006.

[10] Wenzheng Feng, Jie Tang, and Tracy Xiao Liu. Understanding dropouts in moocs. *Association for the Advancement of AI*, 2019.

[11] Mark Guzdial and Elliot Soloway. Teaching the nintendo generation to program. *Communications of the ACM*, 45(4):17–21, 2002.

[12] John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1):81–112, 2007.

[13] Starr Roxanne Hiltz. Collaborative learning in asynchronous learning networks: Building learning communities. 1998.

[14] Michael D Kickmeier-Rust, Birgit Marte, SB Linek, Tiphaine Lalonde, and Dietrich Albert. The effects of individualized feedback in digital educational games. In *European Conference on Games Based Learning*, pages 227–236. Academic Publishing Limited, 2008.

[15] Päivi Kinnunen and Lauri Malmi. Why students drop out cs1 course? In *ACM ICER*, pages 97–108, 2006.

[16] Michael J Lee. Teaching and engaging with debugging puzzles. *University of Washington, Seattle, WA*, 2015.

[17] Michael J Lee and Amy J Ko. Personifying programming tool feedback improves novice programmers' learning. In *ACM ICER*, pages 109–116, 2011.

[18] Michael J Lee, Amy J Ko, and Irwin Kwan. In-game assessments increase novice programmers' engagement and level completion speed. In *ACM ICER*, 2013.

[19] Lin Y Muilenburg and Zane L Berge. Student barriers to online learning: A factor analytic study. *Distance education*, 26(1):29–48, 2005.

[20] Emerson Murphy-Hill and Gail C Murphy. Recommendation delivery. In *Recommendation Systems in Software Engineering*, pages 223–242. Springer, 2014.

[21] Lei Qu and W Lewis Johnson. Detecting the learner's motivational states in an interactive learning environment. In *AI in Education: Supporting Learning through Intelligent and Socially Informed Tech.*, pages 547–554. IOS Press, 2005.

[22] Ma MT Rodrigo and Ryan S Baker. Coarse-grained detection of student frustration in an introductory programming course. In *ACM ICER*, 2009.

[23] Christopher Watson and Frederick WB Li. Failure rates in introductory programming revisited. In *ACM ITiCSE*, pages 39–44, 2014.

[24] An Yan, Michael J Lee, and Amy J Ko. Predicting abandonment in online coding tutorials. In *IEEE VL/HCC*, pages 191–199, 2017.

Dealing with Uncertainty: a PiecewiseGrid Agent for Reconnaissance Blind Chess[*]

Timothy Highley, Brendan Funk, Laureen Okin
Department of Mathematics and Computer Science
La Salle University
Philadelphia, PA 19141
highley@lasalle.edu

Abstract

Reconnaissance Blind Chess (RBC) is a two-player chess variant where players do not have complete knowledge of the game state. Each turn, a player has a sense action and a move action. For the sense action, the player chooses a square on the board and then is informed of the identity of all pieces in that square and the eight surrounding squares. The only information about the locations of opposing pieces comes from the results of a sense action or when an opponent captures one of your pieces. The move action is a standard chess move with slight rule changes to adjust for the incomplete knowledge of the game state. Whenever a player captures an enemy piece, the player learns that a capture took place but not which enemy piece was captured. This paper presents a rules-based agent for playing RBC that took second place in the first worldwide RBC competition: the Fall 2019 NeurIPS RBC tournament sponsored by Johns Hopkins University Applied Physics Laboratory. Following a more detailed explanation of RBC, the fundamental underlying data structure that the agent uses to track game state is presented: the PiecewiseGrid. The PiecewiseGrid maintains a separate probability distribution for each piece, reflecting the agent's belief about where that piece might be located on the board. This allows the agent to track possible game states and their relative likelihood in a space-efficient manner. Strategies for choosing where to sense, making a move, and updating the PiecewiseGrid are also presented.

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

The game of chess has long been a testbed for advances in artificial intelligence. Reconnaissance Blind Chess (RBC) is a recently introduced variation of regular chess [2] in which the players do not know exactly where the opponent's pieces are. That is the "blind" part of the game. The reconnaissance part of the game comes from the sensing actions that the players take. On a player's turn, they take a sense action and a move action. In a sense action a square is chosen, and then that square and the eight immediately adjacent squares – a three by three block – are revealed. The contents of those squares (if anything) are revealed but only to the player that took the sense action. Only what is in the sensed squares is certain; everything else is unknown. Aside from that, the rules of the game are generally the same as regular chess. The king is the target. However, rules for check and checkmate are ignored since players may not be aware that they are even in check. Instead, the game ends when a king is actually captured. If a player attempts an illegal move, such as a rook moving through an enemy piece, the piece will move as far as it can. This may result in an unexpected capture.

Markowitz, et al analyzed RBC and determined that when a player is facing an uncertain situation in RBC, on average the number of possible game states that are consistent with prior observations exceeds that of a player in Texas Hold 'Em poker [1]. They looked at several different approaches to RBC and how they impact the number of game states that a player is facing.

An RBC player, whether human or an AI agent, must deal with uncertainty in order to succeed. In the agent that is described in this paper, piecewise probability distributions are used to generate sample boards that are fed to a leading open-source chess engine: Stockfish [3]. The engine is used to both decide the agent's own moves and to guess the opponent's moves. The predictions of opponent moves are then used to update the piecewise probability distributions. For sense actions, the agent essentially senses the individual square that has the greatest uncertainty (i.e. a piece whose probability of being in a specific square is closest to 50%).

Some of the strategies employed here are refinements of other RBC strategies published by Markowitz, et al [1] or that have been discussed elsewhere. However, the piecewise probability distributions for representing the game state that are presented here is a novel approach in the context of RBC.

2 RBC Agent

Any RBC agent must tackle four problems. In the face of uncertainty, how will the game state be represented? As moves are taken, how will the game state be updated? Based on the game state, what move should the agent take?Based on the game state, what sense action should the agent take? This section answers each of these questions in turn.

2.1 PiecewiseGrid: Representing the Game State

The board state is represented as a collection of 32 PiecewiseGrid objects: one for each piece in the game. Each PiecewiseGrid consists of a probability distribution that is represented as a two-dimensional array: a probability for each square on the board that indicates the agent's belief that the piece is in that particular square. The actual probability grid is the principal element in a PiecewiseGrid object, but there are also a few other pieces of information to help keep track of a piece's state. There are Boolean variables to indicate if a piece has been captured or is a promoted pawn. There is a list of other pieces (the capturedList) that is populated when an enemy piece is marked as captured. The list can be used to correct the game state if it is later determined that marking the piece as captured was an incorrect conclusion. For each pawn, there is a list of columns that the pawn might possibly be in.

One of the most common operations involving the PiecewiseGrids is to generate a possible board state. Most of the time, the exact board state is unknown, but the PiecewiseGrids represent the agent's beliefs about what the board might look like. To generate a possible board, each piece is placed onto the board, one after the other, each into a square based on its own probability distribution. If two pieces happen to be randomly placed into the same square, the second piece that would be placed there is simply placed in a different square instead. This skews the probabilities for any piece that gets bumped from its initially chosen square, but it is a quick way to generate a potential board without maintaining a massive collection of possible boards. When generating a board, the pieces are placed onto the board in order of importance, beginning with the kings and queens and ending with the pawns. In that way, if any piece suffers from skewed probabilities, it is less likely to be one of the more important pieces.

2.2 Updating the PiecewiseGrid

At the start of the game, the location of every piece is known with 100% certainty. Throughout the game, a player always knows the locations of their own pieces. As a consequence, the 16 probability grids for a player's own pieces always have the same format: 1.0 for the square where the piece is located and 0.0 for the other 63 squares. (If the piece has been captured, all 64 squares are 0.0.) The probability grids for the opponent's pieces are not so simple, and they must be updated regularly.

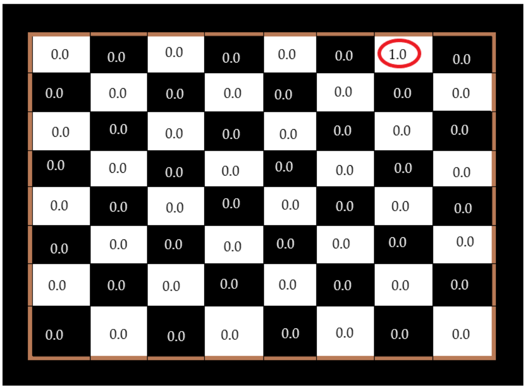
There are three points at which the game state is updated: after the opponent's move action, after a player's own sense action, and after a player's own move action.

After the opponent's move action, there was either a capture or not. If there was a capture, the agent will make a guess about which enemy piece performed the capture. If there was no capture, the agent makes a guess about what move the enemy made.

To guess which enemy piece made a capture, a board is first generated that contains only the pieces where the agent is fairly certain of the location (i.e. probability > 0.99). This includes all of the player's own pieces and some of the enemy pieces. After these pieces are "locked" in, it is then determined, based on each piece's probability grid, how likely it is that each enemy piece (including those not locked in) would be able to attack the square where the capture occurred. The pieces that are locked in may block certain pieces from attacking the square in question, eliminating those pieces from consideration and allowing the agent to better zero in on the actual attacking piece. Ultimately, the exact piece might not be determined. In either case, the probability grids are updated under the assumption that the probability a particular piece was the attacker is proportional to the probability that the piece was in a position to attack. Naturally, if only one piece had a non-zero probability of being in a position to attack, then the agent is sure about which piece performed the capture.

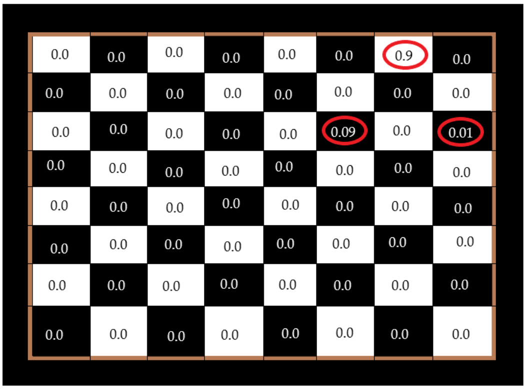
For each enemy pawn a list of possible columns is maintained. A capture is the only way that a pawn can change columns, so whenever the enemy captures a piece, the pawns possible-columns lists are updated. For example, a pawn that begins in column A is known to still be in column A as long as no capture has occurred in column B.

If no capture took place on the opponent's turn, then updating the game state after an opponent's move takes a very different approach. Using the probability grids, a number of possible boards are generated. Each board is analyzed using Stockfish. The best move on each board according to Stockfish is noted. Other good moves on each board according to Stockfish are also noted as good moves, and any moves that might put the king in check or pin a piece are also noted as good moves. (Those moves get special attention because in RBC, putting the king in check and pinning a piece are stronger than they are in regular chess. Those moves can lead to an immediate win if the opponent does not sense them.) Moves that are not noted as the "best" move or a "good" move are still noted as "other" possible moves. The agent places all of the possible moves into one of those three categories, and the probability grids are updated with the assumption that the opponent was most likely to have taken one of the "best" moves, less likely to have taken one of the "good" moves, and very unlikely to have taken one of the "other" moves. As each move is processed, the probability that a piece is in the move's starting square is decreased and the probability that the piece is in the move's ending square is increased by an equal amount. Figure 1 gives a simple example, where the two possible enemy knight moves are processed, and one of the moves is deemed much more likely than the other.



0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 1: An enemy knight's probability grid before an opponent's move



0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.09	0.0	0.01
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 2: An enemy knight's probability grid after an opponent's move. The agent thinks the knight probably didn't move, but if the knight did move, it probably moved toward the center of the board.

After a player's own sense action, more information is known about what the opponent did than was known immediately after the opponent's turn. Many potential moves that were processed after the opponent's turn may be known to be impossible after the sense action. The agent rolls back all impossible moves.

If a square that was known to be empty now contains an enemy piece, any move not involving that piece is impossible. If the agent was sure of an enemy piece's location and that square is now empty, then any move not involving that piece is impossible. Similarly, if the piece is still in the same location, then any move involving the piece is impossible. This allows the agent to narrow down the opponent's possible move to just a small set of viable candidates, or possibly even to identify the move exactly. After rolling back any moves that are deemed impossible, the probabilities of the remaining viable candidates are proportionally increased and updated.

The reader may wonder why there is a two-step process to determining the opponent's move. After an opponent's move, why not wait until after the agent's own sense action to guess what the move was? By first processing a larger number of potential moves immediately after the opponent's action, more information is available for the agent to decide where to sense. Then, after the sense action, the information from the sense action can be used to better determine which move the opponent actually took.

After a player's own sense action, the probability grids are also updated for the 9 squares that are sensed. (This is in addition to refining the guess about the opponent's move, as described above.) For any square that contains a piece, the corresponding piece's grid is updated to 1.0 for that square and 0.0 for every other square, since the piece's location is known with 100% certainty. If there are multiples of that type of piece still on the board (e.g. if a rook is seen but it is not known which rook it is), the best guess is assumed to be the correct piece. Every piece's probability grid is zeroed out for any squares where it is known that they are not located. Whenever a probability is zeroed out, the probabilities for that piece being in other squares are proportionally adjusted upward to so that each piece's grid remains a valid probability distribution. If the agent sees an enemy piece in the sense result but thought that all enemy pieces of that type were already captured, then it considers two possibilities. First, the agent considers that perhaps it was wrong in an earlier guess about which enemy piece was captured. If that possibility is detected, a correction referred to as a "resurrection" is issued. Second, if no resurrection is possible then the agent considers that perhaps an enemy pawn was promoted.

The third time that the agent updates the game state is after the agent makes its own move. If no capture occurred, the update is simple: change the moved piece's probability grid and zero out the destination square in all other pieces' probability grids. If a capture did occur, the agent knows where the capture occurred but is not told which enemy piece was captured. In most cases, the capture happens because the agent knew where the enemy piece was and intentionally captured it. However, there is almost always at least a small chance of being wrong, and sometimes the capture is unexpected. Whenever an enemy piece is captured, the agent makes its best guess about which piece was captured, and updates that piece's probability grid by zeroing out every square and marking it as captured. The agent also creates a list, known as the capturedList, of all other enemy pieces that had a non-zero probability of being in the square where the capture occurred.

The probability grids are merely the agent's best guesses about the game state. They are almost never completely accurate. Notably, it is possible for a grid to indicate a 1.0 probability for a particular square when the enemy piece is not actually in the square. Similarly, it is possible for an enemy piece to show up in a square even though the probability grid indicates a 0.0 probability. This can happen because not every possible board is investigated and not every possible move is accounted for. When the agent generates possible boards to consider, the actual, ground truth board might not be (and in fact probably isn't) actually generated. As a result, some moves that are possible on the actual, ground truth board might never be seen, considered, or taken into account in the probability grids. This can lead to the agent truly being caught off guard.

For example, the agent might end up capturing a piece in a square where the probability grid for every single enemy piece indicates a 0.0 probability of being in the square. In this case, the agent has no idea what piece was just captured, but the agent will try to guess anyway. If there is a pawn that is not captured and could possibly be in the given column, then the agent marks such a pawn captured. That keeps the count of defeated enemy pieces accurate. The agent also adds almost every enemy piece to the capturedList for that pawn, so that if the guess proves to be incorrect the agent can roll back and try to correct the error.

2.3 Choosing the Move

When deciding where to move, the agent generates a number of possible boards based on the probability grids. The number of boards that are generated depends on the amount of uncertainty in the probability grids and the amount of time remaining. (In a standard game, each player has a total of 15 minutes for the whole game.) For example, if the agent knows where every piece is located then only one board is considered. If the agent knows where almost every piece is located, then only a few boards are considered. If there is a greater degree of uncertainty, more boards are considered.

The agent generates a number of boards and for each board, it uses Stockfish to determine the best move on that board. After considering those boards, the result is a set of candidate moves. All of the candidate moves are then evaluated on a new set of generated boards. Stockfish gives every candidate move a numerical score on each of the new boards, and the scores are tallied.

The move that has the highest total score is chosen as the move to take.

2.4 Choosing Where to Sense

When choosing where to sense, the agent chooses the area of the board with the greatest uncertainty.

Each piece is given an uncertainty score for each square. As a first step, the uncertainty score is a number from 0 to 0.5 that indicates how certain the agent is about whether the piece is in the square. If the piece's probability grid contains a 0.0 or 1.0 for that square, then the agent is certain about whether or not the piece is in the square and the uncertainty score for that piece/square combination is 0.0. If the probability grid contains a 0.5 for the square in question, then the agent has no idea whether that piece is in the given square. That indicates a great deal of uncertainty about whether that piece is in the square, and the uncertainty score for that piece/square combination is 0.5. The uncertainty score for a piece/square combination is the absolute value of 0.5 minus the probability that the piece is in the square.

That describes the basic uncertainty score, but then several adjustments are made to the score. If the piece can attack the agent's king from the given square, then the uncertainty score is increased. If the piece pins one of the agent's pieces from the given square, then the uncertainty score is increased. If the piece cannot attack any of the agent's pieces from the given square, then the uncertainty score is greatly reduced. If the piece is a pawn that is near promotion, the uncertainty score is increased.

After calculating the uncertainty scores for every piece/square combination, an uncertainty score for each individual square is calculated. To calculate the uncertainty score for a square, the agent considers every piece in combination with that square, and the uncertainty score for the square is the maximum uncertainty score for any of those piece/square combinations.

The agent will choose to sense the square with the maximum uncertainty score, but it does not necessarily sense that square directly. Instead, it chooses the 3-by-3 block of squares that contains the chosen square while maximizing the sum of the uncertainty scores of all 9 squares that will be sensed.

3 Tournament Results

The agent described here took second place in the Fall 2019 NeurIPS RBC tournament sponsored by Johns Hopkins University Applied Physics Laboratory. There were 22 entries in the tournament, and each entrant played every other entrant 24 times (12 as black and 12 as white). The agent had a record of 11-13 against the eventual winner and 10-14 against the fifth place finisher, but other than that it had a winning record against every other entrant, and at least 20 wins against every finisher outside the top 5.

4 Opportunities for Improvement

Some losses in the tournament came because the agent lost track of a piece. For example, in one game a knight moved forward from its starting position and aggressively went for (and reached) the agent's king. Opponent moves that were terrible chess moves but good RBC moves (like moving a knight aggressively toward the king) were less likely to be sensed by the agent. Stockfish is an engine that plays regular chess. The agent uses Stockfish to both decide its own moves and predict the opponent's moves. If the agent used an engine that was tuned specifically for RBC to decide moves and predict opponent moves, it could do better.

There are a large number of constants embedded within the agent. Tuning of those constants could improve the performance of the agent. Here are just a few of the constants that could be used for tuning purposes:

- When choosing a move, how many boards should be considered? What about when predicting an opponent's move?
- When choosing a move, how long should Stockfish think about each board? What about when predicting an opponent's move?
- When predicting an opponent's move, how many "good" moves should Stockfish return?
- When predicting an opponent's move, what percentage of the time should the agent assume the opponent is taking one of the "best" moves versus a "good" move versus all the other moves?
- When evaluating a move, bonus or penalty points are granted for capturing the opposing king, putting the opposing king in check, moving the agent's own king into check, and moving a piece away from allied pieces (so that it is harder for the opponent to sense the whole board). How many bonus points are each of these situations worth? Should the bonuses differ when moving versus predicting the opponent's moves?
- When evaluating the uncertainty of a square, a bonus is given if the king could be in check or if a piece is pinned. A penalty is given if the piece would not be able to make any attacks from the given square. How much should the bonuses and penalty be?

Reconnaissance Blind Chess is a new chess variant that has been proposed as a testbed for AI and machine learning research that requires agents to deal with uncertainty. As one of the top finishers in the first worldwide RBC tournament, the approaches taken by this agent offer a promising foundation for future RBC agents as the research community explores approaches to dealing with uncertainty in this context.

References

[1] Jared Markowitz, Ryan W Gardner, and Ashley J Llorens. On the complexity of reconnaissance blind chess. *arXiv preprint arXiv:1811.03119*, 2018.

[2] Andrew J Newman, Casey L Richardson, Sean M Kain, Paul G Stankiewicz, Paul R Guseman, Blake A Schreurs, and Jeffrey A Dunne. Reconnaissance blind multi-chess: an experimentation platform for isr sensor fusion and resource management. In *Signal Processing, Sensor/Information Fusion, and Target Recognition XXV*, volume 9842, page 984209. International Society for Optics and Photonics, 2016.

[3] Tord Romstad, Marco Costalba, and Joona Kiiski. Stockfish chess. http://www.stockfishchess.org.

Real-World Assignments at Scale to Reinforce the Importance of Algorithms and Complexity[*]

Jason Strahler[1], *Matthew Mcquaigue*[1], *Alec Goncharow*[1]
David Burlinson[1], *Kalpathi Subramanian*[1]
Erik Saule[1], *Jamie Payton*[2]
[1]*Computer Science*
UNC Charlotte, Charlotte, NC 28223
{jstrahl1, mmcquaig, agoncha1, dburlins, krs, esaule}@uncc.edu
[2]*Computer and Information Sciences*
Temple University, Philadelphia, PA 19122
payton@temple.edu

Abstract

Computer Science students in algorithm courses often drop out and feel that what they are learning is disconnected from real life programming. Instructors, on the other hand, feel that algorithmic content is foundational for the long term development of students. The disconnect seems to stem from students not perceiving the importance of algorithmic paradigms, and how they impact performance in applications.

We present the point of view that by solving real-world problems where algorithmic paradigms and complexity matter, students will become more engaged with the course and appreciate its importance. Our approach relies on a lean educational framework that provides simplified access to real life datasets and benchmarking features. The assignments we present are all scaffolded, and easily integrated into most algorithms courses. Feedback from using some of the assignments in various courses is presented to argue for the validity of the approach.

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

Algorithms courses are generally a cornerstone of computer science bachelor's degree programs. They introduce concepts critical to CS students' theoretical foundations, and are a significant part of the core curriculum recommended by ACM in their 2013 CS guidelines [9]. Meanwhile, students often see these classes as being overly theoretical; they express frustration that calculating complexity, proving correctness, and studying obscure paradigms are the focus of the entire class. While some mathematically-minded students will appreciate the content, most perceive the class as unimportant. As such, the failure rate in algorithms classes is often high, and they are known in many universities for being the weed-out class.

Runtime efficiency of algorithms is a hard topic to teach. Gal-Ezer and Zur showed that students have many misconceptions about runtime efficiency, and wrote "Concepts like big-O..., are known to be difficult to conceive and difficult to teach and learn" [6]. Misconceptions in algorithms courses are common and have received some academic attention [4]. While some efforts were made to design tools to help students understand algorithms [11, 7], little effort has focused on getting students engaged in the topic.

We present extensions to the BRIDGES framework which are designed to help engage students with the content in algorithms courses. In particular, BRIDGES provides access to real-world data which can be processed by algorithms to solve real-life problems. Algorithms can be visualized in multiple ways to improve students' understanding of how a particular algorithm works. Moreover, our framework provides larger scale instances which enable performing run-time benchmarking of algorithms. These different features enable students to develop a keener understanding of why algorithmic content is relevant to them and their career. The BRIDGES framework and scaffolded assignments are available online (http://bridgesuncc.github.io/).

2 Related Works

What Makes Students Engaged. Strategies for engagement seem to come from two complementary approaches. **Teaching style engagement strategies** focuses on how a course is taught and managed. Active learning techniques have become popular in recent years to promote student engagement and can include any combination of lab-based instruction, flipped classrooms, gamification, peer-learning, and use of multimedia content [8]. **Content based engagement strategies** present the topics of the course using learning materials that capture the interest of the students. This is a common thread in the popular assignment repositories such as Nifty Assignments [14], EngageCSEdu [12], and game-themed assignments [3]. Real-world and large datasets in course projects have been demonstrated to successfully engage students [1], in contrast to tiny contrived examples.

What Algorithms Courses Typically Look Like. Looking at topics and learning outcomes in curriculum guidelines [9], algorithms textbooks [2], and concept inventories of algorithms courses [4] paints a picture of a typical algorithms course. Algorithms courses typically start with a discussion of runtime estimations, using complexity notations like Big Oh and Big Theta, followed by methods for proving algorithm correctness, and computing runtime complexity. Courses may look at advanced data structures such as trees, graphs, or hash tables to define problems, or as a way to solve problems in the class. Algorithm design techniques such as brute force, divide and conquer, greedy algorithms and dynamic programming are introduced to solve a variety of problems. Advanced courses may contain discussions of calculability, NP-Completeness and methods such as Branch and Bound methods, and approximation algorithms.

Existing Educational Efforts in Algorithms Courses have been aimed at visualizing what a data structure or algorithm looks like [13, 10]. These efforts have focused on creating a visually interactive way to show and teach algorithms to students, and was shown to be effective at learning algorithmic concepts [5]. There have also been efforts to bring real world map data into algorithm courses [15], for use in sequential search, graph traversal, Dijkstra's algorithm, and convex hulls. This effort shows an interest in bringing real world data into algorithm courses instead of using synthetic or contrived data.

3 The BRIDGES system

The BRIDGES system enables assignments relevant to the goals of introductory CS courses, including algorithms courses. The system provides bindings for Java, C++, and Python based on a commonly used object hierarchy. Output from assignments are highly visual and can easily be shared with friends and family.

Scalable Dataset Access. A simple API provides access to external data sources such as USGS Earthquake data, Wiki Data, IMDB actor/movies, Genius' Song Lyrics, and OpenStreet Maps. Usually a single function call returns a set of easy to understand objects. Assignments that leverage these interesting datasets seem more real and relevant. When possible, these data are accessed live. BRIDGES plays the intermediary, dealing with credential issues, access policies, etc. Because the data is live and real, the datasets can be as large as

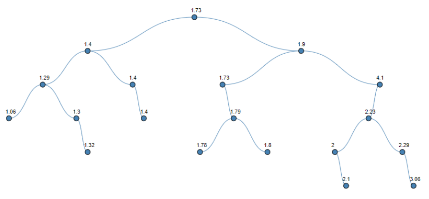
Assignment	Topics	Engagement
Plotting Complexity	Order of Growth	Visual Output, use own machine spec
Sorting Implementation	Order of Growth, Divide/Conquer, Decrease/Conquer, Heaps	Visual Output, Real Data
Book Distance	Order of Growth, Trees, Hash Maps	Visual Output, Real Data Analysis, Interest:literature
Mountain Path	Order of Growth, Optimization, Dynamic Programming, Shortest Path, Greedy Algorithms, Problem Modeling	Visual Output, Real Data, Real Problem, Interest:hiking
Routing in a City	Order of Growth, Shortest Path	Visual Output, Real Data, Real Problem, Choose own city, Interest:social
Hollywood Analysis	Order of Growth, Graphs, BFS, Graph Construction	Visual Output, Real Data Analysis, Interest:entertainment, Fun

Table 1: A set of assignments using real data, real problems to teach algorithms.
they need to be. One could access a map of every street in the United States from Open Street Map or get information on every single movie ever released. To minimize network transfers and computational costs, the data is cached, both within the BRIDGES infrastructure and on the student's machine. The expectation is that having data at that scale will let students explore different problems, see practical applications as part of their studies, and realize that problems can be at large scale without appearing to be made up.

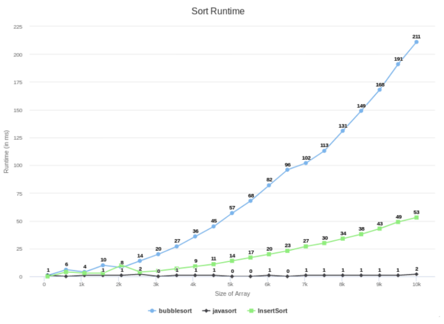
Performance and Benchmarking Features. Algorithm teaching tools typically lack elements to understand questions related to the runtime of algorithms. BRIDGES provides features to plot performance charts. Elaborate visualizations of complexity [16] have been designed for trained algorithm experts, but BRIDGES uses classic runtime plots where algorithms are associated with pairs $(size, time)$ displayed using a classic line chart (See Figure **??** for a sample output). Secondly, BRIDGES enables automatic benchmarking of particular problems. Take sorting of an array as an example. The student-implemented sorting function is passed to a benchmarking object that will run the algorithm at different sizes to extract performance information. When possible, the benchmarks are run using real-world data, to avoid students' feeling that the problems have been scaled up for their own sake.

4 A Set of Engaging and Scalable Algorithm Assignments

Next we present a set of assignments leveraging BRIDGES which are appropriate for an algorithms course. Table 1 presents concisely the assignments, the topics they map to, and their engagement characteristics. The assignments cover most topics in an algorithm class, often use real data, perform a real analysis, or solve a real life problem. The assignments are linked to areas of interest usually popular with students. Assignments scaffolded for C++, Java, and Python are accessible online (http://bridgesuncc.github.io/newassignments.html).



(a) Binary Search Trees can be used to sort



(b) Timing experiment

Figure 1: Sorting in BRIDGES

The Classic Plotting of Complexity consists in plotting the cost of a few algorithms given their precise instruction count for given machines and shows that an algorithm with higher complexity running on a much faster machine will still be slower than an algorithm with a better complexity running on a smaller machine. For instance, the runtime an algorithm using $10^4 n$ operations and an other one using $5*10^4 n$ operations on a machine at 1MHz would be much faster than a machine at 100MHz running an algorithm taking $10^2 n^2$ operations. Engagement comes from the visual output, which is easy to understand and from personalizing the assignment, by using a student's own machine.

The Classic Sorting Assignment consists in implementing classic sorting algorithms. Insertion sort and selection sort are quadratic decrease-and-conquer algorithms often covered when talking about arrays or correctness. While Merge sort and Quick sort are often used to show an $\Theta(n \log n)$ algorithm using a divide-and-conquer approach, sorting by leveraging a tree data structure such as a Heap or a Binary Search Tree can also be used.

BRIDGES can be used to understand how the sorting happens across different sorting algorithms by visualizing the current state of the algorithm. Figure **??** shows a Binary Search Tree that can be then in-order traversed to extract a sorted array. BRIDGES provides unit testing and benchmarking of the algorithms. Figure **??** shows the performance of insertion sort and bubble sort compared to Java's standard sorting function.
Engagement comes primarily from the visualization of the algorithm and of the runtimes. Real data can be used for what is being sorted: Figure **??**, for instance, shows sorting using the magnitude of recent Earthquakes. But without a more precise scaffold, it can appear artificial to students.

Computing Book Distance is inspired by Natural Language Processing. The idea is to compare how close books are using a bag-of-word model. For a particular book, one can compute how many times each word appears and normalize that count to form a vector representation of the book (maybe the word "dog" appears 1% of the time.) These frequency vectors can be leveraged to compare books, for instance, by using the L1 distance or a cosine similarity function. This analysis will highlight that books from a particular author are more similar than books from different authors.

The core of the assignment is the implementation of a Dictionary which can be done using different data structures: array, linked list, binary search tree or a hash map. The application in the assignment is counting words and computing distance between sparse vectors. BRIDGES provides access to data for this assignment through Project Gutenberg. Using small Shakespearean poems is very good for debugging. But computing the distance between larger books will drive home the importance of complexity. A comparison of all the works of Shakespeare against all the works of Mark Twain will take hours if one uses a $\Theta(n)$ implementation of Dictionary such as an unsorted array while it will take only seconds using an $O(1)$ hash maps.

Engagment. The Dictionary can be visualized and debugged using BRIDGES. The application is real: this type of analysis was conducted to identify who wrote the Federalist Papers. Students with interests in literature will appreciate this assignment and plug in their favorite classic authors.

Optimizing Path through a Mountain. The mountain path assignment first appeared as a Nifty assignment [14]. Given an elevation map (image) the task is to find the lowest cost path, from one end of the image to the other, where the cost is defined as the sum of difference in elevation between consecutive pixels in the path. A simple greedy approach is used to make local decisions and the selected path of pixels is drawn in color (shown in Figure 2).

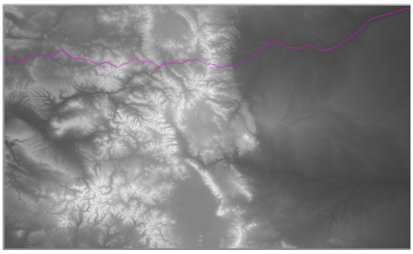


Figure 2: Path of Least Elevation (Greedy)

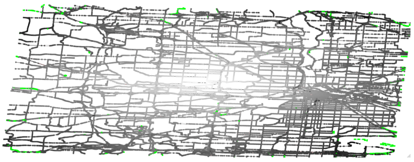


Figure 3: Map of Minneapolis, MN colored by distances to the center of the map.

This problem from Nifty is particularly good to teach algorithms for optimization problems. The algorithm is a greedy heuristic. However, one can easily build a variant where you always have to go right, but are allowed to choose whether to go up-right, straight-right, or down-right by looking at the

entire map. This variant can be solved using dynamic programming or a brute force method. With a large map, the difference in computation time between the greedy heuristic, and the optimal algorithm will be obvious and enable discussions of time-quality tradeoff. A second variant is to go from a pixel to any of the 8 neighbors and the problem becomes a classic shortest path problem.
Engagement. The output is visual, and addresses a real problem using real data. It will echo well in students who love hiking. History-buffs may be interested in figuring out where Hannibal should have crossed the Alps, or whether Xerxes the Great had to fight the Greek army at Thermopylae. Finally, BRIDGES lets students choose the elevation map they will use in the assignment, for instance, their own campus, city, or a favorite hiking area.

Scalable Routing through a City. Implementing Dijkstra's algorithm with best case complexity is difficult because it relies on a Fibonacci Heap [2]. Often implementations seen in algorithms courses use a regular Binary Heap which has a higher complexity. This is not a major problem when the map is small as the difference is hard to notice. BRIDGES provides access to Open Street Map data and enables students to access maps of the continental US for any latitude/longitude bounding box at different resolutions (the graph of Minneapolis is shown in Figure 3). This enables BRIDGES to benchmark automatically Dijkstra's implementation.
Engagement. Students get to choose the map they will use, can relate to the need for routing in a city, and get the feeling they are solving a real problem.

Analysis of Hollywood Movies. Computing the Bacon Number of an actor consists in figuring out how many movies away an actor is from Kevin Bacon by only going from actor to one of his/her movie and from a movie to one of the actors that played in it. Listing the actors and movies in the chain is a game known as the "Six Degrees of Kevin Bacon". Provided a graph composed of movies and actor with edges if the actor played in the movie, computing the Bacon number of an actor is achieved with a BFS traversal.

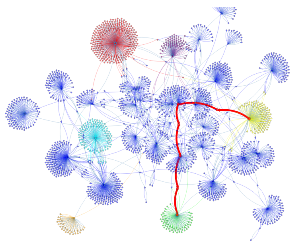


Figure 4: Bacon Number

BRIDGES enables accessing a toy actor-movie graph from IMDB with about 2000 edges. This graph can be styled to highlight the shortest path between an actor and Kevin Bacon (see Figure 4).

It is also possible to access all the data of all English movies since the 1910s. The students will build graphs of about 1M edges and BRIDGES enables to easily query particular time intervals. This makes the dataset very versatile, permitting students to study who was the most central actor for a

particular decade; this calculation requires computing a BFS from each actor in the dataset. While a single BFS computation takes about 0.5 sec. on a 1M edge graph, calculating BFS from all vertices takes on the order of a day, thus emphasizing that calculation time compounds quickly at real world scales.
Engagement. The analysis that students perform is real and a big data problem, and a dataset students are often interested in is explored visually.

5 Results: Student Reactions

We deployed several of the above projects in courses on algorithms, data structures and object oriented systems. We obtained student feedback on the projects through reflection and project surveys, performed after each project. The reflection survey gives students the chance to describe their learning experiences and specifically what they liked and did not liked about the assignment. Other questions focused on engagement, completion time, preparedness, assignment difficulty, and if the assignment increased their interest in computing.

Book Distance Project. The project was assigned in a data structures class in Fall 15 as a four-part project. Student started by writing the book distance comparison using a provided Dictionary implementation using unsorted arrays, then they implemented a Dictionary object using sorted arrays, binary search trees, and hash maps. They computed distance between books of different sizes. No formal reflection or engagement quiz was given. The following comments come from the notes of the instructor (an author of this paper).

Students initially felt that the implementing the application was difficult, but eventually appreciated seeing distances between books. Computing the distances using sorted arrays was very slow and students computed only some of the distances. Using the BST, students were able to compute all the pairwise comparisons. Many students were surprised at the speed of hash maps, and commented on how they "killed their laptop" on computation that was unnecessary with better data structures.

Mountain Path. We only considered the greedy algorithm. BRIDGES was used to display the input elevation image and the final path. The project was given in Fall 18, Spring 19 and Spring 20 in an object oriented programming course. Students found the project highly engaging (96%, 100%,85%) and difficult (85%, 74%, 78%). Short answer responses also indicated difficulties with programming concepts, clarity of instructions, and also satisfaction with the assignment challenges. Students really seemed to enjoy the project, with remarks such as 'excellent practical example of the greedy algorithm', 'very challenging and learned quite a bit', 'at first ... intimidated, ... after thoroughly reading...

I felt a bit more confident', 'liked the assignment challenged my programming ability'

Bacon Number. The project was given to an algorithms course in Spring 16 and Fall 17 to an algorithms course. Only the first part of the project was given to the students, to implement the Bacon Number project on the 2000 node graph. Students found the project to be difficult (57% vs. 30%, 45% vs. 9%) but increased their interest in computing (51% vs 30%), and felt it was relevant to their career goals (48% vs. 16%, 54% vs. 18%)

Students were excited about the visual output "This is the first time in ALL of comp sci, that we could actually visually see the data structures", interest in lowering runtime: "an individual [may be] required to work with a large data structure and ... to effectively traverse [it] in a reasonable computation time", liked working with graphs, "graphs are more fun/interesting" and found them relevant, "Bfs is an extremely useful and popular algorithm and graphs are used frequently in computer science and tech industry".

6 Conclusion

This paper presents strategies to engage students with the content of typical algorithms courses. The main strategy is to assign course projects that are or look like real-life problems and rooted in domains students care about and use visualization. The data used in the projects are extracted from live sources, grounding the projects in reality. We presented 6 assignments/projects that cover most of the content of a typical algorithms course. Three of the projects were assigned to students in various courses. The projects were deemed hard but were eventually perceived positively by the students. A threat to the validity of this work is that the projects were not all in a single algorithms course, and one probably one would not assign all of them in such a course. We will address that in the future by performing such an intervention and conducting formal studies.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant no. DUE-1726809 and CCF-1652442.

References

[1] David Burlinson, Mihai Mehedint, Chris Grafer, Kalpathi Subramanian, Jamie Payton, Paula Goolkasian, Michael Youngblood, and Robert Kosara. BRIDGES:

A system to enable creation of engaging data structures assignments with real-world data and visualizations. In *Proc. ACM SIGCSE 2016*, pages 18–23, 2016.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, third edition edition, 2009.

[3] Peter Drake and Kelvin Sung. Teaching introductory programming with popular board games. In *Proc. of ACM SIGCSE*, pages 619–624, 2011.

[4] Mohammed F. Farghally, Kyu Han Koh, Jeremy V. Ernst, and Clifford A. Shaffer. Towards a concept inventory for algorithm analysis topics. In *Proc. SIGCSE*, pages 207–212, 2017.

[5] Mohammed F. Farghally, Kyu Han Koh, Hossameldin Shahin, and Clifford A. Shaffer. Evaluating the effectiveness of algorithm analysis visualizations. In *Proc. SIGCSE*, pages 201–206, 2017.

[6] Judith Gal-Ezer and Ela Zur. The efficiency of algorithms—misconceptions. *Computers and Education*, 42(3):215 – 226, 2004.

[7] Michael T. Goodrich and Roberto Tamassia. Teaching the analysis of algorithms with visual proofs. *SIGCSE Bull.*, 30(1):207–211, March 1998.

[8] Mark Guzdial. A media computation course for non-majors. In *Proceedings of the ITICSE 2003*, pages 104–108, 2003.

[9] Joint Taskforce on ACM Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM/IEEE Computer Society, 2013.

[10] Jeff Lucas, Thomas L. Naps, and Guido Rößling. Visualgraph: A graph class designed for both undergraduate students and educators. *SIGCSE Bull.*, 35(1):167–171, January 2003.

[11] Joan M. Lucas. Illustrating the interaction of algorithms and data structures using the matching problem. In *Proc. SIGCSE*, pages 247–252, 2015.

[12] Alvaro Monge, Beth A. Quinn, and Cameron L. Fadjo. EngageCSEdu: CS1 and CS2 materials for engaging and retaining undergraduate CS students. In *Proc. of ACM SIGCSE*, pages 271–271, 2015.

[13] Thomas L. Naps, James R. Eagan, and Laura L. Norton. JHAVÉ - an environment to actively engage students in web-based algorithm visualizations. In *Proc. SIGCSE*, pages 109–113, 2000.

[14] Nick Parlante. Nifty assignments, 2018.

[15] James D. Teresco, Razieh Fathi, Lukasz Ziarek, MariaRose Bamundo, Arjol Pengu, and Clarice F. Tarbay. Map-based algorithm visualization with METAL highway data. In *Proc. SIGCSE*, pages 550–555, 2018.

[16] Jeyarajan Thiyagalingam, Simon Walton, Brian Duffy, Anne Trefethen, and Min Chen. Complexity plots. In *Proc. EuroVis*, pages 111–120, 2013.

Activity Based Learning for Cloud Computing[*]

Michalina Hendon and Loreen Powell
Information Technology, Analytics, and Business Education
Bloomsburg University
Bloomsburg, PA 17815
{mhendon, Lpowell}@bloomu.edu

Abstract

Cloud computing technology is relatively untapped as the world learns how to utilize the cloud's potential to provide greater efficiencies in hardware and software to meet and exceed business needs and goals. Organizations continue to explore the flexibility that cloud computing can provide. There is a growing need for employees to continue an organization's digital enterprise transformational strategies. Industry is looking for skilled students that are introduced or have been immersed in different specialties of the Information Technology (IT) field. Cloud computing sources skills from different areas of IT and Computer Science (CS). As such, educators and universities seek to provide a cloud computing curriculum. Literature, however, is lacking in hands-on pedagogical resources to enhance textbook theory. The purpose of this paper is to provide educators with a framework of applied hands-on tools and resources to enhance a student's experience in an introductory cloud computing course and feedback.

1 Introduction

There are many variations defining cloud computing. One of the most popular definitions by the National Institute of Standard and Technology (NIST) defines cloud computing as: "A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

(e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [12]. Simply stated, cloud computing, in general, describes the multitude of on-demand data and storage centers, as well as the computer resources and services available to organizations via the internet. Typically, there are three types of clouds; public, private, and hybrid [11], which provide back-end or front-end considerations. This paper focuses on public, back-end cloud computing.

Large public and private cloud providers, for example, Amazon's Web Services (AWS) [16] or Microsoft's Azure allow organizations to take advantage of provided services of the PaaS (Platform as a Service), SaaS (Software as a Service) IaaS (Infrastructure as a Service) and BaaS (Back-end as a Service). The service models that are available to the organizations allow for scalability, lower costing for on-demand hardware lessens infrastructure concerns, provides software with subscription, and more. However, the pay as you use model, which enables flexibility, is most enticing for an enterprise [8]. As organizations begin to take advantage of larger cloud providers like AWS, the provider can supply the model framework, but the enterprise/organization is still responsible for the security, development, authorization, and maintenance of the cloud. Hence, there's both a need and an opportunity to provide educational training to equip students with the necessary skills to compete in this high-demand market. Teaching cloud computing concepts covering the full-stack business solution requires introducing students to different full-stack layers.

Today, cloud computing is a vital factor in enterprise computing as more enterprise workloads move to the cloud [17]. Thus, organizations' need for skilled employees to handle the change in culture to digital enterprise architecture is rapidly growing. Chun, [3] states that information technology and computing students must possess cloud computing knowledge and skills to be properly prepared for the workforce. He argues that it is crucial for cloud computing to be incorporated into the curriculum.

Currently, there are several theoretical cloud computing texts and resources available for instructors. However, there are a limited number of applied/hands-on exercises for students. This paper seeks to add to the body of literature regarding applied cloud computing resources and activities. Specifically, we aim to provide recommended applied pedagogy cloud computing resources that can aid an instructor teaching an introductory cloud computing course. In this work, we believe that applied/hands-on exercises will help enhance the students' cloud computing knowledge and skills. This work has practical implications for higher education institutions, faculty, and computing degree programs by providing framework and resources that can help assist faculty with applied resources to increase cloud computing awareness within IT and computing disciplines. The remainder of this paper is structured as follows: background/review of the literature, methods, applied resources, and conclusion.

2 Background/Review

2.1 The Future of Cloud Employability

As the cloud is operated more frequently in personal and enterprise use, the need for skilled employees is growing. According to Forbes, in 2019, "50,248 cloud computing positions are available in the U.S. today available from 3,701 employers and 101,913 open positions worldwide today" [4].1]. Accordingly, the top cloud providers are providing lucrative opportunities in cloud computing, with a median salary of $125,097 for a cloud engineer position. According to Indeed [7] skills (depending on the cloud position), which include but are not limited to; knowledge of at least one programming language, networking, database management, application development, system development, and understanding in business modeling [3]. However, demands and uses of cloud technology have changed; the need to provide a specific course in cloud computing will aid in student employability after graduation.

Challenges and benefits from a pedagogical perspective, as it relates to providing students with theory in practice, can be thwarted with a lack of student introduction to coding, database management, network architecture before taking the cloud computing course. However, an introduction to how the various skills are combined to provide cloud service can have career implications for student's employ ability [7]. Educators attempt to match industry needs with the curriculum offered within their course [13].

In the past, students were offered an introduction to the cloud as an intermediate course offering. The intermediate course provided a limited introduction of the cloud's capabilities due to the time constraints of the course. As Mew and Money[9] found, with the phenomenon of the cloud being still new, the skill requirement may be self-serving and distorted as to what may be actually needed within an organization. Currently, the introduction of skills for development within a cloud computing course includes highlighted uses of; database management, infrastructure, security, and administration positions that have the opportunity for exploration[9]. The topics discussed in an introduction to the cloud course can provide the students with the direction of the potential concentration of study. Additionally, as the uses of cloud computing and the demand for skilled employees grow, an introduction to the theory of cloud computing allows students to find a direction in addition to established skills from other courses [3],[17],[2],[8],[6],[14].

2.2 Obstacles of Hands-on Application

The availability of direct learning resources in cloud computing sources can be complex With a multitude of providers and trial availability of sophisticated platforms, there is the accessibility to engage in theory with practice but not without obstacles. When discussing adding courses to the curriculum, several variables need to be considered. First, the availability of faculty to instruct the course can be an issue. As cloud computing is a newer technology, the availability of faculty that are educated in the technology can be a barrier to offering such a course. The consistent and swift speed of advancements in cloud technology may impede the learning curve to match organizational demands [1]. The skills of the students and when the course is offered in comparison to the student's course schedule may have an impact on basic knowledge of foundational networking, database, and security brought to thee course.

Finally, the cost of providing the students the ability to provide hands-on activities can be costly to the students. For example, if students would like to provision a server and utilize the sandbox tools offered by AWS the student is asked for a credit card to establish login credentials to initiate the sandbox even though it is provided by AWS Educate (which provides training for educational institutions) [16]. The trend of available training by cloud providers is usually based on the student's ability to providing a credit card authorization. An obstacle is faced when students can not provide credit card for registration. Faculty teaching the course may not be authorized to use university credit cards as an additional option. Additionally, the ability of instructors to secure funding or resources from the educational institution can impact providing students activity-based learning. At most public Universities, budgetary restraints are common [10]. Instructors are looking to implement training that is at minimal cost to the department as well as cost passed on to the students [10].

2.3 Implications for Curriculum Development

A principal course in cloud computing can cover a variety of topics. Cloud delivery and deployment models should be discussed within the course. Explanation of the communicating the culmination of the different skill sets to create, maintain, and develop both the virtualization of the cloud and the underpinning of the back-end with activity-based assignments can add to students understanding of theory. Providing an environment that allows students not only to discuss how cloud theory can be performed on a theoretical basis, but students can also visualize and apply the back-end elements [15]. Suggestions for the inclusion of activity-based learning utilizing free training and sandboxes from cloud providers can be implemented.

3 Method

The goal of this paper is to demonstrate how an applied framework is to provide an instructor implementing activity-based assignments into a fundamental cloud course resources to align with theoretical concepts introduced within the course. The following framework was utilized within the Cloud Computing course. The course was offered in the Fall 2019 semester as special topic elective within an accelerated IT degree program. Although the course was a condensed course of 6 weeks, the following framework is mapped to a 16 week semester to provide a broader perspective of activities available. The majority of students enrolled in the course as a point of interest with little to no understanding of cloud technology and little to no experience with programming languages. With the varying degrees of education, the need to convey the fundamental theory in cloud computing was a visible component of the course. The students required reading included the text *Cloud Computing: Concepts, Technology, and Architecture*[5], to provide a foundation of theory development and reference. The text was dated in sections; however, the videos in the activities provided current insight into theoretical application. An additional outcome of the course activities not only provide the students with hands-on activities but supply certificates of completion from the service providers.

4 Applied Cloud Computing Framework and Resources

The framework was first created from a break down of activities listed by major cloud providers low to no-cost training as well as InLearning resources (the university subscribes to the service).

4.1 Resources

Resources provided in Table 1 demonstrate a list of hands-on activities that can be applied within a fundamental cloud course. Table 1 provides a shortlist of activities by the provider and refers to the certification of completion that students can achieve. The availability of free training can enhance the student's interest in the subject and increase understanding of cloud computing concepts. The list is then broken down to match a suggested stream of content in sequential learning order, building on established concepts.

4.2 Applied Cloud Computing Framework

As noted in Figure 1, the course is broken into weekly sections (suggestion). When implemented in topical sections, the activities will build upon theoretical strength. Introducing students to available free training with a learning path will provide the students with tools to develop a role in cloud computing and cloud certification. It was found within the introduction course that the activities enhanced the learning method by providing students with the application to support the class discussions regarding theoretical learning.

The flow of the course begins with an introduction to cloud computing. Within the first two weeks, the dissemination of a broad theory of cloud computing is vital to the base of conceptual understanding. The LinkedIn Learning videos and quizzes allowed students to further theories and test their knowledge. The next week's follow of the suggested sequence of topics. Infrastructure is suggested as the next topic to build upon the theory. The Infrastructure section allows for the introduction of providers and of the provider service models. Next virtualization will build upon the previous sections. Providing two weeks to focus on virtualization affords students the ability to use the knowledge to complete the activity of launching virtual machines in a cloud environment.

Once the virtualization theory and application are discussed, the networking portion of the course will demonstrate the connection of cloud services. As such, the AWS activity with an Introduction to CloudFront can provide students with learning more about content delivery. The Azure training introduces students to the options that are available through the Azure platform. The next topical area discusses storage. There is a multitude of activities available depending upon the concentration of service. To introduce students to storage concepts and demonstrate backup and restore, the different options available from providers(AWS and Azure) for storage, including Amazon's Elastic File Storage (EFS), and Elastic Block Storage (EBS).

Finally, the course ends on the topic of security. This section should only be accomplished when an individual understands the inter-working of the cloud. The activities under the security topic introduce students to management and governance through cloud platforms as well as support theory learning for best practices. The activities suggested are on the AWS platform and guide students through the creation of Identity and Access Management (IAM), as well as AWS security module. The security module allows for implementing roles, and access demonstrates the importance of the human factor of security and in different security-oriented services.

Feedback Generalized student feedback found the resources applicable to the theory learned within the text, but students could also see a learning path for future exploration and training. Signified by a certificate of completion or badges the student (after completion of the activity) the badges demonstrate growth, which can be applied to future or potential professions. For example, the InLearning Certificate of Completion can be linked to the students' LinkedIn account/virtual profile. Badges and Certificates from AWS and Azure can also be applied to a digital profile or included within a section of the re-

Provider	Offering	Certification	Assessment
InLearning	Introduction to Cloud Computing for IT Pros David Rivers https://www.linkedin.com/learning/introduction-to-cloud-computing-for-it-pros/next-steps?	InLearning Certificate of completion (LinkedIn Learning)	Video and Quiz
	Learning Cloud Computing: Core Concepts David Linthicum https://www.linkedin.com/learning/learning-cloud-computing-core-concepts-2/change-your-career-with-cloud-computing?	InLearning Certificate of completion (LinkedIn Learning)	Video and Quiz
	Building iOS Apps with AWS Mobile Bear Cahill https://www.linkedin.com/learning/building-ios-apps-with-aws-mobile/client-server-side-with-zero-coding?u=42462916	InLearning Certificate of completion (LinkedIn Learning)	Video and Quiz
AWS Services	AWS Cloud Practitioner Essentials (Second Edition): AWS Core Services AWS Storage Offerings https://www.aws.training/Details/eLearning?id=29700	AWS Certification of Completion	Video and Quiz
	Introduction to AWS Identity and Access Management (IAM) https://amazon.qwiklabs.com (free training)	Badge of Completion	Sandbox
	Security on AWS https://amazon.qwiklabs.com (free training)	Badge of Completion	Sandbox
	Launch a Linux Virtual Machine https://aws.amazon.com/getting-started/tutorials/launch-a-virtual-machine/ Introduction to Amazon CloudFront https://www.aws.training/Details/Video?id=15891	No Certificate	Sandbox
AZURE	Azure fundamentals Core Cloud Services - Introduction to Azure Core Cloud Services - Azure networking options Select a cloud deployment model Core Cloud Services - Azure data storage options Create a Windows virtual machine in Azure Manage apps in Power Apps Microsoft https://docs.microsoft.com/en-us/learn/paths/azure-fundamentals/	Badge of Completion	Sandbox

Table 1: Activities by Provider and Certification of Completion Awards

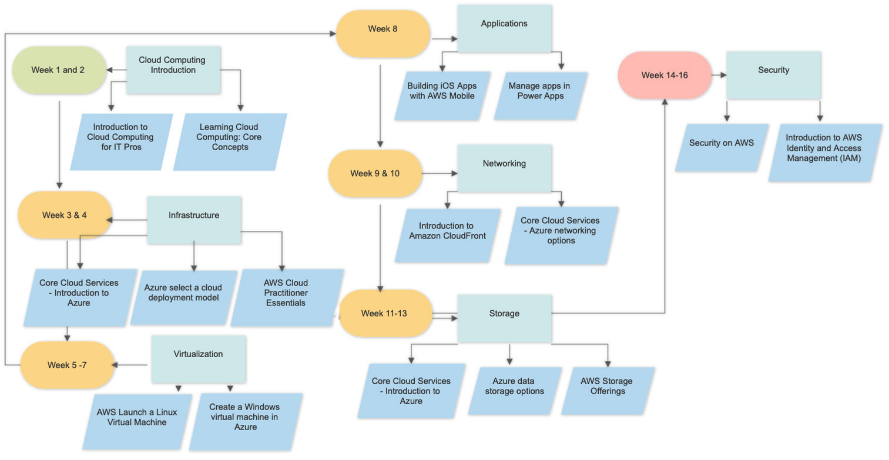

Week 1 and 2
Cloud Computing Introduction
Introduction to Cloud Computing for IT Pros
Learning Cloud Computing: Core Concepts
Week 3 & 4
Infrastructure
Core Cloud Services - Introduction to Azure
Azure select a cloud deployment model
AWS Cloud Practitioner Essentials
Week 5 -7
Virtualization
AWS Launch a Linux Virtual Machine
Create a Windows virtual machine in Azure
Week 8
Applications
Building iOS Apps with AWS Mobile
Manage apps in Power Apps
Week 9 & 10
Networking
Introduction to Amazon CloudFront
Core Cloud Services - Azure networking options
Week 11-13
Storage
Core Cloud Services - Introduction to Azure
Azure data storage options
AWS Storage Offerings
Week 14-16
Security
Security on AWS
Introduction to AWS Identity and Access Management (IAM)

Figure 1: Course Flow of Activities by Topic

sume. The advantage of using training that provides certifications or badges of completions can also offer the student with a sense of accomplishment and can encourage further training.

5 Conclusion

Cloud Computing is essential to computing science curriculum and is important to be included within the curriculum as organizations are expanding their digital enterprise transformational strategies. This paper provided pedological and theoretical concepts for instructors teaching or developing a cloud computing course. This paper also provides a foundation for instructors to utilize and include activity-based assignments in a cloud course.

While this paper adds to the body of literature, limitations due exist. Limitations include the larger cloud providers are the front runners that are supplying the availability of training. This paper is limited to specific cloud providers. Another limitation is the availability of the training over semesters as many trials are limited to a week or can extend for three months. A standard semester is usually four months, which is outside of the trial access. The trial also limits students from using the same software or access to resources in another course after the initial trial expires. Finally, the list of resources are suggestions and a selected list which can be expanded.

Future research should address these limitations and build additional pedological resources within this area. The principle of providing students the ability to increase skills and put theory in practice utilizing activity-based learning can provide students not only application skills but also future learning paths.

References

[1] The year ahead data will drive the enterprise 2019. *Database Trends Applications*, 32(6):4 – 10, 2018.

[2] L. Chen, Y. Liu, M. Gallagher, B. Pailthorpe, S. Sadiq, H. T. Shen, and X. Li. Introducing cloud computing topics in curricula. *Journal of Information Systems Education*, 23(3):315, 2012.

[3] W. Chun. Cloud computing and running your code on google cloud. *Journal of Computing Sciences in Colleges*, 34(5):81–81, 2019.

[4] L. Columbus. Where cloud computing jobs will be in 2019. https://www.forbes.com/sites/louiscolumbus/2018/11/27/where-cloud-computing-jobs-will-be-in-2019/#a5096276add5, 2018.

[5] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edition, 2013.

[6] Dahai Guo and Anna Koufakou. A comprehensive and hands-on undergraduate course on cloud computing. In *Proceedings of the ASEE Southeastern Section Conference*, 2018.

[7] Indeed. Cloud engineer salaries in the United States. https://www.indeed.com/salaries/cloud-engineer-Salaries, 2019.

[8] D. Irwin, P. Sharma, S. Shastri, and P. Shenoy. The financialization of cloud computing: Opportunities and challenges. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–11, July 2017.

[9] L. Mew, W. H Money, and S.C. Charleston. Cloud computing: Changing paradigms for information systems development service providers and practitioners. *Journal of Information Systems Applied Research*, 2018.

[10] Kathleen Masterson Michael Mitchell, Michael Leachman. A lost decade in higher education funding state cuts have driven up tuition and reduced quality, 2019.

[11] Microsoft. What is cloud computing? https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

[12] National Institute of Standards and Technology. The NIST Definition of Cloud Computing : Recommendations of the National Institute of Standards and Technology National Institute of Standards and Technology, url= https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[13] R Panko. Cloud education lags at universities: 4 professors' perspectives. https://clutch.co/cloud/resources/cloud-computing-education-2017, 2017.

[14] M Suhail Rehman, Jason Boles, Mohammad Hammoud, and Majd F Sakr. A cloud computing course: from systems to services. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 338–343. ACM, 2015.

[15] J. C. Sandí Delgado and P. Bazán. Educational serious games as a service: Challenges and solutions. *Journal of Computer Science & Technology*, 19, 2019.

[16] Amazon Web Services. Training and certification. https://aws.amazon.com/training/, 2019.

[17] R. Sjodin and M. Lotfy. Integrating cloud computing into the curriculum. In *Journal of Computing Sciences in Colleges.*, volume 35, pages 29–38, 2019.

Cohorting Incoming Students in a CS1 Course: Experiences and Reflections from the First Year of Implementation*

Adrienne Decker[1], *Christopher Egert*[2], *Erin Cascioli*[2]
[1]*University at Buffalo, Buffalo, NY 14260*
`adrienne@buffalo.edu`
[2]*Rochester Institute of Technology, Rochester, NY 14623*
`{caeics, edcigm}@rit.edu`

Abstract

The expansion of computing into K-12 classrooms has created many opportunities for students to be exposed to and choose computing as a field of study. Despite this growth, others are still left with little to no exposure to computing. This creates a disparity of skill levels in CS1 courses, which can, at times, lead to interesting classroom dynamics for both the student- teacher relationship and the student-student relationship. In an effort to help create a more nurturing learning atmosphere for our CS1 students, we undertook a process of cohorting students into course sections based on past experience. The mechanism for grouping the students was based on a self- report survey. Students were placed either into our normal course sequence or accelerated course sequence. In this paper, we will discuss our survey and selection criteria and present results from the first year which resulted in changes in DFW rate in the courses. Even with this success, we end with reflections on what needs to be improved on the process going forward and how this choice may shape our curriculum in the future.

1 Introduction

In CS1 classrooms across the country, we are faced with increasing enrollments in our courses [8]. This comes as both a blessing and a curse. While it shows an

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

increased interest in students in our field and shows that students are recognizing computing as a skill they need to have, it brings a much wider array of students to our doors [7]. Something that is lurking behind those doors is the fact that CS1 isn't easy for many students and several do not master the material adequately enough to pass the course or continue on to further courses [4]. However, we also know that the level of exposure is not uniform for all students in all areas [7]. The problem seems untenable, more students, more diverse backgrounds, and teaching a subject where students have historically had varied levels of success.

Faced with this issue in our own courses, we sought out ways to create a better classroom environment for our students, with the goal to get more of them to successfully complete the course and continue with their studies in computing. As such, we looked towards cohorting students based on prior experience as a possible solution. In this paper, we discuss some other previous attempts at grouping students in introductory courses and describe our approach for determining the cohorts. The paper concludes with our findings from the pilot year of the cohort (2018-2019), some instructor observations, and our plans for the future of cohorting our first-year sequence.

2 Background

Over the last decade, computing programs in the United States expereienced record growth. The CRA update [6] to the Generation CS report [8] shows an incredible increase in the average of computer science majors, noting an increase of 52% in average new majors and an increase of total majors of 35% over the period from 2015 through 2018 [6]. Despite such proliferation of interest and access to computer science education, the introductory course still suffers from low pass rates ranging anywhere from 67-72% [4, 5].

For many CS and computing educators, this pass rate is disturbingly low, and many look to techniques that alter the pedagogy, such as choice of language or paradigm [3, 9, 14], use of mental models [20], or rearranging the relationship between problem solving and programming skills [11, 14]. Others choose techniques to change the learning environment by incorporating active learning exercises [19], using peer instruction techniques [16], and adoption of techniques to address potential disruption of defensive climates in the classroom [2].

Still another method is the separation of introductory students into different classes based upon perceived aptitude and prior experience. Such separation can be desirable, as mixing students with different skill levels in the same course can create issues with efficacy and degrade the classroom experience [13]. However, one of the challenges faced is how to place students in the correct introductory course.

The literature notes several instances of the use of placement tests to determine the proper entry course based on mathematical skill [12], ability to write programs [17], online surveys [18], and self-tests [10]. However, Archer et al. [1] noted that placement tests done before the start of the semester can cause students to be overwhelmed as the activity is lost in the jumble of opportunities and expectations slated for freshman students. Placement instruments can also fail, not providing the outcomes originally intended [15]. Despite such challenges, placement into classes of similarly experienced cohorts holds great potential for students and educators alike.

2.1 Our Course

Our department (Interactive Games and Media) is situated in a college of computing at a large technical university in the northeast United States. Incoming students to the university are accepted into their major from their first year of study and begin taking courses within their department immediately upon entry into the university. The department has roughly 800 students and 30 faculty. Most of the courses in the curriculum are taught in computer labs of 30 students, including the first-year sequence, which is equivalent in topic coverage to a CS1-CS2 sequence.

The language of implementation of the sequence is C. In the past, we have tried to accelerate students with AP CSA credit out of our first course in the first semester, but the language shift (AP CSA is taught in Java), has proved a challenge for the incoming students. Therefore, we made the decision several years ago to not give placement to those AP CSA students with scores of 4 or 5 into the second course. We also did not have any type of challenge or placement exam for our courses that students could have taken to place into the second course. All students, regardless of background, were taking the first course together. The side effects of this were two-fold. First, students who did have prior experience were expressing feelings of boredom and restlessness with both the course and the program. They felt they were not being challenged enough in their first course. Second, students without prior experience were feeling overwhelmed by the ones that did. It was creating, at times, a less than welcoming environment for true novices.

3 Cohorting the First Course

In response to the problems we were facing in our first course, we decided to pursue a cohorting solution for the start of the 2018-2019 academic year. Students who were entering the program/university as first year students scheduled to take our introductory course were asked to complete a survey that we designed to help us determine their exposure to programming prior to enrolling at the university. This survey was advertised to incoming students in the packet they received about attending orientation, which also asks students to take a math placement exam and fill out other forms. The addition of this requirement, while new to our program, was not something radically out of step with other activities the incoming students were asked to do.

3.1 Survey Contents and Administration

We adapted our survey from an unpublished survey we obtained from faculty at the University of Oklahoma. The survey as our students saw it is presented in Appendix A of this paper. The contents of the survey are largely demographic questions asking students about their background with programming or programming courses, including but not limited to, AP courses. The final three questions ask the students to provide answers to programming questions. The first asks for the result from a series of assignment statements. The second asks them to write a loop that prints out the numbers from 1 to 100 in any programming language of their choice. The third asks them to write a function that given a collection and an element returns the index of an element if found and -1 if not, in any language of their choice. The survey delivery system (Qualtrics) is a simple text editor with no functionality for compiling or running code.

3.2 Analyzing Survey Results

Once they survey closed, all student responses were downloaded into a spreadsheet and specific answers were analyzed to determine which section the students should be placed into (accelerated or regular). Of the 207 incoming students who were asked to complete the survey 135 did so, a 65.2% completion rate. Students were placed into the *regular section* if they:

- Indicated they had never written a computer program before or had never taken a programming course before (21 students).
- Indicated that they had programmed before, but for a duration of less than 6 months, may have taken AP CSA with an exam score of 1 or 2 or no score reported, and had indicated that they had only written programs less than 5 screen lengths long (20 students).
- Indicated larger amounts of programming experience with longer programs, answered correctly or nearly correctly on the first two programming questions, but did not answer the third programming question (22 students).
- Indicated larger amounts of programming experience with longer programs but did not give a satisfactory answer to the third programming

question (10 students). For purposes of the analysis of the survey responses, a satisfactory answer consisted of a function and indicated parameters, a loop that iterated over entire collection (or until element found) and returning an index of the element or -1 if not found. Syntactical errors that did not impact the algorithm were not weighed in our assessment of the solutions.

Students were placed into the *accelerated section* if they:

- Reported a score of 4 or 5 on the AP CSA exam (23 students).
- Reported taking either AP CSA or both AP CSA and CS Principles, said they had been programming for more than six months and produced a program that was more than five screen lengths or files. Quickly looking at their programming question answers indicated that they answered all three to some reasonable degree (15 students).
- Reported having programmed in one or more programming languages, had programmed for over six months and produced programs of five or more screen length. A quick look at their programming question answered indicated that they answered all three to some reasonable degree (24 students).

In the end, out of the 135 students, 62 students were placed into the accelerated section and 73 students were placed into our regular section. Students who did not complete the survey were also placed into our regular section unless they had an AP CSA score of 4 or 5 [17 students].

From a workload standpoint, while 135 students took the survey only 83 needed to answer the programming questions due to the skip logic in place for some of the answers. The third question was the most work to grade but because students skipped the question or were able to be placed with a combination of their other answers, only 41 responses needed to be carefully screened. Thus, aside from building initial Excel formulas (which can now be re-used), assessment of the student responses took about 1 hour by one instructor.

4 Course Results

We will discuss some general observations of the classroom and the process in Section 5. This section presents results in terms of grades and retention for the students who were our incoming students for the 2018-2019 academic year.

In the fall term 2018 (August-December 2018), there were eight sections of the first course taught by four instructors (A, B, C, D). Three sections were accelerated and five were regular. Instructors A and B taught the accelerated sections. Instructors A, C, and D taught the regular sections. For the spring

2019 term (January-May 2019), there were seven sections of the second course taught by four instructors (A, B, C, E). Three sections were accelerated and four were regular. Accelerated sections were taught by A and B again. The regular sections were taught by B, C, and E. For all sections throughout the year, each section had approximately 30 students and all were taught in- person in computer labs.

By university policy, students are allowed to elect to receive a grade of W (withdraw) up until the end of the 11 th week of the 15-week term. Grading across the sections of the courses was based on roughly the same criterion and in many cases, the homework assignments were the same across all sections of the course.

4.1 First Course Results

Of the 78 students that were enrolled in the accelerated sections of the introductory course, two students elected to go to the regularly-paced sections of the course as opposed to staying in the accelerated sections during the first week of class, leaving 76 students in the accelerated sections. Table 1 presents the final grades for students in the accelerated sections of the first course. The DFW rate for the accelerated sections was 7 out of 76 (9%).

Based on the results of the survey, 73 students were placed into the regular sections of the course. They were joined by the two students mentioned above, and 56 students who did not complete the survey for a total of 131 students. Table 1 presents the final grades. The DFW rate for the regular sections of the course was 20 out of 131 (15%). Overall, the DFW rate for the first course for the incoming students was 27 out of 207 (13%).

Table 1: Final Course Grades in first course in 2018-2019 academic year

Final Letter Grade	Accelerated (n=76)	Regular (n=131)
A	36 (47%)	51 (39%)
A-	12 (16%)	14 (11%)
B+	8 (11%)	10 (8%)
B	7 (9%)	18 (14%)
B-	2 (3%)	6 (5%)
C+	1 (1%)	6 (5%)
C	1 (1%)	6 (5%)
C-	3 (4%)	0 (0%)
D	3 (4%)	13 (10%)
F	3 (4%)	3 (2%)
W	1 (1%)	4 (3%)

4.2 Second Course Results

Students who did not receive a grade of C- or better in the first course were not allowed to take the second course in either the accelerated or regular sections. Students in the accelerated section of first course could opt to take the regular section of second, but none chose to do so. Students were not allowed to move from the regularly-paced first course to the accelerated second course.

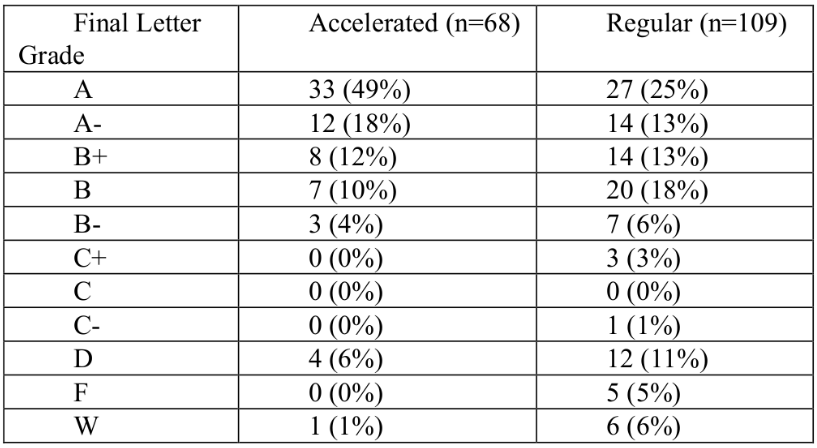
There were 68 students (out of 69) enrolled in the accelerated sections of second course for the second term of the academic year, retention rate 99%. Table 2 presents the final grades for students in the accelerated sections of the second course. The DFW rate for the course was 5 out of 68 (7%).

There were 116 students enrolled in the regular sections of the second course, but only 109 were students from the most immediate preceding term. Therefore, the retention rate was 109 out of 131 (83%). Table 2 presents the final grades for students in the regular sections of the second course. The DFW rate for the course was 23 out of 109 (21%). Overall, the DFW rate for the course was 28 out of 177 (16%).

5 Discussion and Observations

While the overall course outcomes give us one dimension of the picture of the success of the cohorting experiment, another piece of this picture is the classroom climate for the students, which we present at this time as instructor observations.

Table 2: Final Course Grades in second course in 2018-2019 academic year



Final Letter Grade	Accelerated (n=68)	Regular (n=109)
A	33 (49%)	27 (25%)
A-	12 (18%)	14 (13%)
B+	8 (12%)	14 (13%)
B	7 (10%)	20 (18%)
B-	3 (4%)	7 (6%)
C+	0 (0%)	3 (3%)
C	0 (0%)	0 (0%)
C-	0 (0%)	1 (1%)
D	4 (6%)	12 (11%)
F	0 (0%)	5 (5%)
W	1 (1%)	6 (6%)

In terms of the survey itself, we feel that we successfully achieved our outcomes. It was easy to administer, reasonably quick to analyze the results, and fairly accurate in terms of not putting under-prepared students into the accelerated section. Since there were only two students who were recommended for the accelerated sections that selected to move to regular, students did not generally feel misplaced by the survey results. Also, instructors did not report students in the regular section who took the survey complaining about their placement and asking to be accelerated.

We had higher than normal retention rates in each of the cohorts and overall, 177 out of 207 retained from the first to the second course for an 86% retention rate. Since 27 of the non-continuing students were DFW, we only lost three students who passed the first course but did not take the second. One student switched majors within the university, one seemed to have left the university because they were not enrolled in the second term in any courses, and one never intended to continue on, taking our first course to fulfill a general education requirement.

Our DFW rates were near or lower than our historical averages for both groups (accelerated vs. regular). Historically, we have had a 15% DFW rate in our courses. This past year, with cohorting, the rates were 13% and 16% respectively. The DFW rate in the accelerated group was even lower, 9% and 7% respectively.

It is unclear that cohorting was the cause of the change of our DFW rates or retention rates. While there were no dramatic overhauls of the curriculum undertaken for the 2018-2019 academic year and the instructors for this cohort have all taught the course before (in some cases for almost a decade), we can't conclusively determine the cause because we did not formally study or control for factors of instructor influence. However, something did change and the change in the DFW rate followed. The one thing that definitively changed was the cohorting.

Looking at the students who ultimately did not succeed in either course, a major factor seemed to be lack of attendance or failure to turn in assignments. Anecdotally, it would seem that this is a common reason students fail any course, so it is not a surprising observation. A regular-section only phenomenon was related to the impact of external help. For many students, they seemed to be receiving a lot of external help on their assignments from other students (near peers or upper-class peers), tutors, or other sources. We are not characterizing this help as inappropriate, but rather, they relied on the help to get them through assignments outside of class and could not demonstrate their knowledge inside of class on assignments, exams, or discussion.

One point that was unique to the accelerated section was an over-inflated opinion of their own abilities. For example, one of the accelerated students would often skip class, thereby missing new coding concepts, and instead try to rely on their prior knowledge from their high school AP course which often didn't work in C.

In terms of general classroom environment, instructors did observe some decrease of the "toxic wizard" attitudes in the regular sections. Since a majority of the students with advanced knowledge were removed from the sections, it seemed to allow students who were unsure to voice their concerns more often. Our solution was not perfect, however, because not all students took our placement survey and therefore, there were most likely students with possibly extensive prior non-AP experience in the regular section because they failed to fill out the survey and we did not have AP scores to use for their placement.

One place that the accelerated students demonstrated an advantage of the cohorting was in the second course. That course has a group project and instructors observed that the student teams in the accelerated sections pushed their projects farther than they remember previously. The project has been the same open-ended project for more than ten years. As opposed to competitive pushing, the instructors saw this more as a cooperative/supportive pushing. They observed that since students were roughly in the same place in terms of experience and background, they could focus on the project and not on students with less depth of understanding of the material. It seemed to allow time for further exploration and deeper development of the team's ideas.

There are limitations to our process for the cohorting and our observations of the results. Both illuminate the need for further investigation. First, this was not in any way a research study. It was definitely a pilot offering of both the survey and the courses cohorted in this way. While we are happy with the results of the survey and feel it adequately helped us determine placement, a more rigorous investigation of the instrument is needed. The classroom observations of both the climate and student interactions are strictly anecdotal. They could be enhanced by formal observation protocols to determine what the classroom climates really are in the two cohorts. Lastly, the fact that not every student took the placement survey makes it unclear that the cohorts were accurate. The instructors definitely felt there were students in the regular sections that probably should have been in the accelerated section but did not complete the survey.

6 Conclusion

In this paper, we discuss the implementation of a largely self-report survey for determining placement in an introductory course. The survey uses information on prior experience, both formal and informal to determine placement. After the first year, we have seen a decrease in the DFW rate in both our first-year courses and have observed that the classroom climate has improved for our students. The pilot was considered enough of a success to once again use the survey for placement in the 2019-2020 academic year. Future work needs to be done to determine the exact impacts on the classroom environment and the impacts of the initial cohorting on the students as they progress through our program and to degree completion.

References

[1] Glen Archer, Briana Bettin, Leonard Bohmann, Allison Carter, Christopher Cischke, Linda M Ott, and Leo Ureel. The impact of placement strategies on the success of students in introductory computer science. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE, 2017.

[2] Lecia Jane Barker, Kathy Garvin-Doxas, and Michele Jackson. Defensive climate in the computer science classroom. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 43–47, 2002.

[3] David John Barnes, Michael Kölling, and James Gosling. *Objects First with Java: A practical introduction using BlueJ*. Pearson/Prentice Hall, 2006.

[4] Jens Bennedsen and Michael E Caspersen. Failure rates in introductory programming. *AcM SIGcSE Bulletin*, 39(2):32–36, 2007.

[5] Jens Bennedsen and Michael E Caspersen. Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2):30–36, 2019.

[6] Betsy Bizot and Stu Zweben. Generation CS, three years later. https://cra.org/crn/2019/08/generation-cs-three-years-later/.

[7] code.org. State of computer science education.

[8] Computing Research Association. Generation CS: Computer science undergraduate enrollments surge since 2006. http://cra.org/data/Generation-CS.

[9] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. *ACM SIGCSE Bulletin*, 35(1):191–195, 2003.

[10] Lucia Dettori, Theresa Steinbach, and Martin Kalin. Is this course right for you? using self-tests for student placement. *Director*, page 07, 2006.

[11] Katrina Falkner and Edward Palmer. Developing authentic problem solving skills in introductory computing classes. In *Proceedings of the 40th ACM technical symposium on Computer science education*, pages 4–8, 2009.

[12] Carl Farrell. Predicting (and creating) success in CS1. *Issues in Information Systems*, 7(1):259–263, 2006.

[13] Päivi Kinnunen and Beth Simon. CS majors' self-efficacy perceptions in CS1: results in light of social cognitive theory. In *Proceedings of the seventh international workshop on Computing education research*, pages 19–26, 2011.

[14] Theodora Koulouri, Stanislao Lauria, and Robert D Macredie. Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)*, 14(4):1–28, 2014.

[15] Cindy Marling and David Juedes. CS0 for computer science majors at Ohio University. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 138–143, 2016.

[16] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. A multi-institutional study of peer instruction in introductory computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 358–363, 2016.

[17] Robert H Sloan and Patrick Troy. Cs 0.5: a better approach to introductory computer science for majors. *ACM SIGCSE Bulletin*, 40(1):271–275, 2008.

[18] Leen-Kiat Soh, Ashok Samal, Suzette Person, Gwen Nugent, and Jeff Lang. Designing, implementing, and analyzing a placement test for introductory cs courses. *ACM SIGCSE Bulletin*, 37(1):505–509, 2005.

[19] Keith J Whittington. Infusing active learning into introductory programming courses. *Journal of Computing Sciences in Colleges*, 19(5):249–259, 2004.

[20] Susan Wiedenbeck, Deborah Labelle, and Vennila NR Kain. Factors affecting course outcomes in introductory programming. In *PPIG*, page 11, 2004.

Appendix A (Survey)

1. Have you ever written a computer program? (Yes/No)
2. Have you taken a computer programming class before? (Yes/No)

If they answer "NO" to both of the above, survey ends.

3. What programming languages have you used?
   Java, Python, C, C++, C#, Visual Basic, HTML, Scratch, AppInventor, Other (fill in)
4. Have you taken an AP Computer Science course? (only need to ask if answer to 2 was "yes")
   a. I have taken AP Computer Science A only
   b. I have taken AP Computer Science Principles only
   c. I have taken BOTH AP Computer Science A and AP Computer Science Principles
   d. I have not taken any AP Computer Science courses
5. What score did you get on AP Computer Science A? (If answered A or C in Q4)
   a. 1 b. 2 c. 3 d. 4 e. 5
6. What score did you get on AP Computer Science Principles? (If answered B or C in Q4)
   a. 1 b. 2 c. 3 d. 4 e. 5
7. How long have you worked on learning to program a computer?
   a. A few hours or less b. More than a few hours, but less than two months
   c. Between two and six months d. More than six months
8. What is the longest program you've ever written?
   a. One screen-length or less; one file or less b. Between one and two screen-lengths or files
   c. Between two and five screen-lengths or files d. More than five screen-lengths or files

Given ONLY to students who answered Q6 with D or E OR Q7 with C or D OR Q8 with C or D, not given if Q5 answered

9. What is the value of x after the statements below are executed?

```
int x, y;
x = 10;
y = 20
x = y;
```

   a. 0 b. 10 c. 20 d. Some other value (write in): e. I don't know
10. Write a loop to print out the numbers from 1 to 100. If you don't know how to do this, select "I don't know"
11. Write a method (called a procedure or function in some languages) to determine whether a given target value is in an array of integers. The method should return the index where the target value was located or -1 if the target is not in the array. If you don't know how to do this, select "I don't know"

Creation of a Virtual Machine for a Database Class[*]

Christine F. Reilly
Computer Science Department
Skidmore College
Saratoga Springs, NY 12866
creilly@skidmore.edu

Abstract

In a database class, it is essential for students to connect to a database server so that they can practice writing and running SQL queries. Providing the same database environment to all students presents a challenge to the professor. This paper discusses the creation of and experience with using a virtual machine (VM) for both an undergraduate and a graduate database class. The VM enabled all students to access the same database environment from their personal computers. Experience with using this VM during three semesters shows that this is a feasible approach. This paper contains instructions and guidelines for creating the VM and distributing it to students. The approach discussed in this paper is useful for other professors who are teaching database courses, and can be modified for use in other types of classes.

1 Introduction

This paper discusses the creation of and experience with a virtual machine (VM) for both an undergraduate and a graduate database class. The professor created a Linux VM that contained all of the software and example databases that were used for in-class examples, assignments, and the database design project. By providing students with this VM in combination with making the same resources available in the department computer lab, the professor could

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

ensure that students had access to the same computing environment regardless of whether they chose to work in the department lab or on their personal computers. The approach described in this paper is likely to be useful in other computer science courses, and was inspired by a set of computer security assignments [4].

The VM described in this paper was first used in undergraduate and master's-level database courses in the Computer Science department at a regional public university in the USA that is designated as a minority serving institution. Many of the undergraduate and graduate students who took these courses were classified as non-traditional students because they may not live on campus, may have full time jobs, may have significant family responsibilities, or may be older than traditional college age. In the USA, the majority of college students fall into the non-traditional category [6], and it is critical for faculty to consider the needs of non-traditional students when choosing course materials and designing assignments. The author has also taught at institutions that mostly have traditional college students and finds the VM approach is useful when students prefer to work on their personal computer rather than go to the department computer lab.

The students who took the database classes described in this paper have a wide variety of experience with the use of computers and with computer system administration, and come from a diverse range of backgrounds. In order to provide all students with opportunities to access the database software, the following principles were followed during the design of the VM:

- Use free and open source software and tools.
- Do not expect students to have prior knowledge about computer system administration.
- Expect that students may have personal computers that have limited computing resources (memory, disk space, processing capacity).

In order to motivate the use of a VM for database classes, other approaches for providing database access to students are surveyed in Section 2. Next, the steps for creating the VM, installing the software for database class, and distributing the VM to students are discussed in Section 3. The experience of using this VM during three semesters of database classes is presented in Section 4, followed by conclusions in Section 5.

2 Related Work

A variety of approaches for providing database access to students have been described in prior work. These approaches are grouped into two general categories: a common platform used by all students, and utilizing the student's personal computer. This paper focuses on a hybrid of these two approaches: having students use a common platform on their personal computer. A discussion of the benefits and drawbacks of each approach follows.

2.1 Common Platform

Providing a common platform for all students in a database class ensures that students are completing their SQL assignments in the same environment where they will be graded. This is beneficial to the students because they can properly test their SQL queries before handing in the assignment. A benefit to the professor of the common platform approach is having a centralized platform to maintain. Benefits and drawbacks of specific methods for providing a common platform are discussed below.

A department database server can be used as the common platform for a database class. As long as the professor is granted sufficient administrative privileges on the department database server, the professor can easily upload customized databases for the class and create database server accounts for students. Students can connect to the department server from the lab workstations at the college and possibly from their personal computer. Due to security concerns, the institution may require the use of a virtual private network for connections from off campus. The challenges of connecting to the department database server from off campus is a drawback to this approach. An additional drawback is that if class assignments include modifying the data in the database then each student must be provided with a separate database and database user.

Another way to provide a common platform is to utilize an online SQL tool. There are a variety of online tools available from third-party vendors and free websites. However, these online tools have many drawbacks from the professor's perspective [2]. Some of the online tools do not allow the professor to upload a custom database, and students are able to search the internet for solutions to homework problems that use the common databases. Other drawbacks are that the online tool tool could become unaccessible in the middle of the semester and that the third-party vendor may not provide adequate customer support.

Simple database software, such as Microsoft Access or LibreOffice Base, can be installed on the workstations in the department's computer lab [2]. Because the database is stored as a single file, the instructor can provide students with a file that contains the example database. The benefits of this approach include the simplicity of installation and use. Drawbacks include that these are not production-level database systems, and that students can use the graphical user interface that is provided with these software products to create SQL queries without actually writing SQL.

Cloud computing providers such as Google Cloud Platform (GCP) and

Amazon AWS are willing to donate computing credits for educational use. With these computing credits, students in a database class can utilize cloud database platforms for their coursework [3]. Benefits of this approach include that the cloud platforms often have a variety of software available for use, allowing students to gain experience with multiple database systems; and students gain useful career experience by working in a real-world environment. The main drawbacks of this approach are related to the computing credits that must be provided to students. The instructor must request donated computing credits prior to each semester and administer the distribution of computing credits to students, and these credits may expire soon after the course ends. It is also possible that students can run out of computing credits prior to completing their coursework, especially if they make mistakes in the use of the cloud platform. The responsibility of monitoring and managing computing credits provides a useful real-world experience.

2.2 Student's Personal Computer

The students can be expected to install a database server on their personal computer and then to load the example databases onto their computer. A benefit of this approach is that many students prefer to use their personal computer rather than the department computer lab, and this approach enables students to do classwork off campus. The drawbacks of this approach include requiring students to have some knowledge of computer system administration, and the difficulty of supporting the variety of operating systems on the students' personal computers.

Students can install simple database software, such as Microsoft Access or LibreOffice Base, on their personal computers. The benefits and drawbacks are the same as discussed with this approach in Section 2.1. An additional drawback is that students may have different versions of the software on their personal computers and compatibility issues can arise when using the database file provided by the professor.

2.3 Virtual Machine

A third approach, and the focus of this paper, is to provide students with a virtual machine (VM) that contains the software and data that is used in the database class. This is a hybrid approach that enables students to use a common database platform on their personal computer. The main drawback of this approach is that each student must have a computer with sufficient resources to run the VM. In order to ensure that all students have access to the resources needed for class, it is best to combine the VM approach with a second option such as connecting to a database server from the workstations in the department computer lab.

3 Providing a VM for DB Class

This paper focuses on how to use a VM to provide the students in a database class with access to the same database environment. A similar approach could be utilized in other classes where it is useful for all students to have access to the same computing environment. As will be discussed in Section 4, this VM has been used in undergraduate and graduate database classes where students complete homework assignments that require them to create SQL statements, and where the final project for the course is a database design project.

The steps for creating the Linux VM are discussed in Section 3.1, then the steps for preparing the VM for use by the database class are discussed in Section 3.2. The methods for distributing this VM to students are presented in Section 3.3. Faculty who are interested in using this VM approach for a different course can overlook the steps in Section 3.2 and instead install the software needed for their course. Table 1 provides a summary of the software that is installed on the VM.

Table 1: Software Installed on Virtual Machine

Category	**Software**
Linux	Lightweight distribution such as Bodhi Linux; Run the system update utility; VirtualBox Guest Additions.
Database	MariaDB Server; Graphical interface such as phpMyAdmin.
Development Software	Java JDK; Java Development Environment.

The overall approach taken when designing the VM is to keep it as small as possible, both in terms of the size of the file that contains the VM and the amount of memory used while the VM is running. A small VM requires less computing resources on the students' personal computers, making this approach accessible to students who have lower cost or older computers. Table 2 outlines the requirements for running this VM on the students' personal computers. When this paper was written in 2019, the low cost personal computers available from retail stores had 4 GB to 8 GB of RAM. The lowest cost personal computers in the netbook category, such as the Google Chromebook, are most likely not supported by this approach because of limitations in the operating system used by these computers. Students who have experience in computer system administration may be able to find a way to modify the netbook operating system and install the VM software.

Table 2: Requirements for Students' Personal Computers

Category	**Requirement**
Operating System	Any that serve as VirtualBox host (Windows, Mac OSX, or Linux); Permission to install new software.
Software	Oracle VM Virtual Box (free and open source software).
RAM	8 GB or more preferred; 4 GB may be sufficient.
Available Storage Space	10 GB or more.

3.1 Creating the VM

Oracle VM Virtual Box [9] was chosen as the VM software because it is free and open source software that is available for all of the operating systems that students are likely to have on their personal computers (Windows, Mac OSX, and Linux). In order to keep the VM as small as possible, choose a lightweight Linux distribution that has low hardware requirements and does not install much software by default. When this paper was written in 2019, Bodhi Linux Standard Release 5.0.0 [1] was chosen as the operating system to install on the VM. Because Linux distributions evolve over time, it is suggested that the reader perform a web search for "lightweight Linux distribution" in order to find a Linux distribution that is appropriate at the time when the reader is creating their VM.

The following steps detail how to create the Linux VM. These instructions assume that the reader is familiar with the use of VirtualBox and with installing and using a Linux distribution. For more information about these topics, consult the product documentation or search the web for tutorials.

1. Download VirtualBox and install it on your computer.

2. Download the Linux disk image to your computer.

3. Open VirtualBox and create a new VM with a name such as LinuxDB. For the Bodhi Linux VM in 2019, 1 GB of RAM and a dynamically allocated hard drive with maximum size of 8 GB were adequate.

4. Start the LinuxDB VM. In the boot image dialog, select the previously downloaded Linux disk image.

5. Follow the installation instructions for the Linux operating system.

6. Document the Linux username and password and provide this information to students so that they can log into the machine and act as the system administrator on their VM.

7. Run the system update utility to install available software updates.

8. Install VirtualBox Guest Additions so that the guest machine works more seamlessly with the host.

3.2 Preparing the VM for Database Class

For database class, install a database server and other software and tools that students will utilize for the class. Note that it is best to carefully consider the software that will be used throughout the semester and include all of this software in the VM that is distributed to students at the beginning of the semester. Two commonly used free and open source database servers are MariaDB and PostgreSQL. MariaDB [5] was chosen for the class discussed in this paper because it, and the MySQL database server that the MariaDB code was forked from, is commonly used in production environments. A pedagogically interesting option is to install database servers from more than one vendor and include comparisons of different database systems as a learning activity [7]. The following steps outline the process of installing the database class software on the VM.

1. Use the package manager to install MariaDB Server.

2. Use the package manager to install a graphical user interface for the database (e.g. MySQL Workbench or phpMyAdmin).

3. Install any other tools and programs needed for the class (e.g. Java JDK and a development environment).

The database server must be prepared for the class. The preparation includes creating database users and creating the test databases that will be used during class. The following steps outline these preparations.

1. Create database users and document the usernames and passwords for the students. A suggested practice is to create at least two database users: one with read only access, and the second with all database privileges. Encourage students to develop good database security habits by connecting as the user with the minimum privileges needed.

2. Create databases and upload data to these databases. For example, include the textbook example database and the databases that will be used for in-class examples and SQL assignments. Grant appropriate permissions for each of the database users on the databases.

3.3 Providing the VM to Students

VirtualBox has a utility for exporting the VM to a file that can be distributed to others. Use the "Export Appliance" dialogue to create a file (with .ova extension) that contains the VM you created. Distribute this file to students and instruct them to use the "Import Appliance" dialogue to add this VM to VirtualBox on their personal computer.

The professor should provide students with a document that contains instructions for installing the VM, information about basic Linux use and use of the installed database, and guidelines for working with a VM. Make sure to include the usernames and corresponding passwords for the Linux user and the database users. Guidelines about using a VM should include instructions for taking periodic snapshots of the machine state, and a suggestion that VM performance can be improved by closing all other programs on the host machine and being patient while the VM starts up and acquires memory resources from the host machine.

4 Experience

This VM setup was deployed in three semesters of database classes. One semester was an advanced undergraduate course, and two of these semesters were masters-level courses. This section discusses the professor's experience with using the VM, including informal feedback that students provided to the professor. In addition to providing this VM to students, the professor also provided access to the same example databases on a server that students could access from the department computer lab. The combination of a VM and the department server was sufficient for providing all students with access to the database resources needed for the classes.

Overall, the use of the VM for the database classes was successful. Because students used the computer platform that would be used for grading, the professor had fewer questions about SQL assignment grades. An active learning approach was utilized for the in-class portions of the SQL unit [8], and the VM was used by students for their pre-class assignment. For the semester when class could not be scheduled in the computer lab, enough students brought laptops running the VM to class so that they could work in small groups on the in-class learning activities.

The lesson of keeping the VM as small as possible was learned in the first semester. In the first iteration of this approach, the professor used the default VM settings in Virtual Box and the default installation of Ubuntu Linux. Some students found that this Ubuntu VM was unusable on their personal computer, and the faculty observed that the VM ran slow on even on high-end computers. Students had fewer complaints about the performance of the VM when it was created with a smaller memory and disk size and when it ran a lightweight Linux distribution as described in Section 3.

The students in these classes had a wide variety of experience with using computers and with computer system administration. Students who had issues with slow performance of the VM told the professor that they would switch to using a different program on their computer while they waited for the VM. This provided an opportunity for the professor to discuss how an operating system virtualizes memory in order to provide more memory resources than are available in RAM. The professor also specifically instructed students who experienced performance problems with the VM to close all other programs running on their computer and to keep the VM window active so that more of their computer's memory would be made available to the VM.

Providing the VM to the students seemed most beneficial to the students with less experience using computers. The professor observed that this group of students continued to use the VM throughout the semester as the class worked on their database design projects. Students who have more experience with computer system administration tended to install database software directly onto their personal computers and abandon the use of the VM after the class finished the SQL unit. This demonstrates the benefit of providing students with a VM in the situation where the class size is relatively large and the professor has limited time available for instruction beyond the class topics or for helping students with computer troubleshooting. By having a VM available, all students in the class had access to the computing platform that they needed for success in the class.

5 Conclusion

This paper provided a demonstration of how to create a VM for a database class, and discussed the experience with deploying this VM in three semesters of classes. The professor found that providing students with the VM enabled students to complete SQL practice problems and assignments on a consistent environment and reduced the number of questions related to platform-specific issues. The VM was especially useful for students with less experience using computers and ensured that this group of students was able to fully participate in the SQL coursework.

An additional benefit of providing students with a Linux VM is that students can use the VM to learn about using and administering a Linux machine. The ability to save snapshots of the VM state encourages students to experiment with Linux because they can easily roll back to an earlier snapshot in the case that they take some action that causes the machine to not function properly.

Because many students prefer to work on their personal computer rather than in the department computer lab, a VM that is configured for a specific class is a good option for database classes and other computer science classes. By providing both a VM and access to the computing resources in the department computer lab, a professor can ensure that all students have access to the resources needed for the class.

Acknowledgements

The students in database classes at University of Texas Rio Grande Valley provided useful feedback about using the VM. Skidmore undergraduates (Selina Almasarwah and Matt Clark) edited the final version of this paper.

References

[1] Bodhi Linux. https://www.bodhilinux.com/ (accessed November 2019).

[2] John Cigas and Barbara Kushan. Experiences with online SQL environments. *Journal of Computing Sciences in Colleges*, 25(5):251–257, May 2010.

[3] Karen C. Davis. Teaching database querying in the cloud. In *2019 IEEE Frontiers in Education Conference (FIE)*, 2019.

[4] Wenliang Du. SEED: Hands-on lab exercises for computer security education. *IEEE Security and Privacy*, 9(5):70–73, September 2011.

[5] MariaDB. https://mariadb.com/ (accessed November 2019).

[6] Alexandria Walton Radford, Melissa Cominole, and Paul Skomsvold. Demographic and enrollment characteristics of nontraditional undergraduates: 2011-12. Technical Report NCES 2015025, National Center for Education Statistics, 2015.

[7] Gary B. Randolph. The forest and the trees: Using Oracle and SQL Server together to teach ANSI-standard SQL. In *Proceedings of the 4th Conference on Information Technology Curriculum*, CITC4 '03, pages 234–236, New York, NY, USA, 2003. ACM.

[8] Christine F. Reilly. Experience with active learning and formative feedback for a SQL unit. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2018.

[9] VirtualBox. https://www.virtualbox.org/ (accessed November 2019).

Jupyter Notebooks versus a Textbook in a Big Data Course[*]

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

Abstract

In building curriculum in new areas of computer science, often the tools introduced in the course are an important component. This is especially true in the area of big data, where the complexity of the problems the area tackles is high. In the 4 years since its inception, my big data course has gone through two major redesigns and has settled on a tool set including: the Hadoop platform, Spark processing engine, the Python programming language, Eclipse IDE, and Jupyter Notebooks. Many of the changes were driven by input from professional peers on big data teams, who were struggling with the complexity resulting from the low-level programming model used by MapReduce. Jupyter Notebook, a type of computational notebook, was added to the course to introduce students to the Python programming language. Data scientists and researchers have found computational notebooks an effective tool to manage their work by providing a way to track their thinking process, their code, and conclusions in one web document. To assess the effectiveness of using Jupyter Notebook in a big data course, students' views on the use of computational notebooks and traditional textbooks were captured and statistically analyzed.

1 Introduction

Software complexity has long been a topic of interest in computer science. It has been written that "the rise of information system complexity, however, is

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

driven by the incredible capabilities and opportunities computing offers us" [15]. To take advantage of those opportunities we must apply our computer acumen to move difficult problems, leading to more layered, complex systems. These same authors note that "earlier research shows that significant gains can be made by using PC development tools, graphical interfaces, packages software, and simpler programming languages" [15]. It can easily be argued that a big data platform, with its 100s of open source, distributed components, reacting to complex queries in a self-adaptive manner represents a complex system. In 2015, I was part of an effort to develop a course that would teach undergraduates about this new area of computer science [7]. The core concepts covered in the course (distributive processing, functional programming, data management, abstraction) were easily decided early in the course design process. The software implementation details (frameworks, programming languages, and IDE) have been much thornier. The experience of teaching the course over the last 4 years has resulted in two major revisions. Looking back on the journey now, it is easy to see that my struggle was about managing complexity and how to teach students about a topic, i.e. big data, that had not yet developed all the tools needed to manage that complexity. As the field developed those tools, the task of education became easier by incorporating those tools into the classroom.

One of those tools is the computational notebook. Many researchers have discovered that computational notebooks are a valuable way to centrally capture the thought process, data discovery and visualizations of their work. These notebooks become interactive documentation that helps researchers manage the complex process of coming to a research conclusion. A computational notebook can be reviewed and executed months after the original work was completed. For that reason, notebooks have become an important collaboration tool among researchers working on the same or related projects. A notebook's usefulness has extended to areas outside universities. I have seen computational notebooks used often by presenters at technical conferences, as well as IT practitioners, to explore and present new technology.

Noting how others were taking advantage of notebooks caused me to wonder how this tool could be helpful in teaching a complex topic like big data. Two collaborators and I did some initial work examining three separate efforts to support undergraduate statistics projects using Jupyter and R [5]. Our findings in this initial work was anecdotal. If I was going to incorporate this new tool, I wanted to be able to measure its impact. I wanted to determine if the students experienced those benefits. Did they perceive an advantage in using a notebook over a traditional textbook? The goal of this project is to answer that question.

For the remainder of the paper, I describe my exploration, implementation and research in the benefits of computational notebooks usage in the following sections: Background, Big Data Course Redesign, Research Methodology, Results, and Conclusions.

2 Background

Formed in 2001 by Dr. Fernando Perez, the IPython project developed a command shell for interactive computing in multiple programming languages. In 2014, the group spun-off the Jupyter project [2]. The new project's main goal was to provide tools for interactive data science. Its first product was Jupyter Notebooks [3], a computational notebook. The Jupyter project development team is made up of contributors from several large universities and prominent technology companies. In 2017, the Jupyter project won the ACM Software System Award. The award committee seeks to recognize software projects "for developing a software system that has had a lasting influence, reflected in contributions to concepts, in commercial acceptance, or both" [1].

The Jupyter designers realized that by clearly identifying the purpose of a section of a web document, you could build specific functionality around those sections. Referred to as a cell, these cells can perform specific tasks, but still reside in the same web document. Cells can be markup cells, where researchers capture descriptions of their project, identify their current understanding, post diagrams or images, and list additional questions to be asked. Code cells (executables) are web page sections in which researchers provide program code, queries or job flows that can provide answers or progress on those questions. Jupyter provides processing kernels that allow the commands in the code cells to be executed. There are approximately 114 kernels available that support many programming languages and frameworks. Output from the code cells, once executed, can also be stored in the notebook. Researchers can build web documents by combining cells of different types based on their research and documentation needs.

As educators in researcher universities saw the notebook's potential, they began introducing Jupyter into their classes. As a result, the interest in 'education-friendly' functionality grew. Instigated by Dr. Jessica Hamrick at UC Berkley, the NBGrader project was founded to develop additional Jupyter functionality that allowed educators to distribute and automate the grading of course assignments in Jupyter Notebooks [4]. With these additional libraries, notebook designers can add cells which can execute and test the results from other code cells, identified as assignments, in the notebook. This provides two important capabilities: it provides instructors the ability to automate the grading of the notebooks, and most importantly, when made available to students, it provides instant feedback to students on their assignment solutions.

2.1 Jupyter Notebooks in Research and Education

Over the last couple of years, references to Jupyter Notebooks have increased in research publications across many disciplines. Randles discovered 91 research articles in the Astrophysics Data System literature that mentioned Jupyter Notebooks [12]. That same year, Rule analyzed 1.23 million Jupyter Notebooks on GitHub across 100,500 users [14]. People gravitate to tools that make them productive and add value.

Researchers, who for decades have dealt with large amounts of data, have streamlined their data delivery process using Jupyter Notebooks. Over the last couple of decades HPC centers have worked to make their data and processing services more easily accessible to researchers and students. The concept of a Science Gateway has developed out of this work [16]. These Gateways began as simple web views of subsets of data. Of late, HPC facilities have implemented Jupyter environments to increase data access and processing flexibility. Researchers have free reign to use the Jupyter server to generate and post individualized notebooks using HPC data. The individualized notebooks can become permanent members of their analytical platform based on peer feedback. Some of the most interesting exploration of Jupyter architectures is happening in this space giving researchers the tools to more easily curate and manage data [11].

Data cleaning and curation is an important step in high-quality analysis. Freire examined how different tools supported the process of data curation [8]. She identified the differences between how a spreadsheet, a computational notebook, and relational database handles data exceptions. Seeing the value of the added functionality in notebooks, her research team developed Vizier, a data curation environment that combines the flexibility of spreadsheets, notebooks and relational databases.

Exposed to computational notebooks used in their research, instructors began considering them as educational tools. Educators have developed a flood of experiential approaches to learning. It is not surprising that education researchers have developed Jupyter Notebooks in support of these experiential learning techniques. Hu found that it was easy to implement a cycle of experiential learning in Jupyter. It easily supported exploration, concept invention, and concept application in notebooks that supported the collaboration needed the POGIL learning approach [10]. Rowe found that the Jupyter environment was a great way to present topic examples, where the students could modify and re-execute code to examine the impact of their changes [13]. In teaching computer science to science majors, Smith found that the simplicity of Python and Jupyter Notebooks allowed him to focus more on the concepts in the labs rather than how to do it, which an interactive development environment (IDE) and a more complex languages like Java would require [17]. In teaching High-

Performance Computing, Glick found that notebooks were a good tool for both directed and self-directed instruction [9]. Students' course feedback pointed out the benefits of being able to execute code examples and then learn by changing the examples and re-executing them. These studies were convincing evidence of the benefits others see in computational notebooks.

3 Big Data Course Redesign

In 2017, I conducted an informal survey on big data implementations from peers that worked on big data teams for large companies. I became aware of the demand for employees with big data knowledge and coding skills in the Python programming language. It was a signal that the industry was shifting its tool set from the MapReduce processing model, often coded in Java, to the more abstract Spark processing model, which used more abstract, functional programming languages like Python or Scala. My course at the time used the Java programming language, which aligned with the language students had experienced in prior courses. If I wanted to convert the course to use Python, I needed a painless way to introduce my students to a new language in a short amount of time. In a course that already used two textbooks, I did not want to assign another text to the course. Nor could I afford the time to go over 8 chapters in a textbook before the students became proficient. The redesign's resulting constraints drove me to examine the use of computational notebooks more closely.

3.1 Notebooks Versus Textbooks

The key contrast between computational notebooks and textbooks is one of functionality versus volume. The interactive nature of notebooks allows a student to read an example and then apply it on the same screen. Notebook code cells can contain executable, instructor-provided examples or can be used by the student to code her own examples to be executed. Results from the execution are displayed in the notebook providing immediate feedback. Unexpected results allow the student to correct and retry. Overall, it provides the student with a platform that encourages exploration, while providing instruction close by. This capability contrasts with the strengths of a textbook.

The ability to enhance text content with multiple examples, illustrations and problem sets is a textbook's strength. By presenting the material in different ways, the student develops the deeper understanding he needs to apply the material. While computational notebooks can include illustrations and multiple examples, they become cumbersome when they grow large. Paging through a large notebook could potentially discourage rather than encourage learning.

One of the goals when designing a notebook is to balance these opposing benefits, while being sensitive to the student's experience in using the notebook.

Any measure of students' attitudes concerning the use of notebooks versus textbooks would need to measure these opposing benefits. Did the student feel that the interactivity provided a strong benefit? Did the student miss the additional content provided in textbooks that was not provided in the notebooks?

3.2 Notebook Design and Implementation

I decided to design Jupyter Notebooks to support the students' initial instruction in Python. Once this part of the course was completed, assignments in support of more complex course topics, such as coding Spark applications, would make use of an Eclipse IDE. After identifying the Python constructs needed for the later sections of the course, a Python topics list was developed (see Table 1). Using this list as a guide, three notebooks were developed. Two notebooks took the students through the basics of Python, another covered data structures.

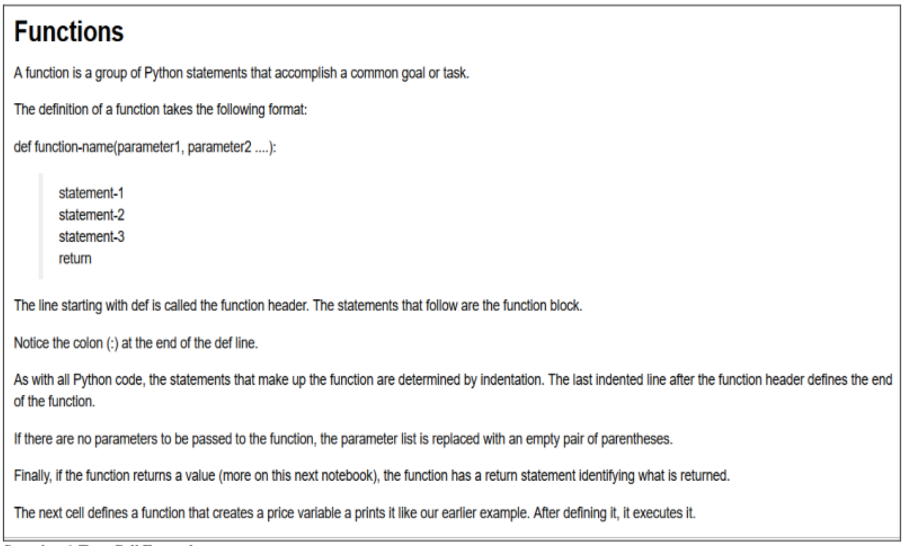
Python Basics	Data Structures
Variables, Data Types, Operators	Sequences, Sets, Dictionaries
Program Flow, Repetition Structures	Generators
Functions	Built-in Functions

Table 1: Python Notebook Topics

Jupyter allows you to incorporate different types of cells in a notebook. These cells can be inserted anywhere in the notebook to support the instruction. Instructional text, coding examples, assignments and autograded cells were developed for each topic. In the following sections, I describe each cell type showing an example from the course notebooks instructing students on the use of functions in Python.

Text Cells for Instructional Content

Text cells provide the instructional content. This can include text or images.



Functions

A function is a group of Python statements that accomplish a common goal or task.

The definition of a function takes the following format:

def function-name(parameter1, parameter2 ....):

> statement-1
> statement-2
> statement-3
> return

The line starting with def is called the function header. The statements that follow are the function block.

Notice the colon (:) at the end of the def line.

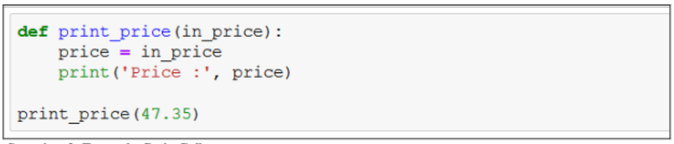
As with all Python code, the statements that make up the function are determined by indentation. The last indented line after the function header defines the end of the function.

If there are no parameters to be passed to the function, the parameter list is replaced with an empty pair of parentheses.

Finally, if the function returns a value (more on this next notebook), the function has a return statement identifying what is returned.

The next cell defines a function that creates a price variable a prints it like our earlier example. After defining it, it executes it.

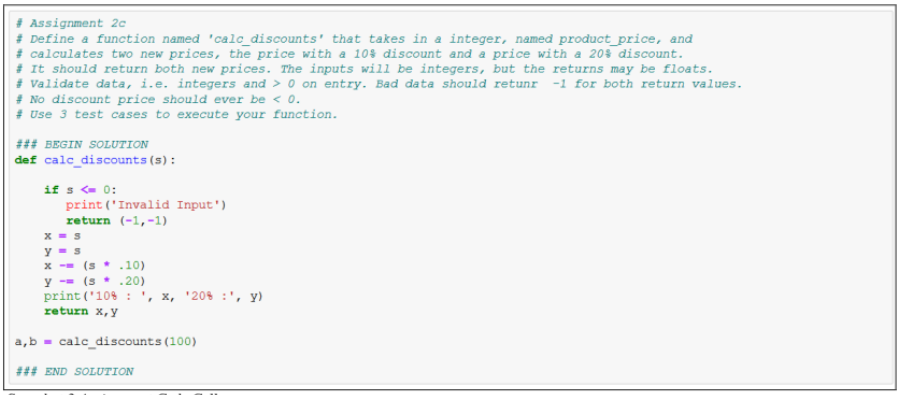
Snapshot 1 Text Cell Example



```
def print_price(in_price):
    price = in_price
    print('Price :', price)

print_price(47.35)
```

Snapshot 2 Example Code Cell



```
# Assignment 2c
# Define a function named 'calc_discounts' that takes in a integer, named product_price, and
# calculates two new prices, the price with a 10% discount and a price with a 20% discount.
# It should return both new prices. The inputs will be integers, but the returns may be floats.
# Validate data, i.e. integers and > 0 on entry. Bad data should retunr  -1 for both return values.
# No discount price should ever be < 0.
# Use 3 test cases to execute your function.

### BEGIN SOLUTION
def calc_discounts(s):

    if s <= 0:
       print('Invalid Input')
       return (-1,-1)
    x = s
    y = s
    x -= (s * .10)
    y -= (s * .20)
    print('10% : ', x, '20% :', y)
    return x,y

a,b = calc_discounts(100)

### END SOLUTION
```

Snapshot 3 Assignment Code Cell

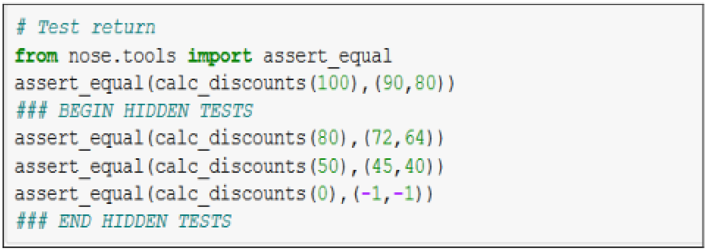
Code Cells for Examples and Assignments

Code cells are used to either provide executable examples or identify assignments to the student. In the former case, instructors can provide students with an executable example, which is working code that demonstrates an earlier instructional text cell. Students can review and execute code to examine how it works. In assignment cells, students are provided with a text writeup providing the details of an assignment. These assignments most often ask the student to write a function, method or class using what they learned in the prior instructional text cell. In this code cell, students can read the assignment requirements, write the appropriate code and execute it.

In defining the assignment cells, instructors include a solution to the assignment for documentation. The section of the cell between Begin Solution and End Solution is excluded when the notebook is distributed to the student.

Autograder Cells

Notebook designers can add cells which can execute and test the results from prior assignment cells in the notebook. This provides two important abilities: it provides instructors the ability to automate the grading of the notebooks, and most importantly, when made available to students, it provides the student instant feedback on the student's solution. After completing an assignment cell, students can execute an autograded cell and get immediate feedback on the correctness of the work. Incorrect assignment cells can be corrected and rechecked, encouraging students to work at a problem until they found the correct answer.



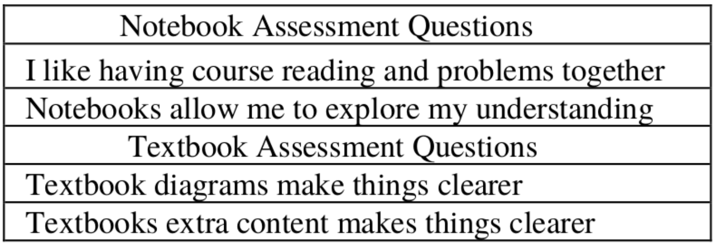
```
# Test return
from nose.tools import assert_equal
assert_equal(calc_discounts(100),(90,80))
### BEGIN HIDDEN TESTS
assert_equal(calc_discounts(80),(72,64))
assert_equal(calc_discounts(50),(45,40))
assert_equal(calc_discounts(0),(-1,-1))
### END HIDDEN TESTS
```

Snapshot 4 Autograder cell

For my big data course, notebooks were used during the third and fourth week of the semester. The notebooks were assigned as pre-work prior to the topic being covered in class (a flipped classroom approach). This required that students read through the notebooks, executed examples, completed any assignments included in the notebooks, and submitted completed notebooks just prior to the topic being covered in class.

4 Research Methodology

The course redesign occurred in the summer of 2017. The redesigned course has been used in both the Spring of 2018 and 2019. To capture student views on notebooks and textbooks, students in the 2018 class completed a survey the week after finishing their last notebook. The survey included 16 questions. Most of the questions looked for feedback on content shortcomings for future improvement. However, 4 of the questions were used to assess how the students evaluated the positive qualities of notebooks and textbooks (Table 2). The answers to those questions were used to develop an evaluation score for notebooks and an evaluation score for textbooks for each student.



Notebook Assessment Questions
I like having course reading and problems together
Notebooks allow me to explore my understanding
Textbook Assessment Questions
Textbook diagrams make things clearer
Textbooks extra content makes things clearer

Table 2: Assessment Questions

4.1 Statistical Analysis

If students did not see positive advantages to using computational notebooks compared to using a more traditional textbook, there would be no benefit to including notebooks in the computer science classroom. Asking students to assess the benefits of notebooks would be a direct reflection on their experiences in using them to learn Python and how effective they felt notebooks were in that process compared to using a textbook. To test this question, a statistical hypothesis test was applied to the survey data to determine if the students perceived these advantages compared to prior experiences with textbooks.

Using the notebook and textbook assessment questions, an evaluation score was generated for both notebooks and textbooks for each student by averaging the scores to the questions (see chart below). Due to the small sample size and that fact that normal distributions could not be guaranteed for the underlying populations, a Wilcoxon Signed Rank Test was applied to the scores. The H 0 in this case, is that computational notebooks functionality is perceived equal or less than the benefits of textbooks to learn the Python programming language.

A rank score above the critical value would signify that the null hypothesis is false, and that the students perceived the characteristics of notebooks more valuable than those of textbooks.

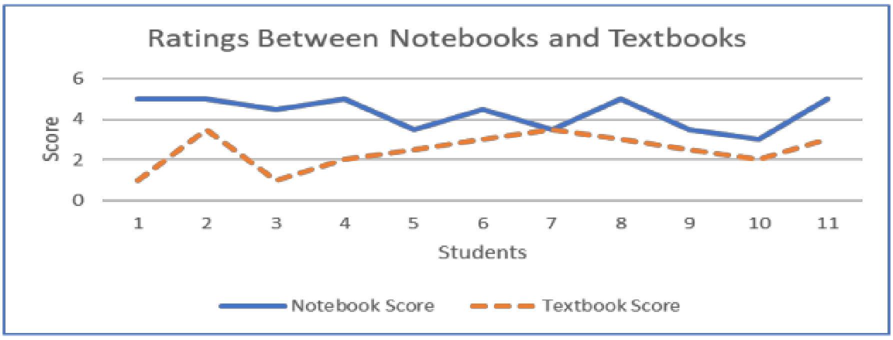


Chart 1. Student Evaluations

5 Results

The calculations for the Wilcoxon Signed Rank are shown in Table 3. The data are the evaluation scores for computational notebooks and textbooks. Both textbooks and notebooks scores are compared. This is a right-tailed test (H_0 : $notebook \mu 1 <=$ textbook $\mu 2$). The critical value for $\alpha = .01$ is W $>=$ 50. The test statistic $W_+ = 55 > W_{.01} = 50$. Therefore, we reject the null hypothesis. This demonstrates a significant statistical difference between the scores of notebooks and textbooks. This is evidence that students rate the characteristics of computational notebooks higher than those of traditional textbooks.

6 Conclusions

Jupyter Notebooks provided my students an interactive environment that they enjoyed using to learn Python. Grades on course assignments later in the semester demonstrated to me that it was an effective learning tool. As a caveat, it is important to note that this is a field study on a limited sample size, not a controlled experiment. The possibility exists that some additional factor impacted the students' evaluation scores on the survey. One possibility is that the cost of textbooks versus a freely supplied notebook drove the students to evaluate textbooks' benefits lower. In any case, the evaluation scores for the notebooks were high, acknowledging the value that the students recognized in them.

Student	Notebook Score	Textbook Score	Sign	Difference	Rank	With Tied Rank	Signed Ranks	
1	5	1		4	10	10	10	
2	5	3.5	+	1.5	4	4.5	4.5	
3	4.5	1	+	3.5	9	9	9	
4	5	2	+	3	8	8	8	
5	3.5	2.5	+	1	1	2	2	
6	4.5	3	+	1.5	5	4.5	4.5	
7	3.5	3.5		0				
8	5	3	+	2	6	6.5	6.5	
9	3.5	2.5	+	1	2	2	2	
10	3	2	+	1	3	2	2	
11	5	3	+	2	7	6.5	6.5	
				W+	55	W-	0	

Table 3: Wilcoxon Signed Rank Calculations

I developed a list of Jupyter pros and cons from my experiences as part of my project documentation that I have listed below. The items are not listed in any order.

An "interactive textbook" that allows further 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
NBGrader provides auto-grading capabilities.	Pro
Each notebook has its own global memory, so careful notebook design is important.	Con
Free open source tool	Pro
Cumbersome Method to Deliver Large Amount of Content	Con
Multiple architecture choices	Pro

Table 4: Jupyter Notebooks Pros and Cons

As the field of big data has progressed tools have been developed to help manage both the programming and analytical complexity. The low-level coding processing model of MapReduce has been replaced with processing engines with higher level, functional programming constructs, like Apache Spark. Similarly, computational notebooks have been introduced to provide a way for data scientists to integrate their thought process and data results, making the hard work of developing and documenting conclusions easier to manage. By incorporating these tools into the classroom with proper instruction, we teach our students about complexity and how best to manage it.

As one last note, publishers and textbook authors are recognizing the benefits of computational notebooks. A new published text introducing computer science students to Python and data science provides Jupyter Notebooks through the book's website [6].

7 Acknowledgments

I would like to thank Drs. Marsha Davis and Garrett Dancik for the initial collaboration on this topic. I also owe a debt of gratitude to Dr. Darius Dziuda for providing some valuable statistical advice. However, any mistakes in applying that advice are purely mine.

References

[1] ACM software system award. https://en.wikipedia.org/wiki/ACM_Software_System_Award.

[2] IPython. https://en.wikipedia.org/wiki/IPython.

[3] Jupyter. https://www.jupyter.org.

[4] NBGrader. https://nbgrader.readthedocs.io/en/stable/.

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

[6] P. Deitel, P.J. Deitel, and H. Deitel. *Intro to Python for the Computer and Data Sciences: Learning to Program with AI, Big Data and The Cloud.* Deitel developer series. Pearson Education, Incorporated, 2019.

[7] Roland DePratti, Garrett M Dancik, Fred Lucci, and Russell D Sampson. Development of an introductory big data programming and concepts course. *Journal of Computing Sciences in Colleges*, 32(6):175–182, 2017.

[8] Juliana Freire, Boris Glavic, Oliver Kennedy, and Heiko Mueller. The exception that improves the rule. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, pages 1–6, 2016.

[9] Ben Glick and Jens Mache. Using jupyter notebooks to learn high-performance computing. *Journal of Computing Sciences in Colleges*, 34(1):180–188, 2018.

[10] Helen H Hu. Using pogil activities to teach cs principles to diverse students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 676–676, 2015.

[11] Michael B Milligan. Jupyter as common technology platform for interactive hpc services. In *Proceedings of the Practice and Experience on Advanced Research Computing*, pages 1–6. 2018.

[12] Bernadette M Randles, Irene V Pasquetto, Milena S Golshan, and Christine L Borgman. Using the jupyter notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pages 1–2. IEEE, 2017.

[13] Penny M Rowe, Haiyan Cheng, Lea Fortmann, Aedin Wright, and Steven Neshyba. Teaching image processing in an upper level cs undergraduate class using computational guided inquiry and polar data. *Journal of Computing Sciences in Colleges*, 34(1):171–179, 2018.

[14] Adam Rule, Aurélien Tabard, and James D Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[15] Scott L Schneberger and Ephraim R McLean. The complexity cross: implications for practice. *Communications of the ACM*, 46(9):216–225, 2003.

[16] Eric Shook, Davide Del Vento, Andrea Zonca, and Jun Wang. Gisandbox: A science gateway for geospatial computing. In *Proceedings of the Practice and Experience on Advanced Research Computing*, pages 1–7. 2018.

[17] Adam A Smith. Teaching computer science to biologists and chemists, using jupyter notebooks: tutorial presentation. *Journal of Computing Sciences in Colleges*, 32(1):126–128, 2016.

Learning Assembly Language through Visual Simulation*

Kamen Kanev[1], Mokhtar Aboelaze[2], Reneta P. Barneva[3]
[1]Research Institute of Electronics
Shizuoka University
Hamamatsu, Shizuoka 432-8011, Japan
`kanev@shizuoka.ac.jp`
[2]Lassonde School of Engineering
York University
Toronto, ON M3J1P3, Canada
`aboelaze@cse.yorku.ca`
[3]School of Business
SUNY Fredonia
Fredonia, NY 14063, USA
`reneta.barneva@fredonia.edu`

Abstract

This work focuses on the design and development of a specialized educational environment for assembly language learning support involving different Instruction Set Architectures (ISA's). We discuss in particular the implementation of the RISC-V Visual Simulator (RVS) and the potential pedagogical benefits of its employment in Computer Organization and Architecture courses at various educational institutions.

1 Introduction

One of the ABET requirements [4] is to include in the computer science curriculum "*exposure to computer architecture and organization, information management, networking and communication, operating systems, and parallel and*

*Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

distributed computing." No surprise, therefore, that intermediate or upper level courses on *Computer Architecture and Organization* are offered in almost all undergraduate programs in the USA and worldwide.

On the one hand, these courses incorporate key hardware topics such as fundamental circuits and CPU design, memory organization and caching, interrupts and exceptions handling, I/O devices integration, and others. On the other hand, they provide a detailed coverage of the respective Instruction Set Architecture (ISA) involving key software topics such as machine language fundamentals, assembly language programming concepts, translation of assembly instructions into machine language, and so on.

While the theoretical aspects of both, the hardware and the software components of the courses could be covered at lecture time, all practical aspects are usually studied in a laboratory environment. The practical aspects of the design of electronic circuits and systems, and their testing and verification are often taught through hardware description languages such as VHDL or Verilog. In some cases, access to physical hardware components matching the adopted ISA and allowing native assembly programming and machine code execution could also be provided. In other cases specialized software that emulates the ISA adopted in the course is used instead for cross compiling and/or simulated execution of the ISA machine code.

Computer Architecture and Organization is one of the foundational undergraduate courses in computing that is taught not only to computer science students, but also to computer information systems, network security, and computer and electrical engineering majors. Despite the immense changes in information and communication technology that lead to new courses on topics like cloud and mobile computing to game development to data analytics to block-chain technology, the essence of the *Computer Architecture and Organization* courses remained stable over the years. Content updates were mostly related to the specific technology developments and the advancements in novel instruction set architectures thus allowing to keep the course in line with the latest trends in the industry.

The contemporary computer science students, belonging predominantly to the Generation Z and being digital natives, are often not so interested in why the technology functions in one way or another, but rather in how to make it function in the way they want. In fact, many of those students perceive the *Computer Architecture and Organization* material as less exciting when compared to the content of other modern technology courses with clearer paths for immediate applications. Some additional accommodations would, therefore, be needed to further engage the students and stimulate their *Computer Architecture and Organization* studies. Given the mix of hardware and software components involved, we believe the learning outcomes in this particular field of studies could be significantly improved by employing highly focused visual simulation tools that attract and retain students' attention more effectively.

In many universities, especially in view of the trends to reduce the cost of higher education, the same *Computer Architecture and Organization* course is offered to students majoring in various disciplines, such as computer science, computer engineering, computer information systems, and network and computer security [13, 11, 14]. This approach, however, poses some didactic challenges since depending on the discipline students may have different hardware/software backgrounds thus requiring the emphasis of the material to be put on different topics. For example, students majoring in computer information systems may have taken courses on Visual Basic and may feel comfortable working with an integrated interactive development environment rather than coding, while students in computer science prefer writing code in a programming language and would not have difficulties to grasp the idea of assembly language instructions.

One of the most widely used textbooks for this course is "Computer Organization and Design: The Hardware/Software Interface" by Patterson and Hennessy [7]. It has been adopted by large renowned schools, such as Stanford University, Cornell, Georgia Tech, Johns Hopkins University, and Northeastern University, as well as by colleges, including colleges in Northeast, such as SUNY New Paltz, St. Bonaventure University, and Villanova University in the USA and by many other universities worldwide. The earlier editions of the book [7] were based on the MIPS ISA while the more recent ones are geared towards RISC-V [9] and ARM [8] ISA's, respectively. On the one hand, this provides convenient specialization and educational support towards one or another instruction set. On the other hand, it is a challenging task to teach a course in which some of the students majoring in one discipline may need to use one of the ISA's, while another group of students need to use the other one. Ideally, there should be a unified software interface and respective support for all the instruction sets, but in practice this is not the case.

In this paper we discuss the design and development of a multi-platform visual simulator, configurable for different architectures (e.g. MIPS, RISC-V, and ARM) and consider its benefits to the pedagogical process. The simulator has been successfully used in *Computer Architecture and Organization* courses taught at York University were it allowed instructors to handle more effectively the programming-related practical components. We believe that this software would facilitate both the instructors offering courses on a multiplicity of ISA's and the students aspiring to learn the specifics of various instruction sets.

The paper is structured as follows: in the next section we consider the concept of the proposed pedagogical approach. In Section 3 we discuss the design and organization of the developed visual simulator. In Section 4 we share our experience from the implementation and use of the visual simulator in the *Computer Architecture and Organization* courses at York University. Finally, we conclude with a discussion and plans for further work.

2 Proposed Pedagogical Approach

As described above, the course on Computer Organization and Architecture consists of two parts – a theoretical part and a practical part. The theoretical part is delivered in the form of lectures and can accommodate large groups of students. During the practical part, the students have hands-on activities and are required to solve a number of problems in a supervised lab environment. The practical part is more interactive so the groups should be smaller, e.g., shall not exceed the size of the sections in regular programming courses.

When it comes to a general computer architecture course, choosing between two different architectures such as RISC-V and ARM, for example, is not a trivial task, especially when the needs of students majoring in different disciplines have to be considered. From pedagogical perspective it would be good, therefore, if the course could be offered for different architectures, possibly at the same time, so that the students might decide individually what would be most beneficial for them. The students could then be assigned to different lab sections depending on their interest or the need to cover a specific instruction set – MIPS, RISC-V, or ARM.

Our idea is, therefore, to have slightly different, but compatible versions of the course that will be shaped after the three currently available editions of the textbook "Computer Organization and Design: The Hardware/Software Interface". The architecture design part of the course can be fully covered by Verilog [3] in a consistent way despite of the differences between the MIPS, RISC-V, and ARM architectures. With respect to the assembly language part, the earlier editions of the book had an appendix covering a freely available MIPS simulator [6, 5]. The two more recent editions are based on the RISC-V [9] and the ARMv9 [8] instruction sets thus providing for more choice and new pedagogical options, but they are lacking respective simulators coverage. Hence, in order to employ this approach, a unified set of properly documented software simulation tools suitable for educational use will be needed.

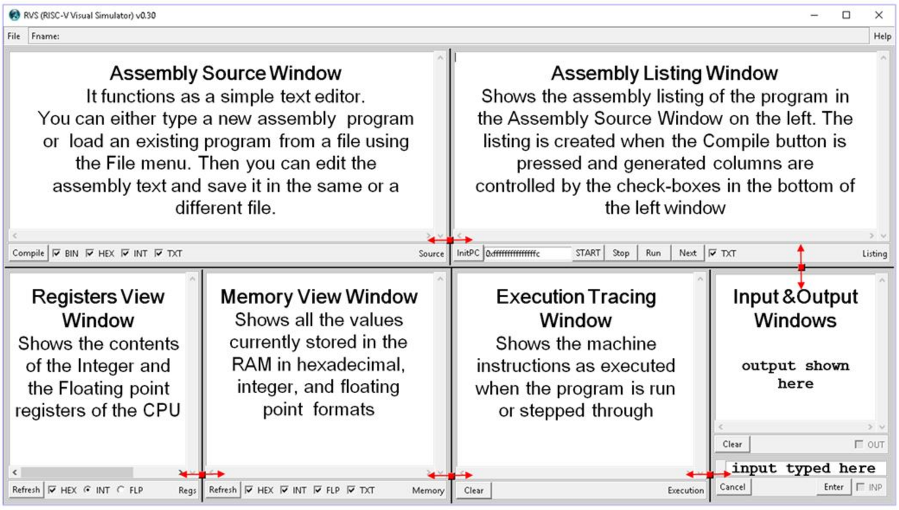
Indeed, while professional software development tool chains are available both for the RISC-V [2] and the ARM instruction set architectures [1], they have been independently developed and thus adhere to different policies, each of them optimized for the specific architecture. In an educational setup where efficiency is of lesser concern, however, a unified design policy becomes a viable option.

3 The Developed Visual Simulator

3.1 Design

With the pedagogical goal described above in mind, we have designed a new visual environment for assembly programming and simulated code execution for different architectures. Our initial implementation target – the RISC-V architecture – has

been tested with all the sample code from the RISC-V edition of the textbook [9]. A snapshot of the GUI of the visual simulator with brief explanatory texts dedicated to each of the six main windows is shown in Figure 1.



Assembly Source Window
It functions as a simple text editor.
You can either type a new assembly program or load an existing program from a file using the File menu. Then you can edit the assembly text and save it in the same or a different file.

Assembly Listing Window
Shows the assembly listing of the program in the Assembly Source Window on the left. The listing is created when the Compile button is pressed and generated columns are controlled by the check-boxes in the bottom of the left window

Registers View Window
Shows the contents of the Integer and the Floating point registers of the CPU

Memory View Window
Shows all the values currently stored in the RAM in hexadecimal, integer, and floating point formats

Execution Tracing Window
Shows the machine instructions as executed when the program is run or stepped through

Input & Output Windows

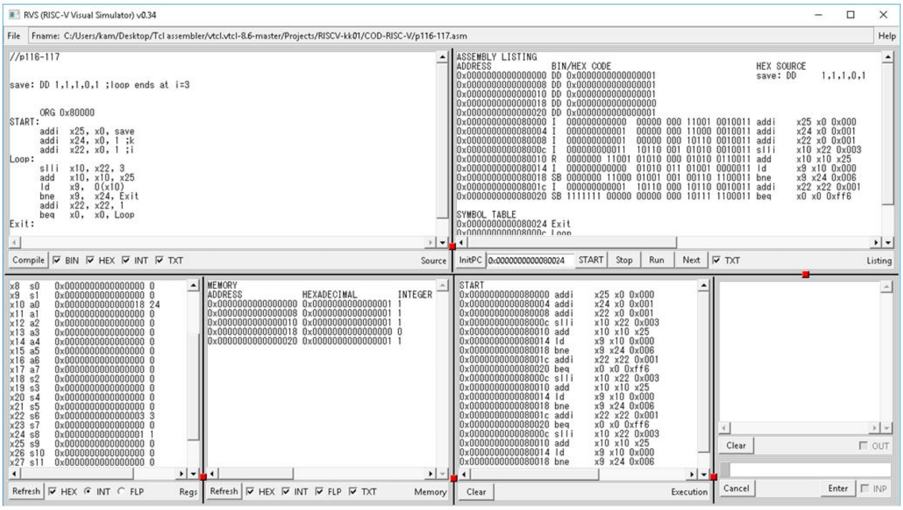
```
output shown
here
```

Figure 1: The GUI of the experimental RISC-V Visual Simulator

The use of the interactive assembly language programming and architecture simulation environment through the above GUI is highly intuitive. Any text loaded or directly written into the Assembly Source Window can be compiled at a press of a button and the outcome is immediately shown in the Assembly Listing Window. The execution of the resulting program can then be controlled through the Start, Stop, Run, and Next controls at the bottom of the Assembly Listing Window. The progress and outcome of the program execution is shown in real time in the Execution Tracing Window. In addition, the two windows at the bottom-left of the GUI provide interactive views of the current state of all simulated registers and RAM. Finally, the window at the bottom-right of the GUI is used for I/O communications with the running code.

In Figure 2 we show another snapshot of the GUI with a complete example incorporating the text of a sample assembly program, its compilation and the output listing, the trace of its execution, and the resulting values stored in some of the registers and in RAM.

The implementation is table driven and its retargeting to other architectures is currently in progress. We are planning to cover the LEGv9 subset of the ARMv9 instruction set as described in the ARM edition of the textbook [8] as well as the MIPS instruction set from the earlier editions of the textbook.


RVS (RISC-V Visual Simulator) v0.34
File
Fname: C:/Users/kam/Desktop/Tcl assembler/vtcl.vtcl-8.6-master/Projects/RISCV-kk01/COD-RISC-V/p116-117.asm
Help
Compile ☑ BIN ☑ HEX ☑ INT ☑ TXT Source
InitPC 0x0000000000080024 START Stop Run Next ☑ TXT Listing
Refresh ☑ HEX ◉ INT ○ FLP Regs
Refresh ☑ HEX ☑ INT ☑ FLP ☑ TXT Memory
Clear Execution
Clear ☐ OUT
Cancel Enter ☐ INP

```
//p116-117


save: DD 1,1,1,0,1 ;loop ends at i=3


      ORG 0x80000
START:
      addi  x25, x0, save
      addi  x24, x0, 1 ;k
      addi  x22, x0, 1 ;i
Loop:
      slli  x10, x22, 3
      add   x10, x10, x25
      ld    x9,  0(x10)
      bne   x9,  x24, Exit
      addi  x22, x22, 1
      beq   x0,  x0, Loop
Exit:
```

```
ASSEMBLY LISTING
ADDRESS             BIN/HEX CODE                                       HEX SOURCE
0x0000000000000000 DD 0x0000000000000001                               save: DD    1,1,1,0,1
0x0000000000000008 DD 0x0000000000000001
0x0000000000000010 DD 0x0000000000000001
0x0000000000000018 DD 0x0000000000000000
0x0000000000000020 DD 0x0000000000000001
0x0000000000080000 I  000000000000  00000 000 11001 0010011 addi    x25 x0 0x000
0x0000000000080004 I  000000000001  00000 000 11000 0010011 addi    x24 x0 0x001
0x0000000000080008 I  000000000001  00000 000 10110 0010011 addi    x22 x0 0x001
0x000000000008000c I  000000000011  10110 001 01010 0010011 slli    x10 x22 0x003
0x0000000000080010 R  0000000 11001 01010 000 01010 0110011 add     x10 x10 x25
0x0000000000080014 I  000000000000  01010 011 01001 0000011 ld      x9 x10 0x000
0x0000000000080018 SB 0000000 11000 01001 001 00110 1100011 bne     x9 x24 0x006
0x000000000008001c I  000000000001  10110 000 10110 0010011 addi    x22 x22 0x001
0x0000000000080020 SB 1111111 00000 00000 000 10111 1100011 beq     x0 x0 0xff6

SYMBOL TABLE
0x0000000000080024 Exit
0x000000000008000c Loop
```

```
x8  s0   0x0000000000000000 0
x9  s1   0x0000000000000000 0
x10 a0   0x0000000000000018 24
x11 a1   0x0000000000000000 0
x12 a2   0x0000000000000000 0
x13 a3   0x0000000000000000 0
x14 a4   0x0000000000000000 0
x15 a5   0x0000000000000000 0
x16 a6   0x0000000000000000 0
x17 a7   0x0000000000000000 0
x18 s2   0x0000000000000000 0
x19 s3   0x0000000000000000 0
x20 s4   0x0000000000000000 0
x21 s5   0x0000000000000000 0
x22 s6   0x0000000000000003 3
x23 s7   0x0000000000000000 0
x24 s8   0x0000000000000001 1
x25 s9   0x0000000000000000 0
x26 s10  0x0000000000000000 0
x27 s11  0x0000000000000000 0
```

```
MEMORY
ADDRESS            HEXADECIMAL        INTEGER
0x0000000000000000 0x0000000000000001 1
0x0000000000000008 0x0000000000000001 1
0x0000000000000010 0x0000000000000001 1
0x0000000000000018 0x0000000000000000 0
0x0000000000000020 0x0000000000000001 1
```

```
START
0x0000000000080000 addi    x25 x0 0x000
0x0000000000080004 addi    x24 x0 0x001
0x0000000000080008 addi    x22 x0 0x001
0x000000000008000c slli    x10 x22 0x003
0x0000000000080010 add     x10 x10 x25
0x0000000000080014 ld      x9 x10 0x000
0x0000000000080018 bne     x9 x24 0x006
0x000000000008001c addi    x22 x22 0x001
0x0000000000080020 beq     x0 x0 0xff6
0x000000000008000c slli    x10 x22 0x003
0x0000000000080010 add     x10 x10 x25
0x0000000000080014 ld      x9 x10 0x000
0x0000000000080018 bne     x9 x24 0x006
0x000000000008001c addi    x22 x22 0x001
0x0000000000080020 beq     x0 x0 0xff6
0x000000000008000c slli    x10 x22 0x003
0x0000000000080010 add     x10 x10 x25
0x0000000000080014 ld      x9 x10 0x000
0x0000000000080018 bne     x9 x24 0x006
```

Figure 2: A snapshot of the GUI with a complete assembly program example

3.2 Multi-Architecture Support

While the adopted strategy is clear, there are many subtle details that need to be taken into account and properly tackled at the design and implementations stages. The different editions of the book, for example, employ different commenting schemes denoted by a semicolon (;), a pair of slashes (//), and a number sign (#). The number sign, however, is also employed for marking of the immediate values in the ARM edition which precludes its use as a universal comment marker for all architectures. Another complication with the ARM edition is that instead of parentheses, square brackets are used to denote the memory addressing modes. While this may be consistent with the richer set of addressing modes in the ARM architecture, the LEGv9 instruction subset which is confined to unscaled addressing could possibly be tuned to mimic more closely the assembly syntax employed in the other editions of the book. With respect to the syntax of the entered machine language instructions, all three architectures employ fixed order operands with minor modifications. The most notable differences are in the way different addressing schemes are denoted in the different architectures. The MIPS and RISC-V instructions sets, for example, refer to memory addresses by providing an offset value as an immediate operand followed by an index register enclosed in parentheses as shown in Figure 3 (a) and (b) respectively. The LEGv9 instructions subset of ARMv9, on the other hand, refers to memory addresses by providing in square brackets a register followed by a comma and an immediate offset value preceded by the number sign (#) as shown in Figure 3 (c).

```
(a) lw     $t4, 8($t2)   //MIPS: load word instruction
(b) ld     x1,  8(x2)    //RISC-V: load double-word
(c) LDUR   X9, [X2, #8] //LEGv9: load double-word unscaled
```

Figure 3: Memory addressing in the MIPS (a), RISK-V (b), and ARM (c) architectures

Note that in the standard operand order, applicable to most other instruction, the register operands come first from left to right while the immediate operand is the last one on the right as shown in Figure **??**.

```
addi  $t1, $0,  1      //MIPS: add immediate instruction
addi  x1,  x0,  1      //RISC-V: add immediate instruction
ADDI  X1,  X0, #1      //LEGv9: add immediate instruction
```

Figure 4: Standard operand order for the majority of the instructions

Both in the MIPS and the RISC-V assembly instruction sets, therefore, the second register and the immediate value are swapped in the case of memory addressing. We tackle this issue through defining a canonical assembly instruction representation that is consistent over all supported architectures. It is implemented as an instruction code followed by a list of operands with possible qualifiers. Simple instructions as the ones shown in Figure 3 can be easily rewritten in canonical form adhering to unified operand syntax as shown in Figure 5.

```
lw       r12 r2 8 //MIPS: load word instruction
ld       r1  r2 8 //RISC-V: load double-word instruction
ldur     r9  r2 8 //LEGv9: load double-word unscaled
```

Figure 5: The instructions from Figure 3 rewritten in canonical form

List elements are separated by white space and automatically enclosed in curly braces in case a white space itself is included in a list element. The resulting list structure is a key component of the architecture simulation engine and is used in the process of instruction interpretation and execution.

The source code in canonical form shown in the Assembly Listing Window in Figure **??** displays all constants in hexadecimal. It is followed on the right by the source code in standard form with all constants converted to decimal. For completeness, the original source including the comments is appended last. The different representations of the source code in the listing can be inspected by scrolling with the slider on the bottom of the window. In this way students can see the values of the employed constants both in decimal and hexadecimal and can use the canonical form as a reference and for comparisons while learning the native assembly code of the specific ISA.

3.3 Binaries and Virtual Memory

While the binary representation of all machine instruction is not strictly required for the proper architecture simulation with the above approach, our system generates bit-accurate machine code which can be used both for educational purposes and for verifying the simulations outcomes. The generated binaries have been successfully matched to the sample machine instructions provided in the textbook [9] and have been compared to object files generated by the RISC-V tool chain [2].

Our implementation employs a virtual memory model implemented through associative arrays which provides access to the entire 64-bit addressing space irrespectively of the physical limitations of the host computing system. We support the following memory initialization and access modes that are switchable as required by the concrete pedagogical objectives of each programming exercise:

- Virtual memory is treated as initialized to zero. Access to uninitialized memory cell silently returns zero. Memory cells are automatically allocated at writing of non-zero values. Writing of zero values releases previously allocated memory cells.
- Virtual memory is treated as non-initialized. Access to an uninitialized memory cell returns an error. Memory cells are automatically allocated at writing of any value.
- Virtual memory is treated as non-initialized. Access to an uninitialized memory cell silently returns a random value. Memory cells are automatically allocated at writing of any value.

Note that the last memory initialization and access mode is closer to the hardware although the first two modes usually produce better learning outcomes.

4 Practical Implementation

We implemented the Visual Simulator described above at York University, where a standard computer architecture course has been on the undergraduate computer science curricula at for over 20 years. While initially, in the early 90s, various textbooks were used [12, 10] the course was eventually streamlined to follow the book of Patterson and Hennessy [7]. The course is offered as a strong synergy of theory and practice and has continuously being updated. As a result, it has been in high demand for all these years with averages of over 1000 students per academic year.

One part of the course is offered in a classroom setting, while another part, devoted to solving practical problems, is offered in a computer laboratory setting. The labs are divided into two parts namely MIPS and RISC-V assembly programming based on the SPIM and the RVS simulators respectively and circuit and CPU design based on Verilog [3]. With respect to the assembly component of the labs diverse pedagogical goals are set since the course is being offered to both engineering students and computer science (CS) students which have different levels of programming skills and different needs.

At undergraduate level the quick and easy access to the essential components of the respective programming environments and tools are of particular importance. To ensure such access the RISC-V Visual Simulation has been made available on a number of university servers and workstations both for direct and remote use. Students can also download the RVS and use it on their own machines. The RVS executables that are currently available for Linux, Windows, and MacOS do not require any specific installation and can be run immediately after downloading.

5 Conclusion and Further Work

Based on our observations, personal communications with the instructors, and the feedback obtained from students, we felt a strong positive sentiment towards using the RISC-V Visual Simulator in the courses on *Computer Architecture and Organization*. We believe that the highly interactive and easy to learn RVS interface helped many students to quickly grasp the basics of assembly language and start making progress despite their limited prior coding experience.

The instructors were also pleased with the software and reported that it allowed an easier way of demonstrating the concepts, models and algorithms. From organizational point of view, the course was structured more efficiently, concentrating the resources for the activities that needed more interactivity.

Device I/O, system calls, and interrupt handling are important components of the assembly language support. A simulated I/O device and a set of system calls for printing and reading the major data types are embedded in the current RVS. Interrupt based I/O device support is planned for the next major RVS release.

While the current RVS implementation is confined to the RISC-V architecture, the Visual Simulator design allows for multi-architectural support. Our next step will be to extend the architecture support to LEGv9 subset of ARMv9 as in [8].

Although the Visual Simulator was developed for courses on *Computer Architecture and Organization*, it could also be used in courses on networks, security, and/or software engineering where work in assembly language is often needed. We, therefore, plan to test the RVS and our pedagogical approach with different courses at other schools.

6 Acknowledgements

Part of the work reported in this article is supported by a Cooperative Research Project of the Research Center for Biomedical Engineering.

References

[1] ARM Developer. https://developer.arm.com/.

[2] RISC-V Foundation Software Tools. https://riscv.org/software-status/.

[3] Verilog Resources. http://www.verilog.com/.

[4] ABET. Computing accreditation commission, criteria for accrediting computing programs. Effective for Reviews during the 2019-2020 Accreditation Cycle.

[5] James Larus. A MIPS32 simulator. http://spimsimulator.sourceforge.net/.

[6] James Larus. Assemblers, linkers, and the SPIM simulator. 2014.

[7] David A Patterson and John L Hennessy. *Computer Organization and Design: The Hardware/Software Interface.* 2014.

[8] David A Patterson and John L Hennessy. *Computer Organization and Design ARM Edition: The Hardware Software Interface.* Morgan kaufmann, 2017.

[9] David A Patterson and John L Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. 2017.

[10] William Stallings. *Computer Organization and Architecture 2nd ed.* Macmillan, 1990.

[11] SUNY Polytechnic Institute. 2019-2020 undergraduate catalog. https://webapp.sunypoly.edu/undergrad-catalog-2019-2020/.

[12] Andrew S. Tanenbaum. *Structured Computer Organization (3rd ed.)*. Prentice-Hall, Inc., 1990.

[13] University at Buffalo. Undergraduate degree course catalog 2019-20. https://catalog.buffalo.edu/.

[14] York University. Supplemental calendars for EECS programs and courses. http://eecs.lassonde.yorku.ca/wp-content/uploads/Undergrad/Supplemental%20Calendars/.

Cooperative Learning in Computer Science: Jigsaw Activity[*]

Anastasia Kurdia
Department of Computer Science
Tulane University
New Orleans, LA 70118
akurdia@tulane.edu

Abstract

Jigsaw is a cooperative learning approach in which a topic is broken into several segments, each student learns one segment of a topic individually, then students work in small groups to review what they learned, to teach their segments to each other, and to construct a complete picture of the topic. This paper aims to help computer science instructors who are interested in developing jigsaw activities by providing a model activity and by sharing the experience of implementing it in the classroom.

1 Introduction

Jigsaw is a cooperative learning approach in which a topic is broken into several segments, each student first learns one segment of the topic individually (Preparation phase), then students work in small groups to review what they have learned (Review phase), to teach their respective segments to each other (Present phase), and to construct a complete picture of the topic (Reflect phase).

Jigsaw was originally developed by Elliot Aronson in the early 1970s to increase collaboration and weaken racial tensions in desegregated schools. Since then numerous research studies have demonstrated advantages of jigsaw approach, such as improved communication, listening and problem-solving skills, self-esteem, tolerance, and compassion, to name a few [1].

[*]Copyright ©2020 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

A typical jigsaw exercise usually spans one class period (or even a portion of a class period) making it a low-stakes easy-to-adopt classroom activity, in comparison with other active learning tools (such as flipped classroom, peer instruction [6], process-oriented guided inquiry learning [3]) that might require restructure of a large portion of the course or the entire course. Consequently, it is an ideal approach to try for graduate teaching assistants and new instructors who are just starting to incorporate active learning into their courses, or for experienced instructors who would like to add variety to their classroom without major effort.

The idea of using jigsaw exercises in a computer science (CS) classroom has been proposed previously [2, 4] but it has not found widespread adoption. One of the main reasons might be linear or hierarchical nature of CS material. It is more common to see CS topics build on one another rather than to see a set of small topics that could be studied independently, which makes it hard to identify the parts that can become jigsaw "pieces". Another reason is the concern of the instructors about its implementation.

The goal of this paper is to help educators who are interested in developing and using jigsaw activities in their instruction. Section 2 provides a model jigsaw exercise that is ready for implementation in an introductory CS course. Section 3 describes the experience of implementing that exercise in a classroom of a medium-size selective residential four-year university and generalize it to other settings and other exercises. The paper is concluded in Section 4 with additional recommendations on developing jigsaw exercises. Appendices 5, 6 provide example student-facing materials that can serve as starters for developing jigsaw activities.

2 Implementation example: binary tree traversals

This section describes Preparation, Review, Present, and Reflect phases of an in-class jigsaw activity on binary tree (BT) traversals that was designed for an introductory computer science class. The three tree traversal algorithms (preorder, in-order, and postorder) naturally become individual segments of jigsaw. The class period in which activity takes place is the second session on trees (following the one with introduction of trees, binary trees, and binary search trees).

2.1 Preparation phase

At the preparation phase, the instructor identifies a printed or an electronic resource describing binary tree traversals in the terminology and details appropriate for the course, and assigns homework requiring to study that resource, for example, ch. 13 in "Data Structures and Algorithms Using Python" [5] and the homework assignment text provided in Appendix 5.

2.2 Review phase

At the start of the class, students who studied the same traversal algorithm are asked to get together and confirm their understanding of their algorithm with other experts on this same algorithm: they're asked to practice their presentations, put examples and pseudocode for their traversal algorithm on the board or on a blank piece of paper, prepare to reason about big-Oh analysis, etc.

2.3 Present phase

After Review phase is finished (or nearly finished), the students are split into small groups so that each group contains one or two students who know a particular algorithm, and all three algorithms are represented in a group. Students are then asked to present their traversal method to those in the group who haven't studied it.

2.4 Reflect phase

Lecturer offers summary and discussion: a one-line verbal summary of each algorithm, followed by the presentation of the actual code for the traversal (if students only used pseudocode in their explanations), discussion of running time, and comparison and contrast of usage of the traversal algorithms.

Students are then asked to solve a few exercises on BT traversals to solidify their mastery of the content and to identify gaps in their understanding (a sample problem handout is provided in Appendix 6). The instructor then discusses correct solutions and answers students' questions. This activity takes almost entire 50-minute class period. If time remains, the instructor solves more tree problems with the students.

3 Discussion and implementation details

This BT jigsaw activity was employed in five incarnations of the course and proved to be more engaging, dynamic, fun, and most importantly, more effective compared to a lecture on binary tree traversals. Similar activities were designed and ran in other courses where several related topics need to be covered (e.g. different CPU scheduling algorithms or locks using different hardware primitives in operating systems course). It worked well in all those instances, even when no other active learning techniques were employed in the course, and when other homework reading assignments or participation were not assessed, formally or informally. In final course evaluations, students repeatedly mentioned how these activities made them feel confident by allowing them to master a digestible portion of the material.

3.1 Preparation issues

The success of the activity depends on the preparation of the majority of the students. The following steps might help facilitate preparation:

- Advance notice. At the end of the preceding class session, I mention that the next session will be experimental and will require preparation (reading a short chapter and preparing a 5-minute presentation); and that different students will prepare different topics. I also mention where to find learning materials and topic distribution.
- Emphasis on a cooperative nature of this assignment. I highlight that this exercise is like a potluck - a communal meal where every guest contributes one dish. While the individual effort is small, collectively the guests compose a great dinner. If everyone brings a small piece of new knowledge for the next class, we all will have an intellectual feast; if too many people show up unprepared, everyone will starve, just like at a real potluck.
- Reminder(s). I send a reminder email a day before the class and make an assignment on learning management system (LMS) that has reminder functionality.

With two reminders most of the students come prepared. Few high-achieving students prepare excessively well using PowerPoint presentations with videos, animations, and code. In order not to discourage such preparation I ask to have those excellent materials and post on LMS for all students to see. Few students come unprepared. I distribute them evenly between the groups, asking them to be critics and "devil's advocates" who should ask all questions necessary to clarify the material. Students are not given a lot of information about the class structure in advance, except for the information that's already provided in the assignment. If students press for details ("Are we *really* going to listen to twenty presentations on the same topic?"), I say that they will present in groups. When a student needs to miss a class, I tell them to study all jigsaw topics on their own. No proof of work is required but some of those students still choose to send their notes as evidence of learning the topics. No in-class participation grade or points are assigned for this exercise (but it might be appropriate to do so in then case when formal participation assessment is built into the course).

My main worry when deciding to adopt jigsaw method was the flow of the class if too many students come unprepared. To eliminate this stress, I prepare the standard lecture notes and tell the students ahead of time that the activity is experimental and the class may need to be stopped and converted to a lecture if the experiment doesn't go smoothly.

When Prepare phase won't work In a classroom where the majority of students can't be expected to prepare ahead of time, Prepare phase can be substituted by in-class Read phase, in which the students read the explanation of one topic in class. They then review it with the group, present it to others, and reflect together with the instructor, just like in the original version of jigsaw activity.

3.2 In-class issues

Scaling A natural question is how many students can participate in a jigsaw activity. My experience pertains to classrooms of 10-50 people. In a larger class, an additional consideration needs to be given to methods of topic distribution, and to grouping during Review and Present phases. With those issues addressed, jigsaw can still work as a small-group activity with all the benefits of required individual participation.

Grouping How students are partitioned into groups is not essential for the activity as long as the grouping process is clear and intentional; mere "find yourself a partner" is too slow and chaotic. Next I describe the grouping method that I employ.

For Prepare phase topic distribution, the class roster is sorted alphabetically and is split into three even parts.

For Review phase grouping, students physically move to a corner of the room designated as a meeting point for a particular algorithm (in a large class, several groups on the same algorithm might be necessary).

To group the students for the Present phase, I walk around during Review phase and give each student a count (1,2,3) in their algorithm group, with the intention for every group to have an expert on every one of three topics. Despite splitting the class in thee even parts in the Prepare phase, in-class Review groups are rarely the same size, because of absences, coming unprepared, etc. I hence use the size of the smallest Review group as the max value for the count and wrap around to 1,2,3, etc. in larger groups. Then I designate a place in the classroom where all 1s should meet, another place where all 2s should meet, and so on. I use index cards with numbers or write group numbers on the board or on the wall (simply pointing hasn't worked well).

Note that this grouping method does not take into account students' academic levels or established social groups, and therefore facilitates student interaction and promotes communication in diverse groups outside of students' regular circle.

Pace Different Present groups progress at a different pace. Short questions or simple exercises can keep fast groups working while they're waiting for the rest of the class to finish presentations. For instance, BT traversal exercise can include questions such as:

- Use the pseudocode to write the actual code of the traversals.
- What is the running time for each of these algorithms?
- Modify your algorithm to find a number of nodes in the tree/a node with a given key/a sum of values in the tree.
- When would each algorithm be used? Why would someone select one algorithm over the other traversal algorithms?

Some groups are very slow and need to be stopped when the majority of the class has finished their presentations.

Getting attention This class is loud. To be able to get the attention of the talking students, it is necessary to agree on a signal for silence at the beginning of the class. This signal can be flipping the lights or asking everyone who is hearing the clap to join in a steady, synchronized clap or asking everyone who is seeing a raised hand to raise their hand and to stop talking.

4 Concluding remarks

When several related topics are covered in a course, jigsaw activity is an effective and active way for students to master those topics. For instance, in introductory operating systems class, the topics of different CPU scheduling algorithms and implementations of software locks using different hardware primitives are good candidates for jigsaw. Jigsaw activities might also be designed for element-based and index-based iteration in Python in introductory CS course; linked list-based and array-based implementations of abstract data types or hashing with different collision resolution strategies in data structures course; queue and stacks issues or different self-balancing trees in algorithms course. It is also reasonable to consider a jigsaw-like approach for project-oriented courses where each team member researches and learns a piece of technology and teaches it to other team members.

Jigsaw activities are small in scale and are a great tool to try for the educators who are beginning to develop teaching materials and to incorporate active learning methods in their instruction. Compared to a traditional lecture, the success of jigsaw activity hinges on students' preparation rather that of the instructor. Preparing an equivalent lecture material can help eliminate the instructor's uncertainty and stress. A variation of jigsaw in which the students prepare right in class can be effective in the cases when it's unrealistic for the students to prepare in advance.

References

[1] The jigsaw classroom, 2020 (accessed February 2, 2020). https://www.jigsaw.org.

[2] Maria Kordaki and Haris Siempos. The jigsaw collaborative method within the online computer science classroom. In *Proceedings of the 2nd International Conference on Computer Supported Education*, volume 2, pages 65–72, January 2010.

[3] Clifton Kussmaul. Process oriented guided inquiry learning (POGIL) for computer science. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, page 373–378, 2012.

[4] Laurie Murphy, Kenneth Blaha, Tammy VanDeGrift, Seven Wolfman, and Carol Zander. Active and cooperative learning techniques for the computer science classroom. *J. Comput. Sci. Coll.*, 18(2):92–94, December 2002.

[5] Rance D. Necaise. *Data Structures and Algorithms Using Python.* John Wiley and Sons, 2010.

[6] Leo Porter and Daniel Zingaro. Peer instruction in CS: Research and experience. *J. Comput. Sci. Coll.*, 28(6):11–13, June 2013.

5 Appendix: Student-facing homework assignment for binary tree traversals in-class jigsaw activity

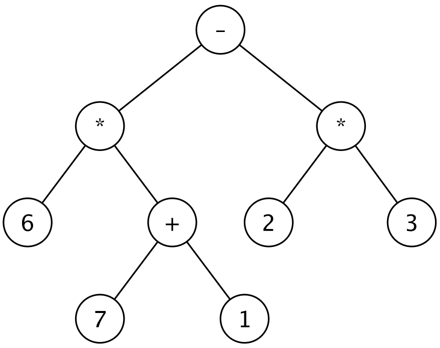
For the next class on Friday, October 18 read your algorithm for tree traversal (ch. 13.2.3, and you are welcome to use any other sources). Prepare to explain the algorithm in front of the class in 5 minutes, to draw an example tree and illustrate how the algorithm proceeds on that tree, and to present Python code or pseudocode for implementing your algorithm.

You may prepare and use Powerpoint (you'll need to print the presentation in advance) or use handwritten notes to help you, but don't plan to read from them.

- If your last name starts with A-H, you're presenting preorder traversal;
- If your last name starts with I-P, you're presenting in-order traversal;
- If your last name starts with R-Z, you're presenting postorder traversal.

6 Appendix: Student-facing handout for binary tree traversals in-class jigsaw activity

1. Consider the following tree:



   A traversal algorithm visits each node of the tree and prints the character stored there. Write down

   - The result of an in-order traversal of this tree:
   - The result of a postorder traversal of this tree:
   - The result of preorder traversal of this tree:

2. Draw a binary tree T that simultaneously satisfies the following conditions:
   - Each internal node of T stores a single character
   - A preorder traversal of T yields COOKIES
   - An in-order traversal of T yields KOIOECS
3. If you're done earlier than the rest of the class, solve a variation of problem 2: draw a binary tree whose preorder traversal yields COOKIES and in-order traversal yields OOKCEIS.

Real-World Data, Interactive Games and Visualizations in Early CS Courses Using BRIDGES*

Confernce Workshop

Kalpathi Subramanian[1], Erik Saule[1], Jamie Payton[2]
[1]Computer Science
The University of North Carolina at Charlotte
Charlotte, NC 28223
`{krs,esaule}@uncc.edu`
[2]Computer and Information Sciences
Temple University
Philadelphia, PA 19122
`payton@temple.edu`

Summary

Despite the huge explosion in CS enrollments in the past few years, retention of CS majors remains a major concern. Grounding Computer Science concepts in reality by solving important real-world problems or fun problems are key to increasing students' motivation and engagement in computing, which may provide a path to improving retention in CS degree programs. This workshop provides instructors with a hands-on introduction to BRIDGES (http://bridgesuncc.github.io/), a software infrastructure for programming assignments in early computer science courses, including introductory programming (CS1, CS2), data structures, and algorithm analysis. BRIDGES provides capabilities for creating more engaging programming assignments, including: (1) a *simplified API for accessing real-world data sets*, including from social networks; scientific, government, and civic organizations; and movie, music, and literature collections; (2) interesting *visualizations* of the data, (3) an easy to use API that supports *creation of games* that leverage real-world data, and,

*Copyright is held by the author/owner.

(4) *algorithm benchmarking*. Workshop attendees will engage in hands-on experience with BRIDGES with multiple datasets and will have the opportunity to discuss how BRIDGES can be used in their own courses.

Using BRIDGES in data structures, algorithms, and other courses have shown better student outcomes in follow-on courses, when compared to students from other sections of the same course. BRIDGES has impacted over 1500 students across 10 institutions since its inception 5 years ago. A repository of BRIDGES assignments (http://bridgesuncc.github.io/newassignments.html) is now maintained for use by BRIDGES users. Example BRIDGES visualizations are illustrated below.

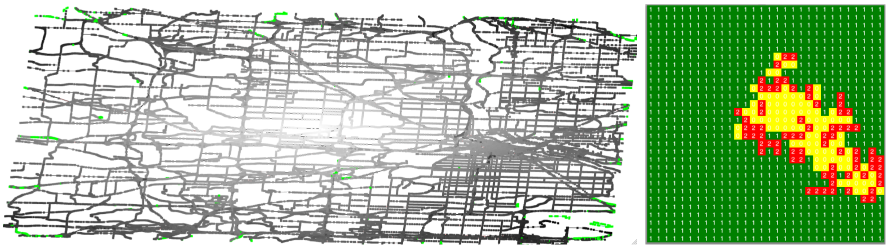


Figure 1: BRIDGES Examples. *[Left:]* Students can explore Dijkstra's shortest path algorithm applied to the streets of Minneapolis (or any city/region of their choice) using Open Street Map data in their assignments in a data structures course (here, gray values are lighter close to the source and darken with distance). *[Right:]* Students can explore arrays and control structures in a simple fire spreading simulation exercise for use in CS1 or CS2.

Workshop Agenda

Participants will be provided accounts on Cloud9, with all needed software installed ahead of the workshop for 3 programming languages. Participants will work in groups.

- **BRIDGES Overview, Design, Demo [10 min]**
- **Setup [10 min]:** Cloud 9 login; execute sample BRIDGES program
- **Hands-on Experience 1 [30 min]:** Scaffolded example of creating a simple BRIDGES program and exploring visualizations of output.
- **Break [10 min]**
- **Hands-on Experience 2 [45 min]:** Participants in each group will choose from a set of 3-4 example problems (a game, or assessing performance of algorithms, or a data structure with a chosen dataset) relevant to a course of interest; attendees will work through the chosen example with provided scaffolder/starter code (extensions will be provided

for early finishers). Examples are available at the BRIDGES site and in our assignment repository. the BRIDGES site (http://bridgesuncc.github.io/) and in our assignment repository (http://bridgesuncc.github.io/newassignments.html)
- **Break [10 min]**
- **Participant Discussion [20 min]:** Each group of participants through a discussion of (1) difficulties faced in creating examples and assignments in early CS courses, (2) opportunities for the use of BRIDGES to address these issues, and (3) challenges in using infrastructure like BRIDGES.
- **Participant Presentation/Discussion [30 min]:** Each group will report on their discussion.
- **Surveys and Closing Remarks [10 min]**

Conference Workshop

How to Create, Host, and Successfully Run a High School Programming Contest

Eric Breimer, Daniel DiTursi, Robin Flatland, Ira Goldstein, Darren Lim, James Matthews, Scott Vandenberg, Pauline White
Siena College
Loudonville, NY 12198
{ebreimer, dditursi, flatland, igoldstein, dlim, matthews, vandenberg, pwhite}@siena.edu

In this workshop, we will describe the numerous processes and tasks involved in successfully hosting a high school programming contest. We will describe the mechanics of running the contest using PC^2, within the logistics of a college campus environment. We will talk about the logistics necessary to support the number of high schools and teams (currently we involve 15+ high schools and over 60 4-person teams). Finally, we will describe our local chapter of CSTA, which strengthens bonds and connections between ourselves and the school's coaches and advisors.

The workshop will start with Jim Matthews introducing the contest itself; the May 1st edition will be the 33rd annual contest held at Siena College. The schools and the number of teams that participate have varied over the years, with many schools who stopped participating until recent developments. He will talk about the average number of teams per contest before 2018. He will then address the drastic format change in 2018, where teams are split into two groups: Gold and Green, based upon the team members' programming experience.

Following the introduction, Jim will continue on details of registration and date/time scheduling. The teams are informed of the contest date nearly a year in advance, with the date carefully chosen so as to not conflict with the schools' calendars. The contest has recently been on a Friday afternoon and evening, with a strict contest date schedule of a welcome/orientation, the programming contest itself, a CSTA meeting, and an awards ceremony.

Darren will then describe the programming contest problem making process. Typically, a problem set consists of 7 problems, generally written in increasing difficulty. Since 2018, the contest has split into two groups. The Green group are teams with members having taken one programming course; the Gold group are teams with more than one programming course and/or an AP course. Two problem sets are made for contests since 2018, although programming problems may be used for both the Gold and Green groups.

After the problem descriptions, Scott will talk about the rooms/programming environments used for the contest. At Siena, we have typically supported Java, Python and C, based upon the requests of the schools who have attended. The responsibility of the room scheduler is to determine which rooms are needed, which (and how many) computers will be used, and which languages/IDEs are found on those computers.

Dan will then discuss the system and networking requirements for using PC^2. Given our network, and the use of predominantly (but not exclusively) wired- connection machines, the Siena contest has been able to support over sixty teams on site. He will describe the preparation work needed to set up the local clients, the judging clients and the scoreboard, which is also covered by Eric.

To support sixty or more teams, a healthy student volunteer population is needed. Robin will talk about the issues of student involvement. The pre-contest phase of student helpers includes the creation of programming contest team folders and T-Shirt organization. The day-of-contest phase includes helpers at registration, problem judging, food distribution, and at all rooms (some of which are actually professor offices) with programming teams.

Since the contest lasts nearly four hours starting around 5:00 PM, we provide food for all participants (students, teachers/coaches, volunteers). Ira will discuss the food issues for feeding over three hundred people. Feeding that many people on a Friday night is not possible through local college catering; a process for ordering and bringing food for all participants will be described.

Finally, the connections that we have made with teachers and advisors over the years helped in the creation of the NY Capital District chapter of CSTA. Since the chapter's formation and membership expansion, the programming contest has attracted participation from a number of new school districts and advisors. Pauline will describe the chapter's creation and its connection to the programming contest. She will also discuss the chapter meeting held during the contest, which includes a review of hints and solutions for the programming contest problems.

From Drawing to Coding: Teaching Programming with Processing

Conference Tutorial

Mihaela Malita[1], *Ethel Schuster*[2]
[1]*Department of Computer Science & Engineering*
Saint Anselm College
Manchester, NH 03102
mmalita@anselm.edu
[2]*Department of Computer & Information Sciences*
Northern Essex Community College
Haverhill, MA 01830
eschuster@necc.mass.edu

We have successfully introduced students to programming using Processing as the language of choice in both our courses: "Introduction to Computer Science" at Northern Essex Community College, and "Introduction to Computer Graphics" at Saint Anselm College.

"Put a pencil in a child's hand and you will have him/her quiet for a long time." The same goes for our students. We have encouraged them to draw and create their own art pieces using colors and shapes while learning to program and write their own code. Programming [4] has served as the tool for them to create and solve problems. It is fun to solve problems when one can visualize the results almost instantly.

Processing, a programming language based on Java, was specifically designed for artists and programmers to visualize works of art [2], [5], [1]. Its syntax is simple, its interface is easy to use, and it enables the students to see their code on a canvas. It is an object-oriented programming language. These factors make it a good fit for a wide variety of students, including non-computer science majors. Our students have been able to create complex and beautiful pieces of art while learning how to manipulate variables and objects, and using control structures that include branching and repetition.

In this tutorial, we will introduce the participants to Processing and its qualities, such as shape, color, and form, which please the aesthetic senses, especially the visual one, and are universal, even among non-majors and people who are not in the field of computer science. We will share how our students have been so excited about programming and problem-solving [6] that they have not been hindered by the required concepts or mathematics. We have found that students struggle with understanding the coordinate system; thus, we will share exercises that facilitate the process. We will show how our students have learned to understand coordinate systems, sizes, and measurements. We will share examples of how natural recursion can be taught by drawing recursive shapes. Art examples will include the works of artists such as Piet Mondrian, Jackson Pollock, Vasarely, Agnes Martin, and Josef Albers.

We will present hands-on problems that entail writing programs to: (1) draw multiple objects of varying sizes and colors — including balls, balloons, pencils, Christmas trees, (2) solve problems that involve conditionals — for example. when the user clicks on a mouse or presses a key, (3) repeat certain actions: draw the same object multiple times, change its location, size, and/or colors as a way to understand loops, (4) animate a scene, create an interactive game, (4) manipulate images: one specific task involves taking students' own selfie and manipulating the image to change colors, distort it, and perform other modifications [3].

References

[1] Processing foundation. http://www.processing.org/.

[2] Ira Greenberg. *Processing: Creative Coding and Generative Art in Processing 2.* Apress, 2013.

[3] Mihaela Malita. Learn processing. https://github.com/mmalita/BeginProcessingArt.

[4] Abdallah Mohamed. Designing a cs1 programming course for a mixed-ability class. In *Proceedings of the Western Canadian Conference on Computing Education*, pages 1–6. WCCCE '19, 2019.

[5] Casey Reas and Ben Fry. *Processing: A Programming Handbook for Visual Designers and Artists (The MIT Press).* The MIT Press, 2014.

[6] Daniel Shiffman. The coding train. https://www.youtube.com/user/shiffman/playlists?view=50&sort=dd&shelf_id=2.

Tutorial on Open Educational Resources and Creative Commons License*

Conference Tutorial

Susan Imberman[1] *and Ann Fidder*[2]
[1]*Computer Science Department*
College of Staten Island
Staten Island, New York USA

[2]*Office of Library Sciences*
City University of New York
New Yo9rk, New York USA
susan.imberman@csi.cuny.edu

Open Educational Resources are quickly becoming the norm for sharing curricula and curricula related materials [3]. The CS education community has always been supportive of the creation and sharing of innovative and new methods for teaching computer science. By publishing work in conference venues and placing materials in repositories, we have openly shared these resources. What we have not paid as much attention to is HOW we share these resources with respect to copyright. By default, if there is no explicit copyright associated with a work, the copyright defaults to "all rights reserved". This means that should someone wish to modify the work, and share the modification back to the CS education community, they would have to get the author's explicit permission. This is fine if one or two people contact the author, but as we share our material in venues such as CCSC, we hope to reach a broad audience. Hence, authors may have many colleagues interested in using and tweaking their published materials. One way to inform the community as to how you wish your work is shared is by adding a Creative Commons copyright license to the work. The various levels of license explicitly define to what extent, and how work is shared back.

Many educators are not familiar with the Creative Commons license, the Open Education Resource movement, and how the two work together to allow

*Copyright is held by the author/owner.

for a broader cycle of improvement and sharing of an academic product. In this tutorial, the authors will share their experience in creating Open Educational Resources (OER) as well as how to submit these to the various OER repositories in order to render them discoverable and available. Many areas in Computer Science are actively evolving. Thus, we often find that material becomes quickly outdated. By sharing our resources under a Creative Commons license, we allow the community as a whole, to update and share, thus keeping content current [1] [2]

Open Educational resources, including open textbooks, was shown to save students considerable amounts of money. In CUNY, over the two years of a New York State funded Open Educational Resource project to adopt and create OER, students have saved over 28 million dollars in textbook costs. We have also found that in OER courses, there was a modest increase in grades and a measurable decrease in course drop, fail, withdrawal rates. [4]

This tutorial will focus on what are open educational resources (OER), definitions of the various Creative Commons licenses, and the tools and methods used to incorporate these into a shared work product. We will instruct on how to find repositories and other sources of OER materials by listing current popular OER sites and their focus. Knowledge of how we can share our resources while broadening their availability, as well as allowing for quick and current updates, enables educators to more easily offer cutting edge, updated course content within a quicker time frame.

References

[1] Creative Commons. CC Licenses and Examples. https://creativecommons.org/share-your-work/licensing-examples/.

[2] Creative Commons. Creative commons. https://creativecommons.org/.

[3] Susan Imberman and Ann Fiddler. Share and Share Alike: Using Creative Commons Licenses to Create OER. *ACM Inroads*, 10(2):16–21, April 2019.

[4] Achieving the Dream. Achieving the Dream New Study Reveals that OER Courses and Degrees Benefit Student Retention and Completion, Improve Faculty Engagement, and Result in Cost Savings for Students;. https://www.achievingthedream.org/press_release/17506/new-study-reveals-that-oer-courses-and-degrees-benefit-student-retention-and-completion-improve-faculty-engagement-and-result-in-cost-savings-for-students.

Using Subgoal Labeling in Teaching Introductory Programming[*]

Conference Tutorial

Adrienne Decker[1], *Briana B. Morrison*[2], *Lauren Margulieux*[3]
[1]*Department of Engineering Education*
University at Buffalo
adrienne@buffalo.edu
[2]*Computer Science Department*
University of Nebraska Omaha
bbmorrison@unomaha.edu
[3]*Department of Learning Sciences*
Georgia State University
lmargulieux@gsu.edu

1 Summary

As global interest in computing increases, so have the enrollments in CS1 courses along with increased interest in computing in the primary and secondary (K-12) environment. Despite considerable research and efforts to improve student outcomes and success in the first programming course, failure rates remain somewhat constant, around 30% (Bennedsen Caspersen, 2007, 2019)[1, 2]. This demonstration will introduce our implementation of using subgoal labels within worked examples and associated practice problems for a Java-based introductory programming courses. Our findings indicate that using subgoal labels have been shown to improve the problem-solving performance of students while helping at-risk students succeed and improving persistence in the course. We will demonstrate 1) recorded videos explaining the worked examples and 2) the eBook implementation of the practice problems that allow students to put into action what they have learned. This demonstration will also provide a brief overview of the research and evidence demonstrating the efficacy of our instructional materials. Additional information about using subgoals in computing can be found at http://cs1subgoals.org/.

[*]Copyright is held by the author/owner.

2 Background

Subgoal labeling is a technique for teaching with worked examples and practice problems has been shown to increase novice user understanding of introductory concepts and problem-solving performance. In our initial research into using subgoal labels in an introductory programming course, we found that students who learned with subgoal labeled instructional materials performed better on weekly quizzes [4] and gave more complete answers (based on the SOLO taxonomy) [3]. Additionally, when looking at students who did not take all the exams in the course or had an average exam score below 70% (passing), approximately half as many students in the subgoal group as the control group fell into this category. Students in the subgoal group were half as likely to withdraw and half as likely to fail than the control group. Subgoal labels also aided in persistence in the course, especially for students most at risk.

3 Tutorial Agenda

1. Introduction and Background (10 minutes)
   (a) Cognitive Load Theory, Worked Examples with Practice Problem Pairs
   (b) Subgoal Learning Framework (which will include labels)
2. Sample Lesson (30 minutes)
   (a) Introduction of Topic and Worked Example Walk-Through (10 minutes)
   (b) Small group exercise/discussion to develop understanding of how to use the worked examples in class (15 minutes)
   (c) Assessment using subgoals (5 minutes)
3. Presentation of current results (10 minutes)
   (a) Quiz, exam, persistence results
   (b) Explain in Plain English results
4. Ebook Implementation (10 minutes)
   (a) Demonstration of worked examples and practice problems online
5. Wrap up Discussion/Questions (15 minutes)

4 Acknowledgments

This work is funded in part by the National Science Foundation under grants 1712025, 1712231 and 1927906. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] Jens Bennedsen and Michael E Caspersen. Failure rates in introductory programming. *AcM SIGcSE Bulletin*, 39(2):32–36, 2007.

[2] Jens Bennedsen and Michael E Caspersen. Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2):30–36, 2019.

[3] Adrienne Decker, Lauren E Margulieux, and Briana B Morrison. Using the SOLO taxonomy to understand subgoal labels effect in CS1. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 209–217, 2019.

[4] Lauren E Margulieux, Briana B Morrison, and Adrienne Decker. Design and pilot testing of subgoal labeled worked examples for five core concepts in CS1. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 548–554, 2019.

Want your students to participate in Open Source? Join us in LibreFoodPantry![*]

Lightning Talk

Karl R. Wurst[1], Stoney Jackson[2], Heidi J. C. Ellis[2], Darci Burdge[3], Lori Postner[3]
[1]Computer Science Department
Worcester State University, Worcester, MA 01602
kwurst@worcester.edu
[2]Computer Science and Information Technology Department,
Western New England University, Springfield, MA 01119
{stoney.jackson, heidi.ellis}@wne.edu
[3]Department of Mathematics,
Computer Science and Information Technology,
Nassau Community College, Garden City, NY 11530
{darci.burdge, lori.postner}@ncc.edu

Student participation in Humanitarian Free and Open Source Software (HFOSS) communities and projects has many benefits — students get experience working on authentic software projects, with large codebases, real users, and communities of experienced developers to interact with. But, the learning curve for working within these project communities can be steep, requiring a significant investment of time and effort to navigate and integrate into the project's community. This may be enough to prevent some faculty and students from attempting to join a community, or to give up after trying.

In order to make it easier and more comfortable for students and faculty to join a project community, we have developed LibreFoodPantry — a welcoming and inclusive, faculty-led, multi-institutional HFOSS community with an on-ramp specifically designed to simplify that learning curve. You can customize the code for your own campus food pantry, or if you don't have one, help other campus food pantries. The LibreFoodPantry community accommodates full classes of students joining for just a semester, as dictated by the

[*]Copyright is held by the author/owner.

instructor's teaching schedule. The community is intended to have a critical mass of developers and mentors, both faculty and current and former students, to provide help and rapid on-boarding. Class members become full members of the development community and are able to help make decisions that affect the direction of the project and receive elevated privileges over project resources. To make it easier for new faculty and classes to join, we have developed documentation and example workflows, and configured development and communication tools. In return, we expect the participating faculty member to participate in the governance of the project for the semester(s) that their class is participating.

Come to this lightning talk to find out more about how joining our community will benefit you and your classes.

Biographies

Karl Wurst (Worcester State University), Stoney Jackson (Western New England University), Heidi Ellis (Western New England University), Darci Burdge (Nassau Community College), and Lori Postner (Nassau Community College), are founding members of the LibreFoodPantry Coordinating Committee. For years they have been working to encourage student and faculty participation in Humanitarian Free and Open Source Software projects and communities. LibreFoodPantry is their latest attempt to make HFOSS participation accessible for both students and faculty.

Supporting Materials

The LibreFoodPantry website (http://librefoodpantry.org) has links to the vision, mission, and code of conduct for the community, as well as documentation on tools, development processes, and workflows. The site also has links to the constituent projects' codebases, issue trackers, and communication channels.

Bringing Industry into the University Experience*

Panel Discussion

[1]Adrienne Decker, [2]Peter DePasquale, [3]Rajendra K. Raj
[4]Matt Jadud
[1]Department of Engineering Education
University at Buffalo, Buffalo, New York 14260
adrienne@buffalo.edu
[2]Tandon School of Engineering
New York University, New York, NY 10003
peter.depasquale@nyu.edu
[3]Department of Computer Science
Rochester Institute of Technology, Rochester, NY 14623
rkr@cs.rit.edu
[4]Applied Research in Acoustics
matthew.c@jadud.com

1 Summary

A large percentage of students in any computing program will graduate and take jobs in industry. Part of the challenge in the college curriculum is creating real-world experiences for students while still in school that will help prepare them for the challenges that lay ahead in their future work in professional practice. This panel will examine different approaches to providing students with industry experiences and perspectives while they are still students to help better prepare them for their professional lives.

2 Adrienne Decker

I am currently involved in a funded research project in conjunction with a company whereby the company is driving the design and development of the project as it aligns with their business goals but the university is providing

*Copyright is held by the author/owner.

support in terms of additional direction and students. The project is entirely distributed as the company and the university are not in the same state. The project team consists of one full-time developer and one manager/developer from the company and three undergraduate students working at the university. I will discuss the nature of the workflow of the projects and how we are facilitating interactions between the students and company. I will discuss the positive and negative aspects of this relationship both from the perspective of the faculty and students and the company.

3 Peter DePasquale

While at The College of New Jersey, I was the Computer Science faculty advisor to the for-credit internship program. For four years, I worked with local and regional companies to facilitate the recruitment of our students for internships. I additionally met with the student's manager during their internship experience to ensure a high- quality experience for both parties. From 2015-2019, I returned to the commercial software development with stops at two start-up companies and one larger organization, focusing on designing and implementing backend software for Enterprise Java applications. I am currently working to develop a course (and possible textbook) in Enterprise software development bringing together many upper-level topics, all of which are utilized in current commercial software development.

4 Rajendra K. Raj

At Rochester Institute of Technology, computing students are required to complete a certain number of hours of cooperative education (co-op) work experiences before they graduate. For example, undergraduate computer science majors need to complete two semesters and one summer of co-op before graduation; note that the different co-op experiences cannot be contiguous. Co-ops typically commence in the third year and must end at least one term prior to graduation. I will present co-op implementations at different universities, and discuss their strengths and weaknesses.

5 Matt Jadud

I have served as faculty at three liberal arts institutions, and have recently joined ARiA full-time, a research and development firm I collaborated with (and sent students to) over the past decade. My role involves algorithm design and development, proposal development and reporting, project management, team development, and recruiting and retention, from interns to new hires. It is humbling to see how quickly one's perspective on curricula and traditional classroom experiences shifts in transitioning away from the academy.

Strategies for Maximizing the Value of Industry Adjuncts: The Tech-in-Residence Corps Model*

Panel Discussion

[1]Susan P. Imberman, [2]Robert Domanski, [3]Shermane Austin, [4]Ross Dakin
[1]The Graduate Center, CUNY, New York, NY 10017
[2]New York City Government
[3] Medgar Evers College, Brooklyn, NY 11225
[4] New Jersey Office of Innovation

1 Summary

Responding to NYC Mayor Bill de Blasio's call to double the number of students graduating with a Bachelors degrees in computer science, in 2017, the Tech-in-Residence Corps (TIRC) program, first-of-its-kind multi stakeholder partnership between academia, government, and private industry, was established at the City University of New York. The TIRC program quickly led to a rethinking of the potential roles that industry adjuncts can take on within CS Departments. Its driving hypothesis was that by employing a coordinated campaign to recruit top-quality industry practitioners to not only teach students advanced applied topics directly, but also provide feedback on curriculum development, this would enable campus departments to align better with industry practices. In addition, industry adjuncts explored opportunities within their companies for student internships and full-time entry-level jobs, with the hope that this would increase students' job competitiveness. Ultimately, this created a sustained feedback loop that lead to other mutually beneficial arrangements between the tech industry and the university's CS Departments. This panel discussion will represent the different perspectives of various stakeholders who have directly participated in the Tech-in-Residence Corps program: department faculty, an industry adjunct, the TIRC program manager, and a university administrator.

*Copyright is held by the author/owner.

2 Susan Imnberman and Robert Domanski

The Tech-In-Residence program is a university wide initiative focused on bringing industry professionals into City University of New York (CUNY) classrooms. Faced with increasing enrollment in tech majors, difficulty in hiring faculty, and the need for more industry focus, a multi campus approach was needed to best leverage available resources. With most graduates entering industry as opposed to academics, there is a shortage of instructors to fill the needs of most campus departments. Individual campus computer science departments do not have the connections, or administrative resources, to recruit classroom adjuncts from industry. The TIRC program coupled the industry relationships of New York City's Tech Talent pipeline with the University's own industry relationships through its workforce development side, to recruit from the NYC tech ecosystem, individuals who can bring into the classroom the tech skills identified by industry as being high in demand. The central administrative approach was crucial in the success of this program. The program has grown from 7 courses across 5 campuses in Spring 2018 to 22 courses across 9 campuses in Spring 2020. Since inception in 2018, the program has run an accumulated total of 73 courses in in-demand topics such as A.I., Blockchain, Cloud Computing, Cybersecurity, Data Science, and more.Due to program success, we have plans to extend this program to other industry-focused disciplines such as business and engineering.

3 Shermane Austin

The Computer Science Program at CUNY Medgar Evers College has been participating in the Tech-in-Residence Corps (TIRC) for three (3) semesters. Our objective has been to enhance our CS curriculum with the rapidly changing needs of the tech industry in order to improve student readiness for the tech workforce. This is the overall purpose of TIRC. The TIRC program has provided us with the opportunity to enrich or develop new application-based courses addressing current industry needs. In our case, we have been able to utilize program industry professionals (TIRCs) provided by the program to bring real world practice, applications, and challenges into the CS curriculum. Faculty work directly with these professionals to develop syllabi fusing outcomes with new or updated content. The two courses implemented at Medgar were "Security Software Engineering" and "DevOps". These courses have been successful for our students and will be continued as electives in our curriculum. From our standpoint, even with this limited experience, TIRC is valuable for a number of reasons. The program engages a broad range of industry professionals, vets them for suitability in the classroom, provides introductory-level training for those teaching for the first time, and works closely with faculty and professionals to insure a positive experience for students. As faculty, we would have difficulty attempting to replicate the recruitment of these industry professionals and implement the thoughtful training process used to engage them in the classroom.

4 Ross Dakin

Ross Dakin will present his experience teaching a Web Development course as an adjunct at Lehman College while simultaneously maintaining a full-time role at Bank of America. His perspective offers valuable insights into both what skills and factors shape students' relative levels of job-preparedness, as well as how CS curriculum can be more aligned to industry hiring processes. As an inaugural Tech-in-Residence Corps Member from the program's inception, Ross also provides a lens through which the program's design and implementation can be analyzed from a first-time industry adjunct's "street-level" perspective.

5 Biographies

Susan P. Imberman is The University Associate Dean for Technology Education at the City University of New York Central Office. Her charge is to foster the development of new and innovative technology programs in CUNY. Besides her current administrative role, Dr. Imberman holds academic positions at both The College of Staten Island, CUNY, where she chaired the computer science department, and The Graduate Center, CUNY.
Robert Domanski Ph.D., is the Director of Higher Education for the New York City government's Tech Talent Pipeline industry partnership. As the former Senior Program Manager of the Tech-in-Residence Corps, he oversaw nearly two dozen advanced Computer Science courses at CUNY each semester taught by industry professionals from NYC's tech ecosystem.
Shermane Austin is a Professor of Computer Science at CUNY Medgar Evers College. She is the Program Coordinator for Computer Science and facilitator for the Tech-in-Residence program in the department. She is also the Site Lead for the Google CSSI-Extension program at the college and is an Affiliate Director of the NASA NY State Space Consortium.
Ross Dakin is a former industry adjunct who taught at Lehman College (CUNY) as an inaugural member of the TIRC program. He is currently employed at the New Jersey Office of Innovation. Previously he was the Senior Strategist of Global Technology Operations at Bank of America Merrill Lynch. Ross was also a Presidential Innovation Fellow in the Obama White House.

Integrating Cloud Computing across Existing Computer and Information Science Courses*

Poster Abstract

Ruth Kurniawati
Department of Computer and Information Science
Westfield State University, Westfield, MA 01086
`rkurniawati@westfield.ma.edu`

The Gartner group [3] notes that most in the IT industry will be using cloud-based resources in 2020 and that future use of cloud computing is only expected to grow. Furthermore, 75% of surveyed organizations currently using cloud-services plan to adopt a "cloud-first" strategy, in which cloud computing solutions are evaluated first before making new investments.

The ACM 2013 Computer Science curriculum includes a basic set of cloud computing topics in the Parallel and Distributed Computing area but leaves these topics as elective. There have, however, been suggestions to make this topic a required topic in the CS curriculum [5]. Additionally, we have seen proposals for modular frameworks for cloud computing courses ([2], [1]), an approach which would make it easier to integrate cloud curricula into the greater CS curriculum.

At Westfield State University (WSU), our approach, beginning in fall 2018, has been to introduce students to cloud computing topics by gradually integrating cloud computing resources into our "traditional" computer and information science classes. We chose to use Amazon Web Services (AWS), because this platform is the market leader in cloud computing [4]. In 2019, WSU joined AWS Educate as an institution, which provided us a better platform to set up and monitor assignments. In addition, we have also begun to utilize Google Cloud Platform and QwikLabs in our courses to give students a more vendor-agnostic view of the cloud.

In this poster, I describe the introduction and integration of cloud computing into our Introduction to Networking and Database courses. In these courses,

*Copyright is held by the author/owner.

we first introduce students to an IaaS service in AWS which feels more natural to the students because they have already worked with an equivalent service running on a local machine. This prompts a discussion of the advantages and disadvantages of the cloud services and becomes a foundation for introducing more advanced cloud services. In this poster, I provide an overview of the challenges faced in adding and integrating cloud computing topics into these classes and outline what approaches have helped in overcoming the difficulties experienced along the way. I also outline of our department's future plans, developed in response to our experiences, student feedback, and the modular frameworks as proposed by Foster *et al.* [2] and Debzani *et al.* [1].

References

[1] Debzani Deb, Muztaba Fuad, and Keith Irwin. A module-based approach to teaching big data and cloud computing topics at cs undergraduate level. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 2–8, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Derek Foster, Laurie White, Joshua Adams, D. Cenk Erdil, Harvey Hyman, Stan Kurkovsky, Majd Sakr, and Lee Stott. Cloud computing: Developing contemporary computer science curriculum for a cloud-first future. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, page 130–147, New York, NY, USA, 2018. Association for Computing Machinery.

[3] The Gartner Group. Advanced cloud strategy. https://www.gartner.com/en/information-technology/insights/cloud-strategy. Last accessed on 2020-01-11.

[4] The Gartner Group. Gartner says worldwide IaaS public cloud services market grew 31.3% in 2018. https://www.gartner.com/en/newsroom/press-releases/2019-07-29-gartner-says-worldwide-iaas-public-cloud-services-market-grew-31point3-percent-in-2018. Last accessed on 2020-01-12.

[5] Joel K. Hollingsworth and David J. Powell. Requiring web-based cloud and mobile computing in a computer science undergraduate curriculum. In *ACM Southeast Regional Conference*, pages 19–24. ACM, 2011.

[6] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* Association for Computing Machinery, New York, NY, USA, 2013.

The Low-Budget Experimental Computer Lab Boosts Students' Research*

Poster Abstract

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

This poster discusses efforts at Rivier University to create and run the experimental computer lab that provides undergraduate and graduate students with opportunities to explore various operating systems, computer architectures, databases, multimedia applications, and data communication means as well as build-up and analyze different system prototypes. This approach helps address the considerable student interest in developing medium-size computing and networking systems from scratch and use the acquired practical skills in their capstone research projects.

The Experimental Computer Science Laboratory was open in a 200-square-feet room right near the IT Office in fall 2011. Since then, it has supported the instructional and research needs of the Computer Science/Information Technology students and faculty. The Lab allows instructors to design new advanced course lab assignments and offer research projects requiring a dedicated network of systems and which would be undesirable and unsafe to conduct on the Rivier University network, including projects involving distributed computing, cybersecurity, and data mining. The Lab serves as a general computing resource for the students and faculty.

There are ten computers connected in a local area network available for students' use in the Computer Science Lab: seven are dual boot Fedora^{TM} Linux and Windows^{TM} machines; and three are single boot Fedora^{TM} Linux systems. The computers have basic design and development software, including compilers, database management systems, web servers and associated development tools. The systems are connected in a local area network through a Nortel^{TM}/Avaya^{TM} BayStack 5510-48T switch. Two servers with powerful

*Copyright is held by the author/owner.

GTX-570 graphic drivers and OpenGL library are regularly used for projects and students' research in Computer Graphics and Multimedia Web Development courses.

One of the servers in Lab equipment is running Fedora^{TM} Linux and is acting as the DNS server for the network. The machines in the network have the Internet access, but filtered through an IT department router to prevent unauthorized access to the equipment. Within the local area network, machines have complete network access to each other and can be configured to bring up local web servers as needed by class activities.

One of the servers in Lab equipment is running Fedora^{TM} Linux and is acting as the DNS server for the network. The machines in the network have the Internet access, but filtered through an IT department router to prevent unauthorized access to the equipment. Within the local area network, machines have complete network access to each other and can be configured to bring up local web servers as needed by class activities.

For the Windows^{TM} systems, the Windows 10^{TM} is installed on the remaining server machines using media supplied by the IT department. A second Linux server running Samba^{TM} with active directory support provides login service and roaming for the windows systems. The final solution utilizes the VirtualBox tool installed on both Windows^{TM} and Fedora^{TM} Linux systems. (Potentially, with this option, it is possible to run any other operating system in the lab). All clients are dual boot and the default operating system is Fedora^{TM} Linux.

Most of the lab equipment (computers, switches, racks, wiring, etc.) was "donated" by the IT department of Rivier University. The lean Lab budget ($1K-2$K annually) was used forpurchasing small items (e.g., graphic drivers and inexpensive CISCO^{TM} small firewall-type routers) requested by instructors for lab assignments in particular courses (e.g., Networking Technologies and Computer Security). The long list of installed software includes "traditional" and specialized packages: GCC, g++, IntelliJ Idea, JDK, NetBeans, PostgreSQL, SQL Developer, Foxit Reader, Android SDK Tools, Eclipse–Android Development Tools, Eclipse–Java Development Tools, Plugin, ActivePerl, Adobe Flash Player, Python, VLC, VMware Player, WinSCP, TELNET, PuTTY, Wireshark, and many others.

For the first time, the Experimental Computer Science Lab was effectively used in the authors' courses in Fall-2011 and Spring-2012 semesters. Dr. Pitts' students successfully used the Lab computer systems in the Distributed Computing classes. They experimented with the development of web services using SOAP and WSDL technologies, culminating in the development of a small application using a web service supported by the National Weather Service. Students also performed a lab using Java's remote method invocation mechanism. This example highlights a major benefit of having this isolated lab facility: the software they used in the Distributed System course activities was identified and discussed during one class meeting; by the next class meeting, Dr. Pitts had downloaded the software, installed it and had it ready for the students to use. He has also used the Lab systems in his High Octane Java course for hands- on activities in Java network programming, as well as courses on parallel programming and database programming.

Later, with the help of Hamid Habibi, a lab assistant, Dr. Riabov and other CS instructors developed a set of hands-on Computer Networks lab assignments to replace the OPNET IT GURU virtual labs that became obsolete after the acquisition of OPNET by Riverbed Technology, Inc. For the High Performance Computing course, Dr. Yili Tseng built five computer clusters connected with Nortel^{TM}/Avaya^{TM} 10/100/1000 routing switches (generously donated by the IT department) that were used by students for improving the system performance in their projects.

Nowadays, the Experimental Lab is considered as a main experimental facility to accommodate the needs of students who pursue careers in computer security, cybersecurity management, multimedia, Internet of Things (IoT), cloud computing, and data analytics. Many students use this facility for their extra-curriculum activities (IEEE/ACM group meetings and CS/IT Club) and work on various challenging projects, including the capstone ones, e.g., "Secure Online Biometric Authentication Solution", "Visualization of Multivariate Data through Chernoff Faces", "Searchable Cryptosystem Secure Cloud Storage", "Clue Computer Game", "Tactus: A World Learning Game for Children with Autism", "Design and Implementation of an IoT Smart Farming System", "Smart Grasses Technology Implementation" (based on the Xamarin^{TM} Cross Platform framework), "Data Mining with 3D-RBT Perturbation Technique", "DEM846: Digital Elevation Modeling" (developed by Kevin Gill, a Rivier's alumnus and recently adapted to several space missions at the NASA Jet Propulsion Laboratory) and others.

In numerous course evaluations and exit interviews, students stated that they became deeply engaged in lab and research project activities in the Experimental Computer Science Lab through examining the complex case studies and challenging problems related to the real-world applications of modern computing technologies.

The Lab design, system settings, and maintenance could not be successfully implemented without the outstanding support of Sr. Therese Larochelle, p.m., a former Vice-President for Academic Affairs; the Office of Information Technologies (Bill Schleifer, Marie McMullen, David Bedard, and Sr. Martha Villeneuve), and the Division of Sciences (Dr. Paul Cunningham).

Integrating Personalized Online Practice into an Introductory Programming Course*

Poster Abstract

Yana Kortsarts[1], *Kamil Akhuseyinoglu*[2]
Jordan Barria-Pineda[2], *Peter Brusilovsky*[2]
[1]*Computer Science Department*
Widener University
Chester, PA 19013
`ykortsarts@mail.widener.edu`
[2]*School of Computing and Information*
University of Pittsburgh
`{kaa108, jab464, peterb}@pitt.edu`

We present our experience in integrating interactive learning tools into an introductory programming course curriculum. Our department offers an undergraduate program leading to a Bachelor of Science degree in both Computer Information Systems (CIS) and Computer Science (CS). Students pursuing both majors take Introduction to Computer Science 1 (CS 1) and Introduction to Computer Science 2 (CS 2) courses in their first year. These courses are essential prerequisites to all subsequent computer science courses. The CS1 course combines a thorough introduction to Python programming language. Supervised laboratory sessions include a sequence of exercises covering the following topics: variables and operations, input/output, decision structures, loops, functions, one-dimensional lists, and value and reference parameters concept. Each course is a 4-credit course and the students spend three hours in class and three hours in the laboratory per week. Despite the rich body of the literature in the area of computer science education related to introductory programming, teaching novices how to program remains a challenging task. Educators are constantly looking for innovative ideas and new efficient ways to instill fundamental programming principles and to build problem-solving skills.

*Copyright is held by the author/owner.

In Fall 2019, to enhance students' learning and engagement, an integrated online practice system, Mastery Grids [6], was introduced to the CS1 course curriculum. Mastery Grids provides access to multiple types of interactive learning content through a personalized interface with social comparison and open learning modeling features [2]. The interface of the Mastery Grids consists of a series of topics (a grid structure) and the learning content is accessed through these topics. The topics and the available content were aligned with the course structure by the instructor. The personalized interface visualized their progress and classmates' progress to help students to reflect on their current level of progress and provide navigational support.

The learning contents available in this study are developed by the University of Pittsburgh Personalized Adaptive Web Systems (PAWS) Lab and Aalto University LeTech group [1]. The content developed by the PAWS lab includes parameterized problems that focus on code behavior delivered by the QuizPET system [4], and worked examples and faded examples (challenges), which focus on program construction, delivered by the PCEX system [3]. The content authored by the LeTech group consists of animated code examples [7] and Parson's coding problems [5]. Animated examples demonstrate how programming constructs are executed through animation steps. In Parson's coding problems, students construct a program by sorting given code fragments. Throughout the semester, the instructor assigned some of the available interactive content as a part of the laboratory assignments and left the rest of the content as an elective practice.

The poster discusses our experience in the integration of interactive learning tools into the CS1 course curriculum and reports the preliminary results including student's self-reflections and the responses to the end-of-semester survey. We present the goals and the objectives of the course, a detailed course curriculum focusing on the utilization of Web-based technology, and weekly assignments. The poster also includes the description of the lectures and laboratory sessions demonstrating various ways to integrate the interactive tools into the course material and future plans. In addition, the technical aspects and logistics of the project will be presented and discussed.

References

[1] Peter Brusilovsky, Lauri Malmi, Roya Hosseini, Julio Guerra, Teemu Sirkiä, and Kerttu Pollari-Malmi. An integrated practice system for learning programming in Python: design and evaluation. *Research and Practice in Technology Enhanced Learning*, 13(1):18, 2018.

[2] Peter Brusilovsky, Sibel Somyürek, Julio Guerra, Roya Hosseini, Vladimir Zadorozhny, and Paula J Durlach. Open social student modeling for personalized learning. *IEEE Transactions on Emerging Topics in Computing*, 4(3):450–461, 2015.

[3] Roya Hosseini, Kamil Akhuseyinoglu, Andrew Petersen, Christian D Schunn, and Peter Brusilovsky. PCEX: interactive program construction examples for learning programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, pages 1–9, 2018.

[4] I-H Hsiao, Sergey Sosnovsky, and Peter Brusilovsky. Guiding students to the right questions: adaptive navigation support in an e-learning system for Java programming. *Journal of Computer Assisted Learning*, 26(4):270–283, 2010.

[5] Petri Ihantola and Ville Karavirta. Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education*, 10(2):119–132, 2011.

[6] Tomasz D Loboda, Julio Guerra, Roya Hosseini, and Peter Brusilovsky. Mastery grids: An open source social educational progress visualization. In *European conference on technology enhanced learning*, pages 235–248. Springer, 2014.

[7] Teemu Sirkiä. Jsvee & Kelmu: Creating and tailoring program animations for computing education. *Journal of Software: Evolution and Process*, 30(2):e1924, 2018.

Becoming Lifelong Learners: CS Learners' Autonomy[*]

Poster Abstract

Ruiqi Shen, Joseph Chiou, Michael J. Lee
Department of Informatics
New Jersey Institute of Technology
Newark, NJ 07102
{rs858,jcc45,mjlee}@njit.edu

Those who choose careers related to computer programming typically need to be lifelong learners. Because of rapidly changing technology, whatever they learned at school or on their own may eventually become obsolete. Therefore, people with computer programming careers must continue to learn to keep up with updates, improvements, and trends. Literature suggests that learner autonomy—where learners take charge of their own learning—plays a vital role in developing lifelong learning. Our work is among the first to examine learner autonomy in the context of computing education. A preliminary interview study revealed that computer science (CS) learners with different autonomy levels have different needs from educators and online educational resources. To further explore this finding and learn more about CS learners' autonomy, we conducted a follow-up study. We surveyed 364 CS learners and found that 1) CS learners have overall medium to high autonomy levels; 2) learning experience contributes to these different levels of autonomy; and 3) CS learners prefer using autonomy-supportive systems when learning about CS topics. Based on these observations, we present and discuss potential implications for computing education and provide suggestions to CS educators and system designers.

[*]Copyright is held by the author/owner.

Jupyter Notebooks in Education*

Poster Abstract

Jeremiah W. Johnson, Karen H. Jin
Department of Applied Engineering & Technology
University of New Hampshire
Manchester, NH 03101
{Jeremiah.Johnson, Karen.Jin}@unh.edu

Jupyter notebooks are widely used in industry for a range of tasks. This is particularly so in areas that involve significant amounts of data analysis or machine learning; indeed, while 5% of Python developers surveyed in the 2018 JetBrains Python Developer Survey report using Jupyter notebooks for their primary development tool, when restricted to those working in data science roles, Jupyter notebooks tied with the PyCharm IDE as the most popular tool for Python development [1], and in the 2019 StackOverflow developer survey, 9.5% of developers surveyed listed Jupyter notebooks as their preferred development environment [2].

Jupyter notebooks provide a format that allows the user to combine code, explanation, and analysis in a single document. The ability to mix educational or explanatory content, including, but not limited to, images, video, typeset mathematical equations, and live code makes notebooks a highly effective communication tool that enables a 'flowing narrative' for students to follow. This has a significant pedagogical advantage, and it is difficult to produce a similar experience in other formats. However, literature on if or how Jupyter notebooks are currently being used in education is limited, and what literature does exist is often tailored to their use in teaching specific narrow topics [3, 4]. There is little guidance in the literature on best practices for incorporating Jupyter notebooks into the curriculum.

In this poster, we present the results of a survey of educators on their use of Jupyter notebooks for education. Our goal is to provide some perspective on how Jupyter notebooks are currently being used in education and to illustrate common sentiments regarding their strengths and weaknesses in the classroom, so that others considering the use of Jupyter notebooks in their courses can use them effectively.

*Copyright is held by the author/owner.

References

[1] JetBrains 2018 developer survey. https://www.jetbrains.com/research/python-developers-survey-2018/. Accessed 25 Nov. 2019.

[2] StackOverflow 2019 developer survey. https://insights.stackoverflow.com/survey/2019. Accessed 25 Nov. 2019.

[3] Roland DePratti. Using Jupyter notebooks in a big data programming course. *J. Comput. Sci. Coll.*, 34(6):157–159, April 2019.

[4] Ben Glick and Jens Mache. Using Jupyter notebooks to learn high-performance computing. *J. Comput. Sci. Coll.*, 34(1):180–188, October 2018.

End-to-End Machine Learning Project Design for Undergraduate Classrooms[*]

Poster Abstract

Karen H. Jin
Department of Applied Engineering and Sciences
University of New Hampshire
Manchester, NH 03101
{Karen.Jin}@unh.edu

A Machine Learning (ML) workflow refers to the complete process for carrying out an ML solution, typically including problem formulation, data acquisition, feature engineering, model training, evaluation, and deployment. As the demand for building high-quality ML applications continues to grow in the industry, the current instruction in undergraduate classrooms still focuses mostly on modeling techniques and algorithms. In this work, we explore project designs that provide upper-level undergraduate students with experience in the full ML application development process, while still being manageable in a classroom setting.

The workflow for building machine learning applications has several distinct characteristics compared to traditional software applications, such as data acquisition and transformation, model evaluation and deployment. ML solutions involve more complex infrastructure and development process than traditional software systems [4]. Challenges and risk factors specific to integrating ML capability into software and services have been described in several case studies of real-world ML solutions [1, 2, 3]. Current applied ML courses are often modeling-centric rather than system-centric. The focus is typically placed on ML algorithms and techniques, not on the development of end-to-end ML systems. Although ML code only contributes to a small fraction of real-world ML systems [4], classroom teaching is often still centered around the modeling stage, and rarely goes beyond the coverage of programming languages (e.g., Python, R), libraries (e.g., Scikit-learn, Keras) and other tools such as Jupyter Notebook. Most students don't have the necessary exposure to the complexity of real-world ML application development even after taking upper-level ML

[*]Copyright is held by the author/owner.

courses. Whereas a real-world ML pipeline starts with obtaining data, running some transformation upon the data, loading the data into an ML model, initiating the training of that model on a large cluster, and putting the model into production, existing classroom projects rarely integrate all these stages. Students are typically given clean datasets for their assignments/projects, and such datasets often require minimal effort for further manipulation. Project submissions are often required in the format of notebook or as local programs, rather than an integrated ML system with multiple software components. Students rarely have experience working on a full ML solution, even on a smaller scale than real-world applications.

We aim to help students understand the complexity of real-world ML application development. Subjects in data engineering, software engineering, as well as in development and operations (DevOp) are integrated. The class projects span the full ML workflow, on a scale manageable for both the students and the instructors. Important components of the projects include data collection/feature engineering, model development, and iterations during model training and evaluation. Our experience indicates that designing and teaching such an applied course require that instructors are equipped not only with ML background but also practical knowledge of software engineering and system design. Moreover, given the plethora of constantly changing tools/libraries/platforms, designing projects that are up-to-date with current industrial practice while still being suitable for classroom instruction is a challenge.

References

[1] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice*, ICSE-SEIP '10, pages 291–300, Piscataway, NJ, USA, 2019. IEEE Press.

[2] Besmira Nushi, Ece Kamar, Eric Horvitz, and Donald Kossmann. On human intellect and machine failures: Troubleshooting integrative machine learning systems. *CoRR*, abs/1611.08309, 2016.

[3] Md Saidur Rahman, Emilio Rivera, Foutse Khomh, Yann-Gaël Guéhéneuc, and Bernd Lehnert. Machine learning software engineering in practice: An industrial case study. *CoRR*, abs/1906.07154, 2019.

[4] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*, pages 2503–2511, 2015.

Applying Three Machine Learning Algorithms to Three Breast Cancer Diagnosis Datasets[*]

Poster Abstract

Lilly Shelomyanov and Sofya Poger
Department of Computer Science
Felician University
Lodi, NJ 07644
lostrovsk@gmail.com pogers@felician.edu

Breast cancer is the second most common cause of mortality in women in the United States [10]. An estimated 268,600 women were diagnosed with breast cancer in 2019, and 41,760 died from the disease [10]. The early detection of breast cancer is imperative for improving chances of survival [5]. Mammography remains the gold standard for screening individuals for breast cancer [3]. However, this screening tool carries risk for false-positive results, especially in women with high-risk features, resulting in unnecessary and invasive procedures to confirm diagnosis [11, 4]. As such, there's an unmet need to develop alternative breast cancer diagnostic tools. Machine learning (ML) is one such tool that is gaining traction in the breast cancer arena, with the goal of accurately predicting breast cancer diagnosis [12, 6, 1, 2]. Current research efforts are focused on finding the most accurate, robust ML algorithms for predicting breast cancer [6]. Many studies that evaluate the performance of ML algorithms either do so using only one algorithm and/or one dataset, thereby limiting the applicability of the algorithm to datasets of varying size and attributes (features) [6]. This study was conducted to provide a more complete overview of three algorithms—generalized linear model (GLM), support vector machine (SVM), and artificial neural network (ANN)—with respect to how they compare to one another and how they perform on three datasets of varying size and features, including the Wisconsin Breast Cancer Original (WBCO) dataset, the Wisconsin Breast Cancer Diagnostic dataset (WBCD), and the Wisconsin Breast Cancer Coimbra (WBCC) dataset.

[*]Copyright is held by the author/owner.

The WBCO dataset includes 11 attributes collected from 699 patients, and the WBCD dataset contains 30 attributes from 569 patients; in both of these datasets, attributes were extracted using Xcyt from digitized images of samples obtained from fine needle aspiration biopsies [9, 8]. The WBCC dataset includes 10 attributes that were analyzed during routine blood work in 116 patients [7]. Performance of the three algorithms was assessed using accuracy, sensitivity, and specificity. Each of the datasets was split into training (66%) and testing (33%) sets, and we used R and RStudio to perform the data analysis. Accuracy results exceeded 96% when the GLM and SVM algorithms were applied to the WBCO and WBCD datasets, but decreased to <74% when these two algorithms were used on the WBCC dataset. Of the three algorithms, ANN was the least accurate across all three datasets, demonstrating the lowest accuracy in the WBCC dataset (48.25%). Sensitivity was highest in the WBCO and WBCD datasets when the GLM and SVM algorithms were used, ranging from 93.06% to 97.44%; however, when applied to the WBCC dataset, sensitivity results of GLM and SVM were significantly reduced (range, 52%-64%). Sensitivity results were lowest across all datasets with the ANN algorithm (range, 0% to 40%). Specificity was highest across all algorithms in the WBCO and WBCD datasets (range, 96.64%-100%), and was lowest in the WBCC dataset (range, 64.1%-92.31%), with ANN being the least specific (64.1%).

References

[1] Adel Aloraini. Different machine learning algorithms for breast cancer diagnosis. *International Journal of Artificial Intelligence & Applications*, 3(6):21, 2012.

[2] Manisha Bahl, Regina Barzilay, Adam B Yedidia, Nicholas J Locascio, Lili Yu, and Constance D Lehman. High-risk breast lesions: a machine learning model to predict pathologic upgrade and reduce unnecessary surgical excision. *Radiology*, 286(3):810–818, 2018.

[3] Joann G Elmore, Mary B Barton, Victoria M Moceri, Sarah Polk, Philip J Arena, and Suzanne W Fletcher. Ten-year risk of false positive screening mammograms and clinical breast examinations. *New England Journal of Medicine*, 338(16):1089–1096, 1998.

[4] Susan G. Komen. Accuracy of mammograms. https://ww5.komen.org/BreastCancer/AccuracyofMammograms.html Accessed January 21, 2020.

[5] Epidemiology National Cancer Institute: Surveillance and End Results Program. Cancer stat facts: female breast cancer. https://seer.cancer.gov/statfacts/html/breast.html Accessed January 21, 2020.

[6] Ricvan Dana Nindrea, Teguh Aryandono, Lutfan Lazuardi, and Iwan Dwiprahasto. Diagnostic accuracy of different machine learning algorithms for breast cancer risk calculation: a meta-analysis. *Asian Pacific journal of cancer prevention: APJCP*, 19(7):1747, 2018.

[7] UCI Machine Learning Repository. Breast cancer coimbra data set. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Coimbra Accessed November 19, 2019.

[8] UCI Machine Learning Repository. Breast cancer wisconsin (diagnostic) data set. https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) Accessed November 19, 2019.

[9] UCI Machine Learning Repository. Breast cancer wisconsin (original) data set. https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original) Accessed November 19, 2019.

[10] American Cancer Society. How common is breast cancer? https://www.cancer.org/cancer/breast-cancer/about/how-common-is-breast-cancer.html Accessed January 21, 2020.

[11] American Cancer Society. Limitations of mammograms. https://www.cancer.org/cancer/breast-cancer/screening-tests-and-early-detection/mammograms/limitations-of-mammograms.html Accessed January 21, 2020.

[12] Wenbin Yue, Zidong Wang, Hongwei Chen, Annette Payne, and Xiaohui Liu. Machine learning with applications in breast cancer diagnosis and prognosis. *Designs*, 2(2):13, 2018.

Differentiating Computer Science Courses in Undergraduate and Graduate Level[*]

Poster Abstract

Songmei Yu and Sofya Poger
Department of Computer Science
Felician University
Lodi, NJ 07644
{yus, pogers}@felician.edu

We have BS and MS programs in Computer Science. As some graduate students come to MS programs without prior computer science background, they need to take certain pre-requisites before they take the master level courses [1]. Meanwhile, we offer the same courses to both programs, most likely these courses are elective courses which are developed based on the current market need, such as Data Mining and Big Data, Computer Vision, Machine Learning, Artificial Intelligence, undergraduate/graduate Capstone Project to prepare students for their future career interests.

Although the basic contents are similar for each course, two complexity levels are offered with different course number and title (i.e., for undergraduate program we name it as Intro to xxx, whereas for graduate programs we name it as Advanced xxx), and at different times of day (the undergraduate level is normally in the daytime, the graduate level is at night). Moreover, we need to adjust the contents and assignments to differentiate the undergraduate and graduate levels, so that students could benefit the most from each course. As an example, the curriculum for Data Mining and Big Data course is illustrated below.

We have both CS430 (Intro to Data Mining and Big Data) for the undergraduate level, and CS675 (Advanced Data Mining) for the graduate level. For CS430, we use Introduction to Data Mining (ISBN: 0133128903). The course contents include Intro to Data Mining, Data basics, Classification: Basics and Alternatives, Association Analysis: Basics and Advanced, Cluster Analysis: Basics and Additional, Anomaly Detection, as well as Big Data and Related Technics. For CS675, we use Data Mining: Concepts and Techniques

[*]Copyright is held by the author/owner.

(ISBN: 9780123814791), as well as research papers from ACM and IEEE Resources. The course contents include Introduction to Data, Data Warehousing and OLAP, Pattern Mining: Basics and Advanced, Classification: Basics and Advanced, Cluster Analysis: Basics and Advanced, Outlier Detection, and Data Mining Trends and Research Frontiers.

For the assignments, CS430 will be assigned a project to implement a selected Algorithm with students having options to choose from one of the three topics: Data mining for weather prediction and climate change, Web Mining Techniques, and Mining of government data for getting valuable information). CS675 will be assigned a research paper where students need to write a 10-12-page research paper to survey current research work on one of the selected data mining areas, present the challenging issues of the current research, and investigate possible ideas or solutions to tackle one of the challenging issues. For other assessments, both courses will be given weekly quiz, midterm and final exam, as well as project/research presentation. The same is done for CS 470 Introduction to Artificial Intelligence and CS 665 Advanced Artificial Intelligence. CS 460 Undergraduate Capstone Project and CS 699 Graduate Capstone Project. Since many undergraduate students are interested in research in Computer Science and pursuing graduate degree, we give the undergraduate students a choice for their term project: programming project or survey research paper [2].

Although both levels of each course are overlapped in certain contents, the two major differences are as follows:

1. Pace and Depth of the Contents

The teaching pace and depth varies between two levels, as we teach more materials and present current research work for each topic on the graduate level, and for the undergraduate level, we start from the basics, and gradually move to the existing algorithms to solve the problem.

2. Assessment

For the undergraduate level, we normally give a project and students are required to implement it based on the given algorithm(s). For undergraduate students, we give an alternative: a programming project or a research paper. Also, weekly quiz and midterm/final exam are required to enhance the learning outcome of the basics. For the graduate level, a research paper is required and plays an important role in the final assessment. Students need to read assigned papers and perform a survey, analysis, and investigation of the possible solutions to the current challenging issue(s). A term paper/project and its presentation, quizzes after each chapter, online and in-class discussions, midterm and final exams serve as the basis for the final grade. This research work serves as the direction for undergraduate students to pursue a graduate degree and for a graduate student to study further.

Our future work will be focusing on other aspects of learning outcome comparisons; and integrating students' job placement into consideration [3].

References

[1] Sorathan Chaturapruek, Thomas S Dee, Ramesh Johari, René F Kizilcec, and Mitchell L Stevens. How a data-driven course planning tool affects college students' gpa: evidence from two field experiments. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, pages 1–10, 2018.

[2] Paul R Pintrich and Dale H Schunk. *Motivation in education: Theory, research, and applications*. Prentice Hall, 2008.

[3] Michael Prince. Does active learning work? a review of the research. *Journal of engineering education*, 93(3):223–231, 2004.

Capstone: Transitioning a Successful Undergraduate Research Program to a Multi-Research Model[*]

Poster Abstract

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

The Capstone course for computing students at the University of New Hampshire at Manchester (UNH-M) was introduced in spring 2011. It was designed as an undergraduate research project whose focus was speech recognition [1, 2]. In its 10th year, the course is now being transitioned into multiple projects with an initial split of two research topics. Much deliberation was needed to help determine the best approach on how to split the course into two components; a seemingly simple task fraught with pitfalls. Two important issues were whether to run parallel projects or alternate between spring an fall semesters, and how to determine the best method on dividing students among the competing projects. What initially seemed a trivial task turned into a complex problem that needed to address potential consequences so as not to undermine the successful mechanism that had been put in place and had worked so well over that decade.

With Capstone at UNH-M being an internal research project where anonymous peer evaluations are the primary driver of student success in the class, being able to ensure workable class size was another consideration. Offering projects in both spring and fall could present occasion where one semester might only yield a small handful of students participating and thus remove a critical element that a large group research project provides where students need to communicate, collaborate, and find ways to contribute to the overall success of the project. Could faculty determine student schedule before their final year and dictate who would take the fall versus spring Capstone to ensure a balanced set of projects, or was that too problematic resulting in a weakened experience? Conversely, would running dual research projects in parallel

[*]Copyright is held by the author/owner.

present a shortage of resources in both faculty and facilities? These were important issues needed to be addressed.

Although offering the course every semester would make it more convenient for students, offering it only once a year adds an element of commitment and responsibility. The decision of simply dropping the course because the student felt unprepared or overwhelmed would mean having to wait an entire year to re-take it. Stated more simply, a student may find it easier, when confronted with a difficult challenge, to drop a course knowing they can simple take it again the following semester, whereas if the course is only offered once a year, that decision has greater ramification. Since an integral part of Capstone is to expose students to experiences to better prepare them for their careers, it was important to balance the convenience of student needs with keeping the principal of Capstone in tact. This can affect how a student might respond to the pressures of a real job. Would they simply walk away or formulate successful strategies to confront it head on – something they perhaps learned in Capstone.

This poster will discuss the various issues involved in the different approaches and the solution developed: concurrent running projects with an added common time to enable cross project interaction. It will highlight the undergraduate research model that has been developed at UNH-M over the past decade. Various materials have been published in computing educational forums on the format and achievements of this model and feedback from colleagues at other institutions has led to the evolution of a strong internal Capstone experience [1, 2, 3, 4]. This is an important model as it helps create an alternative to the traditional work-embedded model that can add strain to a computing department as much work is needed to build industry relationships. Though UNH-M has built its own network of industry partners, it has more freedom in how to engage them without the need of their companies providing places for students.

References

[1] Michael Jonas. Capstone experience: engaging students in speech processing to excite them about STEM: faculty poster. *Journal of Computing Sciences in Colleges*, 26(6):180–181, 2011.

[2] Michael Jonas. Capstone experience: lessons from an undergraduate research group in speech at UNH Manchester. In *Proceedings of the 2011 conference on Information technology education*, pages 275–280, 2011.

[3] Michael Jonas. Capstone experience: achieving success with an undergraduate research group in speech. In *Proceedings of the 15th Annual Conference on Information technology education*, pages 55–60, 2014.

[4] Michael Jonas. Capstone: impact of a successful undegraduate research program: faculty poster abstract. *Journal of Computing Sciences in Colleges*, 31(6):50–51, 2016.

What Makes Students' Capstone Projects Successful?[*]

Poster Abstract

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

Practices of computer-science majors in capstone-project developments for the last 15 years have been analyzed revealing the lack of novelty, weakness of applications, and low quality of the project- related artifacts. The project criteria are revised and changed guiding the students to work successfully on the state-of-the-art challenging research projects, to build solid project portfolios, and to go "an extra mile" in their starting careers. The examples of students' outstanding capstone projects (developed within the frame of the revised criteria) are reviewed.

The Capstone Project is the last course (some sort of the Master Theses) in the Computer Science curriculum. Students select project topics and work individually on the design and implementation of moderately large software systems as the deliverables for this course. At the end of the semester, every student presents the results of her/his project work and the system demo to the rest of the class.

In the past (before the course revision), many students (about 70% of the class) made "shopping" for "reasonably-good" grades and selected basic topics (which are "popular" on the Internet) for their capstone projects, e.g., "Library Online Management System", "Furniture Online Shop", "Online Bus Reservation System", "Mobile Billing System", etc. These "shallow" projects did not reveal neither the students' actual potentials nor their readiness to deal with the career challenges of applying the state-of- the-arts technologies in the high-tech company projects.

Several years ago, Rivier's CS/IT faculty had reviewed practices of running the Capstone Project course, revised the course criteria and even renamed the

[*]Copyright is held by the author/owner.

course to the Professional Seminar. Course instructors have started promoting class discussions covering various modern societal issues, e.g., “Environments Become Smart”, “Life after the Internet”, “How Biology Became an Information System”, “Environment in Human-Centered Systems,” and others. Also, every student has led discussions on two selected peer-reviewed articles from professional journals and/or books on the future of computing. There are a few examples of the successful students’ survey reviews: “A New Experimental Website Converts Photos into 3D Models”, “Taking Measure of SaaS Reliability”, “An Overview of Malware”, “Automatic Information Extraction from Large Websites”, “Online Analytical Processing”, “Computers ’Taught’ to Search for Photos Based on Their Contents”, “Quantum Computing”, “The Future of Electronic Displays”, and others. This search for knowledge helps students stay at the cutting edge of computer science. In addition to these activities, students read the Craft of Research textbook that helps them develop research skills.

In this poster, the revised course criteria (such as the orientation on practical application, novelty, the structured methodology of project development, quality of the project-related artifacts, creating a system demo, and building a project portfolio), the “extra-mile” opportunities, and examples of students’ outstanding CS capstone projects are considered in details.

For selecting of a project topic, students are encouraged to conduct the feasibility analysis of the system potential users and the expected valuable services provided by the proposed system to the users. Students, who have experience of working in high-tech companies or taking internships there, have typically selected the interesting, challenging project topics promising strong practical applications.

Natural curiosity and novelty bring the talented students the greatest ecstasy that governs them in discovery endeavors in their academic lives and, later, in professional careers. The students have tried to introduce new elements in their capstone projects and got the outstanding results. The novelty has featured in the wide variety of students’ recent projects, including “Modeling a Digital Video Cluster”, “Design and Implementation of an IoT Smart Farming System”, “Applying XamarinTM Cross Platform Framework for Smart Glasses Design”, “Personal Encrypted Talk Tool”, “Cloud-based Searchable Storage Cryptosystem”, “Development of the Personal Accounting WebApp with AngularTM and SpringTM Frameworks”, “A Public Resource Computing Platform for Simulating N-Body Galaxies”, “A Multi-Domain Musician’s Web-service Using Ruby-on-Rails, SOAP, FLEX, and AJAX”, “Secure Online Biometric Authentication Solution”, “Visualization of Multivariate Data through Chernoff Faces”, “Tactus: A Learning Game for Children with Autism”, “DEM846: Digital Elevation Modeling”, and others. Some of these projects are reviewed in the poster.

As the general requirement for this course, students should follow the established project- development procedures ("stages" of project planning, feasibility and functional analyses, system design, code programming, and system prototype testing) and standards. Typically, this process takes 12-14 weeks. Prior to these activities, students develop mini projects on small system prototypes in various core and elective CS courses, including the Object-Oriented System Analysis Design, Computer Architecture, Operating Systems, Software Engineering, Database Management Systems, Multimedia Web Development, Computer Security, Java Programming, C/C++ Programming, Exploring Perl Ruby, Computer Graphics, and Software Quality Assurance. The acquired knowledge and skills have been successfully used by students in their work on the capstone projects.

In the project evaluations, the course instructors draw special attention to the quality of various project-related artifacts (e.g., Unified Modeling Language diagrams, conceptual client-server architecture, user-computer interfaces, windows navigation diagrams, normalized data tables, entity- relational diagrams, and codes). After every stage of the system development process, the corresponding test plans have to be created. The special requirements were developed for the high quality of the project reporting (the style of project report organization, citation of sources, image quality, etc.) as well as for the project oral presentations. These efforts help students develop strong "soft skills" that have become in the growing demand in high-tech companies and in the scholar community.

Finally, at the end of the course, students are required to submit the project portfolios with all the project documentation burnt on a CD or accessible from the student's personalized website. In many cases, students use these portfolios during the internship or job interviews and for the job promotions.

Faculty encourage students to pursue "extra-mile" scholarly activities (e.g., publish the first article in a peer-reviewed research journal, or present a paper at a conference) and share their research results with the global community of scholars. Several talented students included their capstone-project portfolios into applications for the further studies in Ph.D./Computer Science programs at M.I.T., W.P.I., UMASS-Lowell, UNH, and other universities. Upon completing successfully these programs, they continue collaborating with our department in many ways, including the lecturing, the supervising of our interns, and the students' mentoring.

In the course evaluations, students stated that they became deeply engaged in capstone-project activities through examining the challenging problems related to the real-world applications of the state- of-the-arts computing technologies.

Assessment of Computer Science Courses in the Context of a Global Knowledge Economy*

Poster Abstract

Viktoria Popova and Sofya Poger
Department of Institutional Assessment
Department of Computer Science
Felician University
Lodi, NJ 07644
{popovav,pogers}@felician.edu

The role of assessment in academic practices is usually associated with evaluating students' knowledge, skills, and competencies exclusively within a given subject area. Thus, a course in Computer Programming is not likely to include assessment of students' persuasive communication skills. Traditional teaching and assessment have taken a very targeted (and siloed) approach to introducing and practicing both applied and theoretical fields of knowledge. The shortcomings of this practice may not have explicitly affected student readiness to enter the workforce in the last millennium. However, it has become increasingly evident in the last 20 years that the exponentially expanding global knowledge economy requires that knowledge workers, such as professionals in computer science fields, are adept not only in their field of study: graduates should be able to demonstrate proficiency in "Human Skills," as well as be able to operate as "Business Enablers." The New Foundational Skills of the Digital Economy report by the Burning Glass [1], reveals the need expressed by employers across various sectors for the new workforce to be equipped with a broader set of skills than that usually required by a single discipline.

To address this need, the Felician University Computer Science Program, housed in the new Institute for Information Sciences, has developed a unique approach, which prompts students to practice a set of "human" and business skills. To move beyond assessing computer science students exclusively for their computer skills, at Felician University, we emulate a business environment, so

*Copyright is held by the author/owner.

that computer science students can obtain and practice a variety of communication skills and training/instruction skills that are required for most business experiences.

In Felician's revision of the BS in Computer Science program, we have established a dual assessment track: one is focused on the employability skills based on the Burning Glass study (cited above); and the second, on the depth of student learning through a sustained project over their degree. In the first term of the first year, students take Information Sciences 100, Information and Knowledge. This course addresses the 14 skills identified in the Burning Glass Report. Students initiate their portfolio (which has a tab for each employability skill) and are expected to contribute leaning assets to that portfolio throughout their entire degree. To facilitate that process, Computer Science faculty have developed a matrix wherein relevant employability skills are identified within each course (i.e., project management, collaboration, data analysis, etc.).

Faculty agree to let students use their projects for course work where relevant (at the faculty member's discretion for relevance). For example, if a student is taking a Database course, and thehomework asks them to normalize a database, they can perform the assignment in their project rather than using the sample in the textbook. Thus, the assessment is interwoven throughout the student's coursework.

To assess students' abilities in integrating discipline specific skills with communication and analytical skills, the following activities have been implemented throughout a number of Computer Science classes at Felician University: project presentation, project presentation critique, in-class discussions and feedback, students participating in delivering a "flipped classroom" teaching modality, and reciprocal tutoring. For example, in the Software Engineering and AI courses the following activities have been implemented:

- A student presents his/her short-term project to the class; classmates provide feedback in the form of a discussion; classmates participate in an anonymous survey by providing a detailed critique on the project, its content, presentation skills, and engagement value.
- Following the project presentation, the instructor provides individual assessment to each student by offering one-on-one discussions for the purpose of addressing student's communication skills and recommendation of further steps for improvement.
- The student's next short-term project is assessed based on the degree of each student having incorporated all of the suggested improvement steps.

In the Computer Vision course, when covering a topic of Image Segmentation, the class is divided into small groups and each group is assigned with a unique algorithm, which student will be presenting to the entire class. The class discusses pros and cons of each technique and students attempt to tutor each other on hard-to-understand issues and finer points. At the final stage, the students are given a quiz involving the entire body of the presented material, and the quiz results serve as an assessment of this approach for developing learning and communication skills.

In the Discrete Structures course, following a lecture on a particular subject area, the class is divided into small groups (3 – 4 students) and each group is assigned a unique problem to be solved. Each group presents a solution to the class, which discusses the accuracy of the solution or, in some cases, suggests a more efficient solution. Following this activity, each student is assigned with a homework task that requires finding solutions to similar problems. This process supports assessment of both student learning of a subject area ("hard skills"), as well as evaluation of communication and critical thinking skills.

As a result of these activities, we are able to observe an improvement in students' application of "human skills." More importantly, students demonstrate an increased willingness and enthusiasm in communicating their "hard skills" via different modes of "soft skills." This point is critical, given that for a mastery of "hard skills" to be highly marketable in the context of global knowledge economy, students will be able to use soft/human skills as a conduit for representing and validating their expert knowledge.

References

[1] Will Markow, Debbie Hughes, and Andrew Bundy. The new foundational skills of the digital economy: developing the professionals of the future. Business-Higher Education Forum, Washington, District of Columbia, 2018.

LibreFoodPantry: Developing a Multi-Institutional, Faculty-Led, Humanitarian Free and Open Source Software Community*

Poster Abstract

Karl R. Wurst[1], *Stoney Jackson*[2], *Heidi J. C. Ellis*[2], *Darci Burdge*[3], *Lori Postner*[3]
[1]*Computer Science Department*
Worcester State University, Worcester, MA 01602
kwurst@worcester.edu
[2]*Computer Science and Information Technology Department,*
Western New England University, Springfield, MA 01119
stoney.jackson@wne.edu, heidi.ellis@wne.edu
[3]*Department of Mathematics,*
Computer Science and Information Technology,
Nassau Community College, Garden City, NY 11530
darci.burdge@ncc.edu, lori.postner@ncc.edu

Engaging students in humanitarian free and open source software (HFOSS) projects allows them to gain real-world software development skills while helping society. For years the authors have been working to encourage student and faculty participation in HFOSS projects and communities, but they have found that participating in an existing HFOSS project, although ripe with learning opportunities, presents a number of hurdles for faculty and students. An alternative to joining an existing HFOSS project community is to participate in a faculty-led HFOSS project. These projects provide the instructor with more control over the learning environment, but often lack an active community outside of the classroom.

This poster describes LibreFoodPantry, a multi-institutional effort to engage a community of developers in creating humanitarian open source projects

*Copyright is held by the author/owner.

to support their on-campus food pantries. Starting a faculty-led HFOSS project involves making decisions not only about the features of the project but also about community norms, tool choices, project development workflow, and inter-institution cooperation.

This poster provides an overview of the creation of the LibreFoodPantry community who is developing a suite of projects that support on-campus food pantries. It describes instances of using LibreFoodPantry projects in various classroom settings over three semesters, the lessons learned from these experiences, and the resulting discussions and decisions made by the LibreFoodPantry Coordinating Committee. This process has led to a community dedicated to easing the on-ramp for faculty who want to help their students contribute to an HFOSS project.

The LibreFoodPantry website (http://librefoodpantry.org) has links to the vision, mission, and code of conduct for the community, as well as documentation on tools, development processes, and workflows. The site also has links to the constituent projects' codebases, issue trackers, and communication channels.

Student Reflections on Learning in HFOSS[*]

Poster Abstract

Gregory W. Hislop[1], *Heidi J. C. Ellis*[2], *Becka Morgan*[3]
[1]*Department of Information Science*
Drexel University
Philadelphia, PA 19104
hislop@drexel.edu
[2]*Computer Science and Information Technology Department*
Western New England University
Springfield, MA 01119
ellis@wne.edu
[3]*Computer Science Division*
Western Oregon University
Monmouth, OR 97361
morganb@wou.edu

Humanitarian Free and Open Source Software (HFOSS) projects provide a rich learning context for students with the additional motivation of developing software to help solve societal challenges and improve the human condition. HFOSS projects also provide the benefit of allowing students to establish a professional portfolio of contributions while still in school. Student involvement in HFOSS has been shown to provide opportunity for learning software engineering, technology, and business skills as reported in [1, 2]. These studies used a quantitative approach based on results of Likert surveys. This poster presents an effort to expand and confirm these results via a qualitative investigation into student experiences in participation in HFOSS in three different undergraduate courses at three different academic institutions.

The instructors all have several years of experience involving students in HFOSS projects. All three courses employed student reflective writing about their class experiences. These writings were used as a source to gather unstructured observations about student learning. The writings utilized no specific

[*]Copyright is held by the author/owner.

prompting about experience with an HFOSS project and one goal was to see which topics became apparent and were common across the students' reflections. The qualitative investigation focused on the following three research questions:

1. What types of knowledge and skill do students report developing by working on an HFOSS project?
2. Do students indicate that HFOSS participation provides motivation or affects confidence about pursuing computing careers?
3. What impact does it have on students to interact with the development community of an HFOSS project?

The poster will provide an overview of the classes and HFOSS projects, report on student observations, and summarize the themes that emerge from student reflections. These themes include knowledge and skill, motivation and confidence, community interactions, and student contributions. The poster will also outline suggestions for adoption for instructors who are interested in exploring student participation in HFOSS. Suggestions include starting with small involvements, collaborating with the HFOSS community, considering extracurricular opportunities and connecting with the community of instructors who are supporting student involvement in HFOSS.

ACKNOWLEDGMENT

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] Heidi JC Ellis, Gregory W Hislop, Stoney Jackson, and Lori Postner. Team project experiences in humanitarian free and open source software (hfoss). *ACM Transactions on Computing Education (TOCE)*, 15(4):1–23, 2015. DOI=http://dx.doi.org/10.1145/2684812.

[2] Gregory W Hislop, Heidi JC Ellis, S Monisha Pulimood, Becka Morgan, Suzanne Mello-Stark, Ben Coleman, and Cam Macdonell. A multi-institutional study of learning via student involvement in humanitarian free and open source software projects. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 199–206, 2015.