The Journal of Computing Sciences in Colleges

Papers of the 37th Annual CCSC Southeastern Conference

November 3rd-4th, 2023
Coastal Carolina University
Conway, SC

Bin Peng, Associate Editor
Park University

Adam Lewis, Regional Editor
Athens State University

Volume 39, Number 5 **November 2023**

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.

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:

Scott Sigman, President (2024), ssigman@drury.edu, Mathematics and Computer Science Department, Drury University, Springfield, MO 65802.

Bryan Dixon, Vice President/President-Elect (2024), bcdixon@csuchico.edu, Computer Science Department, California State University Chico, Chico, CA 95929.

Baochuan Lu, Publications Chair (2024), blu@sbuniv.edu, Division of Computing & Mathematics, Southwest Baptist University, Bolivar, MO 65613.

Ed Lindoo, Treasurer (2026), elindoo@regis.edu, Anderson College of Business and Computing, Regis University, Denver, CO 80221.

Cathy Bareiss, Membership Secretary (2025), cathy.bareiss@betheluniversity.edu, Department of Mathematical & Engineering Sciences, Bethel University, Mishawaka, IN 46545.

Judy Mullins, Central Plains Representative (2026), mullinsj@umkc.edu, University of Missouri-Kansas City, Kansas City, MO (retired).

Michael Flinn, Eastern Representative (2026), mflinn@frostburg.edu, Department of Computer Science & Information Technologies, Frostburg State University, Frostburg, MD 21532.

David R. Naugler, Midsouth Representative (2025), dnaugler@semo.edu, Brownsburg, IN 46112.

David Largent, Midwest Representative(2026), dllargent@bsu.edu, Department of Computer Science, Ball State University, Muncie, IN 47306.

Mark Bailey, Northeastern Representative (2025), mbailey@hamilton.edu, Computer Science Department, Hamilton College, Clinton, NY 13323.

Shereen Khoja, Northwestern Representative(2024), shereen@pacificu.edu, Computer Science, Pacific University, Forest Grove, OR 97116.

Mohamed Lotfy, Rocky Mountain Representative (2025), mohamedl@uvu.edu, Information Systems & Technology Department, College of Engineering & Technology, Utah Valley University, Orem, UT 84058.

Tina Johnson, South Central Representative (2024), tina.johnson@mwsu.edu, Department of Computer Science, Midwestern State University, Wichita Falls, TX 76308.

Kevin Treu, Southeastern Representative (2024), kevin.treu@furman.edu, Department of Computer Science, Furman University, Greenville, SC 29613.

Michael Shindler, Southwestern Representative (2026), mikes@uci.edu, Computer Science Department, UC Irvine, Irvine, CA 92697.

Serving the CCSC: These members are serving in positions as indicated:
Bin Peng, Associate Editor, bin.peng@park.edu, Department of Computing and Mathematical Sciences, Park University, Parkville, MO 64152.
Brian Hare, Associate Treasurer & UPE Liaison, hareb@umkc.edu, School of Computing & Engineering, University of Missouri-Kansas City, Kansas City, MO 64110.
George Dimitoglou, Comptroller, dimitoglou@hood.edu, Department of Computer Science, Hood College, Frederick, MD 21701.
Megan Thomas, Membership System Administrator, mthomas@cs.csustan.edu, Department of Computer Science, California State University Stanislaus, Turlock, CA 95382.
Karina Assiter, National Partners Chair, karinaassiter@landmark.edu, Landmark College, Putney, VT 05346.
Deborah Hwang, Webmaster, hwangdjh@acm.org.

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.

Gold Level Partner

Rephactor

Welcome to the 2023 CCSC Southeastern Conference

Welcome to the 37th Southeastern Regional Conference of the Consortium for Computing Sciences in Colleges. The CCSC:SE Regional Board welcomes you to Conway, SC, the home of Coastal Carolina University, for our second visit to this beautiful campus. The conference is designed to promote a productive exchange of information among college personnel concerned with computer science education in the academic environment. It is intended for faculty as well as administrators of academic computing facilities, and it is also intended to be welcoming to student participants in a variety of special activities. We hope that you will find something to challenge and engage you at the conference!

The robust conference program is highlighted by a four sessions of three tracks each, including engaging guest speakers, tutorials, student posters, a nifty assignment session and five sessions of high quality refereed papers. We received 28 papers this year of which 14 were accepted to be presented at the conference and included in the proceedings – an acceptance rate of 50%.

Two exciting activities are designed specifically for students – a research contest and an undergraduate programming competition, with prizes for the top finishers in each.

We especially would like to thank the faculty, staff, and students of Coastal Carolina for their help in organizing and publicizing this conference. Many thanks also to the CCSC Board, the CCSC:SE Regional Board, and to a wonderful Conference Committee, led by Conference Chair Dr. Jean French and programming contest coordinator Dr. Andy Digh. Thank you all so much for your time and energy.

We also need to send our deepest appreciation to our partners, sponsors, and vendors. Please take the time to go up to them and thank them for their contributions and support for computing sciences education — *CCSC National Partners:* Rephactor. *Sponsoring Organizations:* CCSC, ACM-SIGCSE, Upsilon Pi Epsilon. *Local Sponsors:* Archetype SC, Aynor Tire and Market, Domino's Pizza, Just Ask Hal Computer Repair Service, Medicine Mart Pharmacy, Nye's Pharmacy, SpartanTec, Horry Elective Cooperative, Palmetto Chevrolet, Santee Cooper, The Horry Independent.

We could not have done this without several excellent submissions from authors, the insightful comments from a great team of 26 reviewers, and the support from our editor Baochuan Lu. Thanks to all of you for helping to create such a strong program for this year's conference.

We hope you enjoy the conference and your visit to Coastal Carolina.

Kevin Treu
Furman University
Program Co-Chair

Adam Lewis
Athens State University
Program Chair

2023 CCSC Southeastern Conference Steering Committee

Jean French, Local Arrangements Chair ..... Coastal Carolina University
Paul Cerkez, Local Publicity Chair ..... Coastal Carolina University
Will Jones, Speakers Chair ..... Coastal Carolina University
Ross Foultz, Vendors Chair ..... Coastal Carolina University
Tally Wright, Local Sponsors Chair ..... Coastal Carolina University
Andy Digh, Programming Contest Co-Director ..... Mercer University
Fahad Sultan, Student Research Contest Director ..... Furman University
Steven Benzel, Nifty Assignments Co-Chair ..... University of North Georgia
Robert Lutz, Nifty Assignments Co-Chair ..... Piedmont University

Regional Board — 2023 CCSC Southeastern Region

Jean French, 2023 Site Chair ..... Coastal Carolina University
Jonathan Cazalas, Regional Board Chair ..... Florida Southern College
Kevin Treu, CCSC:SE Regional Representative ..... Furman University
Karen Works, Treasurer ..... Florida State University Panama City
Kevin Treu, Program Co-Chair ..... Furman University
Adam Lewis, Program Chair/Regional Editor ..... Athens State University
Stephen Carl, Publicity Chair ..... The University of the South
Jean French, Local Registrar ..... Coastal Carolina University
Kevin Treu, 2024 Site Chair ..... Furman University

Reviewers — 2023 CCSC Southeastern Conference

Farha Ali ..... Lander University, Greenwood, SC
Chris Alvin ..... Furman University, Greenville, SC
Scott Barlowe ..... Western Carolina University, Cullowhee, NC
Brian Bennett ..... East Tennessee State University, Johnson City, TN
Andrew Besmer ..... Winthrop University, Rock Hill, SC
Darrell Norman ..... Burrell Marymount University, Warrenton, VA
Prashanth Reddy BusiReddyGari .....
... . University of North Carolina at Pembroke, Pembroke, NC
Paul Cerkez ..... Coastal Carolina University, Surfside Beach, SC
Crystal Cox ..... Coastal Carolina University, Conway, SC
Lawrence D'Antonio ..... Ramapo College, Mahwah, NY
Hilmi Demirhan . University of North Carolina Wilmington, Wilmington, NC
Andy Digh ..... Mercer University, Macon, GA
Gulustan Dogan . University of North Carolina Wilmington, Wilmington, NC
Joe Dumas ..... University of Tennessee at Chattanooga, Chattanooga, TN
Wayne Goddard ..... Clemson University, Clemson, SC
Adrian Heinz ..... Georgia Gwinnett College, Lawrenceville, GA
Mark Holliday ..... Western Carolina University, Cullowhee, NC
Gongbing Hong ..... Georgia College and State University, Milledgeville, GA
William Jones ..... Coastal Carolina University, Conway, SC
Adam Lewis ..... Athens State University, Athens, AL
Ronald James .... Nowling Milwaukee School of Engineering, Milwaukee, WI
Adewale Sekoni ..... Roanoke College, Salem, VA
Scott Spurlock ..... Elon University, Elon, NC
Syed Fahad Sultan ..... Furman University, Greenville, SC
Henry Suters ..... Carson-Newman University, Knoxville, TN
Kevin Treu ..... Furman University, Greenville, SC
Karen Works ..... Florida State University, Panama City, FL
Fei Zuo ..... University of Central Oklahoma, Edmond, OK

Sentiment and Topic Modeling Analysis of Reddit Posts on Changing Ones Major*

Nathan Green[1] *and Karen Works*[2]
[1]*School of Technology and Innovation*
Marymount University
Arlington, VI 22207
ngreen@marymount.edu
[2]*Department of Computer Science*
Florida State University
Panama City, FL 32405
keworks@fsu.edu
Both authors contributed equally to this research.

Abstract

Producing professional, responsible alumni is a primary mission of all institutions of higher education. Hence it is important to support students in their progress towards this goal, particularly when they consider changing their major. This research aims to evaluate the sentiment and topics used in Reddit posts on changing one's major. By looking at the commonalities in these posts, we can better understand the questions and concerns of students changing their majors.

1 Introduction

Within the first three years of school, roughly one third of undergraduate students change their major at least once [10]. Changing one's major can be very stressful. In fact, the life regret most often identified by Americans is their educational choices [16]. It is beneficial for students changing their majors to receive guidance from academic advisors. [11] found that academic advising of

students changing their majors can have a positive effect on student's academic performance. However, students may not always reach out for such support. Many such students seek help from social interaction and discussion websites such as Reddit. In this paper we explore the posts made by students who seek such support via Reddit using sentiment analysis and topic extraction.

2 Related Work

Automated techniques to discover trends is a current topic in natural language processing. One such technique that utilizes a probabalistic approach is Latent Dirichlet Allocation (LDA) [2]. LDA has been applied to the discovery of trends in many domains [4, 1, 15].

There are many projects that apply text mining techniques to monitor education trends. [14] used unsupervised text mining to identify trends in the integration of ethics into introductory CS courses. [9] used LDA and trending analysis to identify common CS topics and trends from selected online forum posts. A standard database query of articles with keywords related to 'computer' and 'instructional technologies' was used in trending analysis in technology education by [13]. Non-negative Matrix Factorization (NMF) [17] (a matrix factorization approach) was used to model topics in programming problems scrapped from the Aizu Online Judge system [7]. Both LDA and NMF models were used by [12] to locate common topics in online beginning Python programming tutorials.

To the best of our knowledge, no one has examined Reddit posts on changing one's major.

3 Methodology

We harvested data via Reddit using the Reddit API on March 10, 2023, on posts that contain "changing major" and similar phrases. All but six of these posts were from individual users (Three users made two posts). The origin of these posts is from 1,365 different Subreddits. We categorize these Subreddits based upon the community that they support as either employment, general advice, professional, school or other. The majority of these posts were in school or professional Subreddits (Figure 1).

Then any Reddit posts with no comments were thrown out (leaving N = 8,987). The extracted posts, consisting of a title and text, were cleaned by removing any non-alphanumeric characters (i.e., special characters and hyperlinks). The text was then converted to lower case. Common stop words [8] and any phrases similar to "changing major" were removed from the text. Finally,

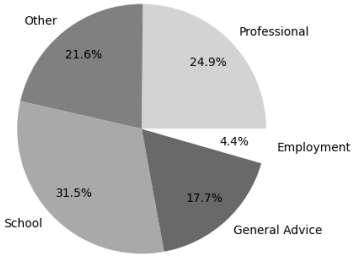


Figure 1: SubReddit Categories

synonyms were replaced with a single term [3]. The resulting text was divided into unigrams for topic and frequency analysis.

As a first step to evaluating the posts, we generated Word Clouds, a popular approach for visualizing the most frequently used terms across different posts where the font size depicts how common the term is [5].

Then we applied an NLP (natural language processing) approach for automatically extracting and classifying sentiment from text called Sentiment Analysis. We used VADER (Valence Aware Dictionary and sEntiment Reasoner), a Sentiment Analysis method designed to evaluate social media context [6]. For our evaluation of the sentiment of each post, we used the compound value, a normalization of the negative, neutral, and positive values.

Then an LDA [2] model was developed for our data set. From the results, we identified the topics and simultaneously classified the posts among these different topics.

4 Results and Discussion

4.1 All Posts

4.1.1 Word Cloud

The Word cloud in Figure 2 visually represents the most frequently used words in the title and text of these posts. It is noted that the most frequently used words within these posts describe actions (“work”, “make”, “think”), and things that influence these (“school”, “year”, “time”, “job”).

4.1.2 Sentiment Analysis

Sentiment analysis across the data collected (Figure 3) shows that the vast majority ($> 70\%$) of the posts were positive (sentiment value between .5 and 1), very few if any ($< 5\%$) were neutral (sentiment value $>$ -.5 and $<$ .5),

Figure 2: Word Cloud

and roughly 25% were negative (sentiment value between -.5 and -1). The distribution of these values (Figure 4) shows that the majority of the posts had strong sentiments (i.e., close to 1 very positive, or close to -1 very negative).

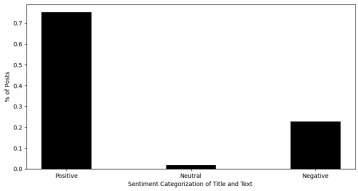


Figure 3: Sentiment Categorization

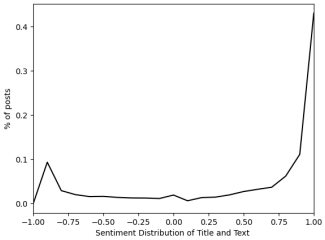


Figure 4: Sentiment Distribution

4.1.3 Topic Detection

Table 1 contains the 10 automatically extracted topics, our self-annotated label of the topic, and the top extracted 7 terms in each topic. We see a variety of subjects on academic paths, academic programs, balancing life, career advice, finances, peer advice, STEM courses, and STEM program. It is interesting that

academic (Paths or Programs), balancing life, and STEM (courses or program) are general areas that occur more than once in the topic labels.

Figure 3 shows the intertopic distance map of the 10 topics. The size of each circle depicts how frequently the topics occurred in the set of posts. Hence, topic 1 has the largest bubble. The distance shows that the following topics are closely related: topics 1 (Finances) and 3 (Academic Paths), and topics 2 (Career Advice) and 6 (Balancing Life).

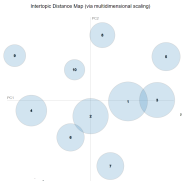


Figure 5: Intertopic Distance Map

Table 1: Topics and Terms

No	Topic Label	Terms per Topic
1	Finances	"job", "time", "work","pay","money","loan","full"
2	Career Advice	"job","degree","career","work","time","experience","advice"
3	Academic Paths	"student", "university", "research", "high", "biology", "english", "chemistry"
4	Peer Advice	"friend", "time", "told", "said", "mom", "got", "back"
5	STEM: Courses	"mom", "math", "course", "programming", "computer", "physic", "science", "learning"
6	Balancing Life	"time", "life", "going", "thing", "make", "much", "think"
7	STEM: Programs	"engineering", "science", "degree", "computer", "program", "university", "currently"
8	Balancing Life	"one", "time", "day", "could", "life", "back", "first"
9	Academic Progress	"semester", "gpa", "course", "take", "student", "first", "credit"
10	Balancing Life	"people", "comment", "make", "game", "thing", "art", "good"

4.2 Posts by Sentiment Categories

We now look at the posts broken into the three sentiment categories, namely positive (sentiment value >=.5), neutral (sentiment value between -.5 and .5), and negative (sentiment value <= -.5).

4.2.1 Word Clouds by Sentiment Categories

The Word clouds in Figures 6, 7, and 8 respectively visually represents the most frequently used words in the title and text of the posts categorized by sentiment

as positive, neutral and negative. The top terms in the positive, neutral, and negative are respectively ("school", "year", "time", "degree"), ("degree", "year", "science", "school"), and ("year", "time", "school", "job"). The neutral has fewer terms with a high frequency of use than the other two. This is as expected given that the size of the neutral post population is significantly smaller then positive or negative. All include "school" and "year". It is interesting that "job" is a common term in negative posts.

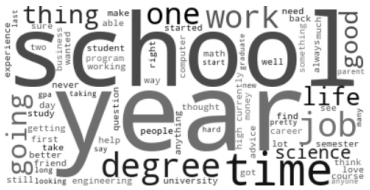


Figure 6: Word Cloud- Positive Sentiment

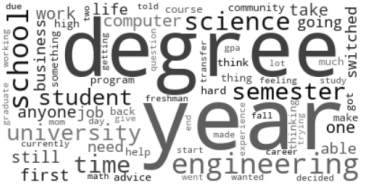


Figure 7: Word Cloud- Neutral Sentiment

Figure 8: Word Cloud- Negative Sentiment

4.2.2 Topic Detection by Sentiment Categories

Tables 2, 3, and 4 respectively contain the 5 automatically extracted topics, our self-annotated label of the topic, and the top extracted 7 terms in each topic for the positive, neutral and negative posts.

The topic labels in the positive posts are on balancing life, career advice, and STEM. For the neutral posts, the topics are academic progress, career advice, and STEM. While the negative posts also have the balancing life topic label, it have new labels not seen before, namely, parents and personal issues.

Table 2: Topics and Terms- Positive

No	Topic Label	Terms per Topic
1	Balancing Life	"time", "one", "people", "life", "thing", "friend", "could"
2	Career Advice	"time", "degree", "job", "work", "going", "university", "one"
3	STEM: Career Advice	"job", "degree", "time", "engineering" , "work", "much", "help"
4	STEM: Programs	"degree" , "science", "course", "math" , "computer", "engineering", "work"
5	Balancing Life	"time", "student", "first", "make", "thing", "work", "going"

Table 3: Topics and Terms- Neutral

No	Topic Label	Terms per Topic
1	Career Advice	"degree", "job", "hard", "able", "get" , "year", "graduate"
2	Academic Progress	"student", "business", "engineering", "year", "since", "get", "first"
3	Academic Progress	"year", "time", "get", "still", "semester", "switched", "went"
4	STEM: Programs	"science", "computer", "year", "degree", "think", "take", "get"
5	STEM: Programs	"engineering", "school", "year", "degree", "university", "semester", "get"

Table 4: Topics and Terms- Negative

No	Topic Label	Terms per Topic
1	Balancing Life	"life", "one", "soul", "job", "going", "take", "need"
2	Personal Issues	"student", "semester", "degree", "going", "health", "one", "work"
3	Parents	"one", "back", "thing", "mother", "much", "got", "make"
4	Parents	"think", "mom", "thing", "could", "issue", "money", "back"
5	Balancing Life	"work", "job", "life", "back", "one", "much", "day"

Figures 9, 10, and 11 respectively show the intertopic distance map of the 5 topics for the positive, neutral, and negative posts. In the positive posts, topic 1 (Balancing Life) is larger than the other 4 topics, and topics 2 through 5 (Career Advice, STEM: Career Advice, STEM: Programs, and Balancing Life) are closely related. While in the neutral posts, all the topics are relatively

the same size and topics 4 and 5 (both STEM: Programs) are closely related. Finally, in the negative posts, topic 1 (Balancing Life) is larger than the other 4 topics, and topics 1 (Balancing Life), 2 (Personal Issues), and 3 (Parents) are closely related.

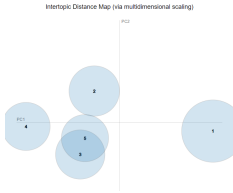


Figure 9: Intertopic Distance Map-Positive

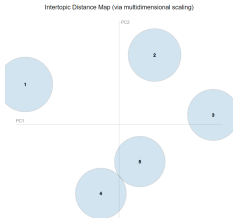


Figure 10: Intertopic Distance Map-Neutral

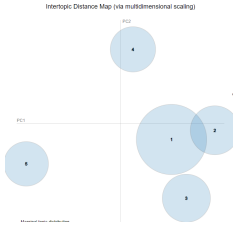


Figure 11: Intertopic Distance Map-Negative

5 Conclusion

Supporting students during the process of changing their major is critical to ensuring their success and ultimately the success of their institutions. We performed sentiment and topic analysis evaluation of Reddit posts posted prior to March 10, 2023, that contain "changing major" and similar phrases. The

results show clearly that overall, these posts were positive and that negative posts contained topics concerning parents and personal issues. In the future, we hope to look deeper into this data set to explore trending analysis on the original and new majors and evaluate the advice provided to these posts by other users.

References

[1] Saqib Aziz, Michael Dowling, Helmi Hammami, and Anke Piepenbrink. Machine learning in finance: A topic modeling approach. *Econometrics: Econometric & Statistical Methods - Special Topics eJournal*, 2019.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[3] Jaime Carbonell, Steve Klein, David Miller, Mike Steinbaum, Tomer Grassiany, and Jochen Frei. Context-based machine translation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 19–28, Cambridge, Massachusetts, USA, 8 2006. Association for Machine Translation in the Americas.

[4] Michael Dowling, Anke Piepenbrink, Saqib Aziz, and Helmi Hammami. Machine learning in finance: A topic modeling approach, 02 2019.

[5] Marti A Hearst, Emily Pedersen, Lekha Patil, Elsie Lee, Paul Laskowski, and Steven Franconeri. An evaluation of semantically grouped word cloud designs. *IEEE transactions on visualization and computer graphics*, 26(9):2748–2761, 2019.

[6] Clayton Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 216–225, 2014.

[7] Chowdhury Md Intisar, Yutaka Watanobe, Manoj Poudel, and Subhash Bhalla. Classification of programming problems based on topic modeling. In *Proceedings of the 2019 7th International Conference on Information and Education Technology*, ICIET 2019, page 275–283, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Karen Sparck Jones, Peter Willett, et al. *Readings in information retrieval*. Morgan Kaufmann, 1997.

[9] Habib Karbasian and Aditya Johri. Keeping curriculum relevant: Identifying longitudinal shifts in computer science topics through analysis of Q&A communities. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–7, 2021.

[10] Katherine Leu. Beginning college students who change their majors within 3 years of enrollment. data point. NCES 2018-434. *National Center for Education Statistics*, 2017.

[11] Deborah McKenzie, Tony Xing Tan, Edward C. Fletcher, and Andrea Jackson-Williams. Major re-selection advising and academic performance. *NACADA Journal*, 37(1):15–25, 01 2017.

[12] Laura Oliveira Moraes and Carlos Eduardo Pedreira. Clustering introductory computer science exercises using topic modeling methods. *IEEE Transactions on Learning Technologies*, 14(1):42–54, 2021.

[13] C Nurzhanov, V Pidlisnyuk, L Naizabayeva, and M Satymbekov. Research and trends in computer science and educational technology during 2016-2020: Results of a content analysis. *World Journal on Educational Technology: Current Issues*, 13(1):115–128, 2021.

[14] Sarah Parsons and Natalia Khuri. Discovery of research trends in computer science education on ethics using topic modeling. In *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 885–891, 2020.

[15] Paritosh Pramanik and Rabin K. Jana. Identifying research trends of machine learning in business: a topic modeling approach. *Measuring Business Excellence*, 2022.

[16] Neal J Roese and Amy Summerville. What we regret most... and why. *Personality and Social Psychology Bulletin*, 31(9):1273–1285, 2005.

[17] D Seung and L Lee. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

Preparing Students for Software Production with DevOps: A Graduate Course Approach*

Brian T. Bennett
Department of Computing
East Tennessee State University
Johnson City, TN 37614
bennetbt@etsu.edu

Abstract

Software Engineering education continues to describe classical methods without fully embracing modern practices. DevOps combines Software Engineering practices with the production and operations of the software itself. This study describes a graduate course in software production that primarily focuses on DevOps practices, while minimally discussing Software Engineering. Student performance in each topic was tracked through formative assessment (on quizzes) and summative assessment (on exams). Results showed that students' performance for learning outcomes improved an average of 22% after participating in lectures, discussions, exercises, and projects, validating the course design.

1 Introduction

Skills in Software Engineering (SE) remain essential for success in today's development industry. The 2020 Computing Curricula Report from ACM notes, "SE emphasizes the use of appropriate software development practices and the integration of engineering rigor with the ability to apply advanced algorithms and data structures developed in computer science" [1]. However, the literature suggests that focusing on these skills alone could result in missing other

essential topics beyond SE [6]. DevOps incorporates both Development and Operations in a way that can deliver value to the customer in a continuous stream. According to Pang et al. [5], few institutions teach topics in DevOps that prepare students for software production using a DevOps methodology.

This study follows the implementation of a DevOps-focused course, "Software Production," offered to graduate students in Spring 2023. The course included twelve graduate students in the initial offering. The course was designed to teach production-based topics after typical software engineering education. Following a review of relevant topics from the literature, the course development, learning outcomes, major topics, and pedagogical design are discussed. The course assessment focused on major topics and learning outcomes using formative and summative assessment methods.

2 Literature Review

The literature on DevOps Education is not extensive. However, a few studies give insights into appropriate topics to cover in a DevOps-focused course. Table 1 shows the topics and their frequency across five individual studies.

Alves & Rocha [2] discuss the challenges of introducing technical and non-technical DevOps concepts in a project-oriented undergraduate course. The authors note several categories of DevOps concepts presented in their course. These concepts include product and project management, build processes, continuous integration, deployment pipeline automation, and monitoring and logging. The authors conclude in part that creating a culture that fosters DevOps practice is a key factor and encouraging the use of different technologies to teach concepts.

Bennett [3] reformatted a second undergraduate Software Engineering course through the DevOps lens. To do so, the author took pre-existing learning objectives and reformatted them to fit into a more focused course on DevOps's development side. As such, the course touched primarily on Software Engineering concepts like Configuration Management, Project Management, Testing, and Planning. However, the course also touched on DevOps concepts, such as Automation, Metrics Gathering, Infrastructure as Code, Continuous Integration, and Cloud or Microservice Architectures.

Hobeck, Weber, Bass, & Yasar [4] review DevOps courses taught at two separate universities: Carnegie Mellon University and Technische Universität Berlin in Germany. The authors list eleven topics required in the courses at both Universities: Introduction to DevOps, Virtual Machines, Containers, Security, Deployment Pipelines, Microservices Architecture, Service Meshes, Postproduction, and Disaster Recovery.

Pang, Hindle, & Barbosa [5] reviewed DevOps education using Grounded

Table 1: A summary of topics suggested for DevOps courses in the literature

	Alves & Rocha [2]	Bennett [3]	Hobeck et al [4]	Pang et al. [5]	Verdicchio [6]	Total
Automated Deployment Pipelines	X		X	X	X	4
Continuous Integration	X	X		X		3
DevOps Conceptual Overview		X	X		X	3
DevSecOps		X	X		X	3
Emerging / Microservices Architecture	X	X	X			3
Build Process	X			X		2
Cloud Computing		X			X	2
Configuration Management / Repository		X		X		2
Infrastructure as Code		X			X	2
Metrics and Telemetry	X	X				2
Project Management	X	X				2
Testing and Automated Testing		X		X		2
Agile Development					X	1
Containers & VMs			X			1
Disaster Recovery			X			1
Infrastructure as a Service					X	1
Organizational Culture					X	1
Planning and Estimation		X				1
Postproduction			X			1
Service Meshes			X			1

Theory qualitative analysis. They identified five significant DevOps terms — Continuous Integration, Testing, Build, Repository, and Deployment — and used these terms to conduct a systematic review. They noted that DevOps and its corresponding topics were not present in the curriculum of many institutions. Only 12.8% of the institutions studied included Continuous Integration or Build Processes. However, a large majority (89.7%) taught Testing.

Verdicchio [6] notes that while Software Engineering continues to be a sought-after career, most curricula fail to provide industry-relevant practice. After reviewing relevant job advertisements, the author introduced eight modules that seek to tie software engineering to industry practice. These modules include Agile development and culture, DevOps concepts, Organizational culture related to DevOps, Cloud Computing, Infrastructure as a Service, Infrastructure as Code, DevOps Deployment Pipelines, and DevSecOps. The author also notes that some modules require a technical background (e.g., Cloud Computing and Infrastructure as Code), while others do not (e.g., Organizational Culture and DevOps Concepts).

As stated by [6] and illustrated by [3] and [5], many university courses focus extensively on the development side without students fully understanding

how that software moves to a production system. This review indicates that a DevOps course should cover several topics. The most crucial topics are Automated Deployment Pipelines, Continuous Integration, DevOps Concepts, DevSecOps, and Microservices Architectures. Other important topics include Build Processes, Cloud Computing, Configuration Management, Infrastructure as Code, Metrics and Telemetry, Project Management, and Testing.

3 Course Development

"Software Production" was created as a new course within the Software Engineering concentration for a Master of Science in Computer Science. Undergraduate courses that survey SE topics do not have time to take an in-depth look at the production part of the process. Therefore, offering either an elective course for undergraduates or a graduate course is a good way to cover DevOps topics.

3.1 Learning Outcomes

The Software Production course had three primary learning outcomes. At the end of the course, students should be able to:

1. Support and defend the selection of DevOps practices over traditional software development practices
2. Evaluate software development processes and their relationships to software production pipelines
3. Construct software development pipelines that utilize continuous integration, continuous testing, continuous deployment, and continuous delivery practices

3.2 Course Topics

The Software Production course covered ten major topics. Table 2 shows the relationship between the course's major topics and maps them to topics discovered during the literature review and the course learning outcomes.

The only topics from the literature review not covered in the Software Production course related to Software Engineering, specifically: Planning and Estimation and Project Management. While DevOps could be considered a way to manage projects, the Software Production course assumes students pick up project management and planning in previous courses. Therefore, an introductory Software Engineering course was considered a prerequisite for Software Production.

Table 2: Software Production Major Topics, mapped to Literature Topics and Course Learning Objectives

	Major Topic	**Literature Mapping**	**Learning Outcome(s)**
1	Preliminaries	VMs	0
2	Environments	Containers Service Meshes	0
3	The Cloud	Cloud Computing Infrastructure as a Service	0
4	DevOps Preliminaries	Agile Development DevOps Conceptual Overview Organizational Culture Metrics and Telemetry	1
5	DevOps Tools	Infrastructure as Code Configuration Management/Repository	1, 2
6	Deployment Pipeline	Automated Deployment Pipelines Build Process Continuous Integration Testing and Automated Testing	2, 3
7	Design Options	Emerging / Microservices Architecture	3
8	Postproduction	Postproduction Testing and Automated Testing	3
9	Secure Development	DevSecOps	1
10	Disaster Recovery	Disaster Recovery	3

3.3 Course Design

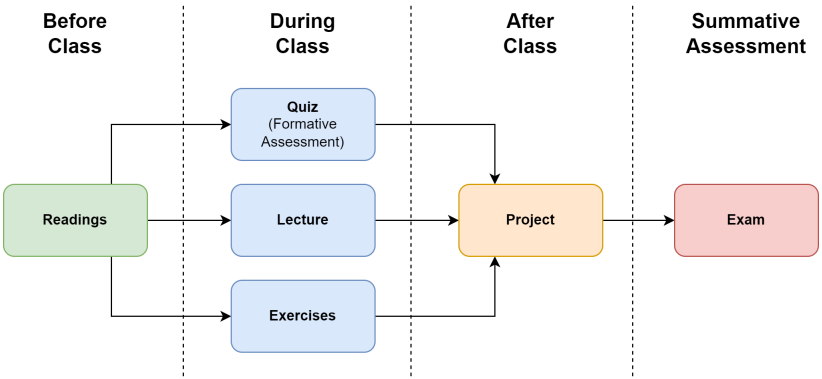


Figure 1: Course Design

Figure 1 shows the flow of information through the class. For each module, students completed reading assignments that fed into the week's quiz, lecture, and any associated exercises during class. Quizzes occurred at the beginning of each class session and were used as formative assessments. Lectures and discussions followed the quiz, and any applicable exercises were completed afterward.

Five projects were assigned during the semester that allowed students to apply what they had learned during class. The first two projects were individual explorations of tools like git and containerization. The last three projects were group assignments that asked students to set up a DevOps pipeline that moved code from development to the integration environment and then to the staging environment. Two exams were given during the semester as a summative assessment of students' learning. A topic did not appear on an exam unless the students had the opportunity to complete a project on the topic first.

4 Assessment Results

4.1 Topic Assessment

Figure 2 shows formative and summative assessment results for specific topics. Formative assessments were taken from each topic quiz, while summative assessments were taken from exam questions that map to each topic. Based on the percentage of "Does Not Meet" results, improvements were seen across all topics during the term except Topic 3 - The Cloud. However, Topic 3 did show that 16.7% of students moved from "Meets" to "Exceeds." In addition, students initially struggled with Topic 5, DevOps tools (63.6% "Does Not Meet"), but rebounded significantly after experiencing the lectures, exercises, and projects. These results indicate that students benefited from the course design, but select topics may need additional reinforcement, such as Topic 3 - The Cloud.

4.2 Learning Outcomes Assessment

Figure 3 shows formative and summative assessment results for the Student Learning Outcomes. Formative results were aggregated from the quizzes for each outcome based on the topic mapping in Table 2. Summative results were taken from exam questions that map to the topics and then to the outcomes shown in Table 2. Results indicate that half or nearly half of all students initially struggled with meeting the learning outcomes. At the formative stage, students would have read the materials but might not have had enough context through discussion, examples, and application to meet the outcome fully. However, over time, the experiences gained during the course resulted in much better summative results. Overall, nearly 77% of students met or exceeded all three learning outcomes by the end of the semester.

5 Discussion & Conclusion

Literature suggests that few Higher Education institutions provide adequate coverage of DevOps [5]. Therefore, creating DevOps-focused courses that pro-

Figure 2: Topic Assessment Results

vide students with an in-depth look at these topics is essential. This study outlined an effective method for teaching DevOps-centric topics, showing that a majority of students met or exceeded the learning outcomes after participating in the class activities. Additional project-based work might assist with understanding some topics, like Topic 3 - The Cloud, Topic 4 - DevOps Preliminaries, Topic 6 - Deployment Pipeline, and Topic 7 - Design Options. The course did not include an extensive project on Cloud deployment or a focus on continuous deployment. Therefore, adding a project focused on these areas might help to increase understanding of all of these topics in future iterations of the course.

Figure 3: Learning Objective Assessment Results

References

[1] *Computing Curricula 2020: Paradigms for Global Computing Education*, 12 2020.

[2] Isaque Alves and Carla Rocha. Qualifying software engineers undergraduates in devops - challenges of introducing technical and non-technical concepts in a project-oriented course. In *Proceedings of the 43rd International Conference on Software Engineering: Joint Track on Software Engineering Education and Training*, ICSE-JSEET '21, page 144–153. IEEE Press, 2021.

[3] Brian T. Bennett. Shifting traditional undergraduate software engineering instruction to a devops focus. *J. Comput. Sci. Coll.*, 36(5):129–138, 1 2021.

[4] Richard Hobeck, Ingo Weber, Len Bass, and Hasan Yasar. Teaching devops: A tale of two universities. In *Proceedings of the 2021 ACM SIGPLAN International Symposium on SPLASH-E*, SPLASH-E 2021, page 26–31, New York, NY, USA, 2021. Association for Computing Machinery.

[5] C. Pang, A. Hindle, and D. Barbosa. Understanding devops education with grounded theory. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 107–118, Los Alamitos, CA, USA, 10 2020. IEEE Computer Society.

[6] Michael Verdicchio. Creating a devops course. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 2*, SIGCSE 2023, page 1311, New York, NY, USA, 2023. Association for Computing Machinery.

The Game *Guillotine* as Inspiration for a Data Structures Course*

Chris Alvin[1], *Lori Alvin*[2]
[1]*Computer Science Department*
[2]*Mathematics Department*
Furman University
Greenville, SC 29613
{ chris.alvin†, lori.alvin }@furman.edu

Abstract

A typical course in data structures covers arrays, linked lists, stacks, queues, and efficiency of operations over those structures. In this paper, we consider the card game *Guillotine* as inspiration and motivation for a cohesive software project that integrates each of these linear data structures. We discuss the game's components, rules, and consider linear data structures and algorithms for those game components.

1 Introduction

Board games rely on many fundamental computing concepts and thus can be popular among students in a course to take ownership of computing concepts within an enjoyable context. Even in early CS courses, board games can be appealing. They can reinforce the rigidity of algorithms when considering player sequencing of 'moves'. They also provide a framework in which students can use a tabletop experience to engage in design and implementation choices about data structures and algorithms. For example, a tabletop representation can be useful for a student to engage in game-play to identify player use cases: possible player steps in a 'turn', interactions among game components, etc. Pedagogically we find games to be most appealing when testing a software

†Corresponding author

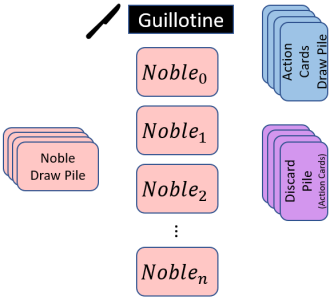


Figure 1: The *Guillotine* playfield.

Figure 2: Layout of a Noble card.

Table 1: Some descriptors of nobles in the *Noble Deck*.

Color	**Count**	**Group**	**Sample Noble Titles**
Purple	15	High Nobility	Baron, Marie Antoinette, Regent
Green	10	Government	Governor, Mayor, Sheriff
Red	11	Military	General, Lieutenant, Palace Guard
Grey	7	Innocents	Hero of the People, Martyr
Blue	7	Religious	Cardinal, Bad Nun, Wealthy Priest

implementation because with a board game, game states can be easily constructed and evaluated offline. In this paper, we discuss the game *Guillotine* [8] and consider different linear data structures for game components as part of a software implementation framework.

2 The Game

Guillotine is a card game for ages 12+ that is set in a cartoonish version of the French Revolution (1789–1799) in which the goal is to earn more points than your opponents by beheading people of the nobility. Hopefully this brief description allows the reader to appreciate the humor in the game tagline: "The revolutionary card game where you win by getting a head."

There are four main components for *Guillotine* shown in Figure 1; together they make up the shared portion of the playfield. Foremost is the *Noble Line*: a line of nobles who are awaiting their turn at the guillotine. The *front* of the *Noble Line* refers to the noble adjacent to the guillotine (e.g., $Noble_0$ in Figure 1). The other three playfield components are distinct decks of cards we consider in turn.

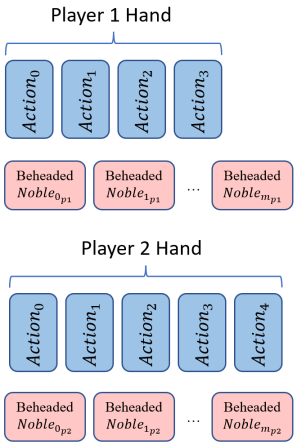


Figure 3: A view of a two player game.

Figure 4: The *Master Spy* noble card.

The *Noble Deck* is a collection of 50 noble cards that are used to populate the *Noble Line*. As shown in Figure 2, a noble card consists of a (a) name, (b) background color, (c) a point value and (d) any special instructions. Table 1 gives more details regarding the color of nobles, how they can be interpreted in the thematic context of the French Revolution, and some examples. As an example of a noble, Figure 4 is named *Master Spy*, has a red background (indicating a member of the military), is worth 4 points, and has the special instructions "After each action card is played, move this card to the end of the line." Points for nobles vary from -1 for grey-bordered nobles and up to 5 for nobles like *Marie Antoinette*. We will analyze these special instructions in more detail in Section 4. Some nobles in the noble deck have no special instructions while some nobles are repeated or omit a specific point value. For example, the *Palace Guard* represents 5 of the 50 nobles and has special instructions related to scoring: "Each Palace Guard is worth a number of points equal to the number of Palace Guards in your score pile (including this one)."

The second deck is a 60-card set of cards called the *Action Deck*. Each individual action card varies according to its text; however, an action either (1) manipulates the *Noble Line*, (2) serves to add (or subtract) points to (from) a player's total, (3) serves to add (or subtract) cards from a target player's hand, or (4) targets another player's already collected nobles. An example of each card type is shown in Table 2. For example, not shown in Table 2, *Forward March* is a card that manipulates the *Noble Line* with the instructions "Move a *Palace Guard* to the front of the line." The last deck in the game playfield

Table 2: An example of each type of action card.

Name	Type	Card Text
Trip	Manipulate the *Noble Line*.	"Move a noble backward exactly 1 place in line. You may play another action card this turn."
Church Support	Modify player point total.	"Put this card in front of you. It is worth +1 point for each Blue noble in your score pile."
Infighting	Modify player hand.	"Choose a player. That player must choose 2 action cards from their hand and discard them."
Clerical Error	Target another player's already collected nobles.	"Choose a player. Collect any noble of your choice from that player's score pile. That player then chooses any other noble from your score pile and collects it."

is a discard pile for action cards that have been used or discarded by players.

3 Gameplay

The goal of *Guillotine* is to earn the most points. Points are earned through a combination of collected nobles and action cards that augment how points are computed.

At the beginning of the game, each player is dealt 5 actions card into their hand; Figure 3 shows Player 1 with 4 action cards in hand while Player 2 has a hand of 5 cards. The game consists of 3 days. Each day the players deal out 12 nobles from the *Noble Deck* to the *Noble Line* where the nobles await their turn to face the guillotine. When the *Noble Line* is empty, the "day ends".

Play consists of each player taking their *turn* in sequence. According to the rules, a turn consists of 3 steps:

1. Play an action card. A player may opt to skip this step.
2. Collect the noble closest to the guillotine in the *Noble Line*. Collected nobles are placed in front of the player face-up for all players to view as shown in Figure 3.
3. Draw an action card (whether or not you played one).

When all 3 days have passed, the game is over and the player's sum their points by observing collected nobles and any applicable action cards.

4 Stack-based evaluation with *Master Spy*.

The rules for *Guillotine* are relatively straightforward and generally serve the gaming audience well by facilitating quick play: players can learn the game on

Figure 5: A *Noble Line* with *Master Spy* at the end.

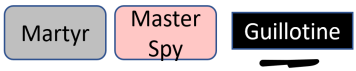


Figure 6: The *Noble Line* after playing the *Trip* card moving the *Martyr*.

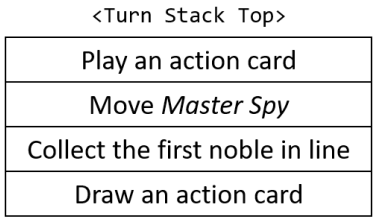


Figure 7: A turn stack if *Master Spy* is in the *Noble Line*.

the fly without having to slog through a complex set of rules. However, the interactions between action cards and the *Master Spy* noble is not addressed well in the rules. This has resulted in many questions and clarification requests by the gaming community [9].

The text of the *Master Spy* card (Figure 4) states that after an action card is played, the *Master Spy* is placed at the end of the *Noble Line*. The fundamental question is: can the *Master Spy* ever be in a position in the Noble Line other than at the end of the line? If not, the *Master Spy* must be the last noble collected. Consider the *Noble Line* consisting of the *Master Spy* and *Martyr* nobles as depicted in Figure 5. The *Martyr* is a grey background character and is worth -1 points thus generally undesirable to collect. To avoid collecting an undesirable noble, the current player plays the action card *Trip* (Table 2) attempting to collect the *Master Spy*. That is, *Trip* allows the player to move the *Martyr* back one position resulting in the *Noble Line* shown in Figure 6; the evaluation of the *Trip* action is now considered complete.

The 3 steps to a player's turn are discrete. When asking students to use a data structure to properly model a player's turn, we want to use the stack shown in Figure 7 that includes an extra step handling *Master Spy* before a player collects a noble. If the *Master Spy* is not in the *Noble Line* then this step is ignored, otherwise, this formal evaluation of steps in a turn reflects the intent of *Guillotine*'s designers. We illuminate a stack-based evaluation of a turn reflecting our scenario involving *Martyr* and *Master Spy*.

1. A turn begins with a turn-stack initialized consistent with Figure 7.
2. Action card *Trip* is played indicating a shift of the *Martyr* back one position taking the *Noble Line* from the state shown in Figure 5 to the state in Figure 6. Pop the stack since the action card has been evaluated.
3. We act on the next step indicated by the top of the stack: move the *Master Spy* to the end of the line (Figure 5). Pop the stack.
4. The player collects noble *Martyr* as the first noble in line.

Table 3: Sample action cards that manipulate the *Noble Line*.

Action Card	Card Instructions
Pushed	Move a noble forward exactly 2 places in line.
Friend of the Queen	Move a noble backward up to 2 places in line.
The Long Walk	Reverse the order of the line.
Extra Cart	Add 3 nobles from the noble deck to the end of the line.
Let Them Eat Cake	If Marie Antoinette is in line, move her to the front of the line.
Bribed Guards	Move the noble at the front of the line to the end of the line.

5. The player draws an action card. Pop the stack.
6. Turn ends as the stack is empty.

While a stack-based evaluation is a bit formal in a game setting, it ensures fairness and repeatability even without the *Master Spy* in the *Noble Line*. This idea is not new. *Magic: The Gathering* [7] instituted a stack-based evaluation of the "spell stack" to ensure clear, fair evaluation of player actions even though it was not defined in the original ruleset.

5 Linear Structures and Considerations

Guillotine has a rating of 1.27 out of 5 complexity rating on BoardGameGeek [3], a reputable community-driven site for all-things board games. A complexity rating this low means *Guillotine* is considered to be a simple game by community standards. While the board game itself may be simple, designing a quality software framework can be a challenge unto itself. Indeed, it may be a reasonable project for a course in software engineering to have groups of students (1) play the game, (2) develop an Object-Oriented software solution, (3) determine interactions among classes, and (4) start conversations about what data structures are most appropriate for game components.

Many of the core components of the game (e.g., *Noble Line*, *Action Deck*, player hands, etc.) can be represented using a linear data structure. We will consider several of these components, but we begin with the *Noble Line* as an example of how the game accesses and manipulates a data structure.

The *Noble Line*. The first time a player engages in *Guillotine* they might conceive of the *Noble Line* as a queue. However, after playing the game, a player might observe the *Noble Line* is one of the most manipulated components. This excessive amount of manipulation has to do with the fact that 39 of the 60 action cards target the line; several examples are provided in Table 3.

It is clear that the *Noble Line* is a linear structure and that players must be able to access both the front of the line (near the guillotine) as well as

the end of the line (the last noble to meet their doom). With respect to the action cards that manipulate the line in Table 3, it is also clear that we cannot treat this structure as a standard queue. In the forthcoming paragraphs, we consider different implementation data structures for the *Noble Line*. We would encourage the reader to ask their students to consider the consequences of each structure as a beneficial pedagogical activity.

It can be argued that an array, with its capability for random access, is a most appropriate structure for the *Noble Line*. It is clear from Table 3 that many of the action cards essentially swap two nobles or remove a noble. However, other cards modify the entire line: *The Long Walk* reverses the line while *Milling in Line* requires a player "Randomly rearrange the first 5 nobles in line." These operations are common in CS1 and CS2 courses, but lack motive; the *Noble Line* provides a clear motivation for these basic operations.

An array is an interesting structure due to its fixed capacity. Each day in *Guillotine* begins with 12 nobles placed in the line; it may seem logical for an implementation to construct an array of capacity 12. However, it is possible for a player (and on rare occasions multiple players) to add more nobles to the line with the *Extra Cart* card (see Table 2). This situation would naturally allow an instructor an avenue to discuss array 'expansion' and the benefits of an object-oriented approach that encapsulates array expansion (e.g., `ArrayList` in Java, `vector` in C++, etc.). This is a particularly good example for students to gain an intuitive understanding of amortization: on many days in *Guillotine* 12 array positions are satisfactory, but on some occasions, more are required.

Pedagogically, we should also consider a linked list structure for the *Noble Line*. An interface for the *Noble Line* clearly requires some form of random access. In Java, we may require a class that `implements` the `List` interface or in C++ overloads the array access operator (`[]`). In the context of a game implementation, this is a strong example of how an instructor can more organically discuss efficiency of random access versus sequential access structures. It is also not necessarily a common operation in linked lists to swap two elements; an interesting and meaningful exercise.

The question of a singly- or doubly-linked list for the *Noble Line* is germane. Since we require access to the front and the back of the *Noble Line*, a doubly-linked list is clearly preferred even though it may be more difficult for students to manipulate. Depending on the goals of the instructor, students could be given the option to choose, but choose wisely.

To address students lacking confidence when seeking external employment, the authors' have been keen to insert questions students might be asked in an interview setting. A common, dreaded interview question asks a candidate to reverse a singly-linked list. Without a motivating context like *The Long Walk* action card in *Guillotine* (see Table 2), students are less likely to take ownership

of the experience. Ideally, students would solve the problem with a recursive linear time algorithm that is also linear in terms of call stack space. However, common solutions include: (a) creating a temporary array containing a copy of the list or (b) an n^2 operation where students construct a new list composed of the last element, second-to-last element, etc. Pedagogically we have found that it is not the solution to this problem that is the most meaningful to student learning. What is most meaningful is the experience of developing a solution, pondering its efficiencies, and then refining. Often, this is a process that iterates several times making the experience a bit daunting, but also incredibly meaningful for students.

Other linear game components. *Guillotine* has other components that require a linear structure: *Noble Deck*, *Action Card Deck*, *Discarded Action Card Pile*, *Player Hand*, and *Player Collected Nobles*.

There are no cards in the game that allow a player to directly manipulate individual cards in either the *Noble Deck* or the *Action Card Deck* other than to draw one card at a time. In order to allow access only to the top card in a deck, the *Noble Deck* and the *Action Card Deck* should be implemented as a stack (although any linear structure with limited element access is reasonable). There is another operation that may arise during play: shuffling. An action card may result in the *Noble Deck* being shuffled. If the *Action Card Deck* is empty, the players can shuffle the discard pile and reset the *Action Card Deck*. An empty *Action Card Deck* is more likely to occur with 4 or more players, but is unlikely to occur in a typical game.

For each of the remaining components (pile of discarded action cards, player hands, and collected nobles), iteration is critical. With the discard pile of action cards, there are a few action cards that allow the user to select an action from the discard pile. Thus, we require iteration, selection, and removal as operations. Similarly for player hands, we iterate through our cards, select one to play, and remove it. In the game of *Guillotine* we iterate through collected noble piles to assess nobles to steal or compute points at the end of the game. This is the perfect opportunity for an instructor to discuss the *Iterator* design pattern [5] along with how to implement these ideas in the target language (e.g., the `Iterable` and `Iterator` interfaces in Java, pointer-based `::iterator` classes in C++, etc.). This discussion might include the non-trivial task of how to safely implement a removal operation with iterators.

6 Some Lessons Learned

Not every project is perfect for all facets of a course; hence, there are a few pedagogical lessons we wish to share. Algorithms associated with linear data structures for *Guillotine* have efficiencies that are either $O(1)$ or $O(n)$ oper-

ations; this is not demonstrative of algorithmic and mathematical diversity. An instructor might introduce more interesting mathematical functions (e.g., $O(n \log_2 n)$ or $O(\log_2 n)$) by integrating sorting and binary searching functionality for the *Noble Deck*, for example.

In teaching with *Guillotine* providing students with a skeleton of the gameplay implementation is often appropriate. This supplement focuses student attention toward implementing and analyzing linear algorithms and data structures and not minutiae of a gameplay engine. Last, as written, *Guillotine* is not a 'course-long' option for CS2 courses that introduce non-linear data structures. In such cases, we suggest implementing player hands as a B-tree where each card type is represented in its own sub-tree. We have provided a few suggestions, but there are many other creative avenues to integrate non-linear data structures or more complex algorithms into *Guillotine*.

7 Related Works

[1, 2] focus on the use of board games for strategy development. The authors report on students implementing a bot and its strategies to engage in automated game-play. The overarching goal is for students to develop and implement strategies that consistently beat a bot player that chooses their moves randomly. While implementing bots is a useful pedagogical approach, our contribution is to describe and analyze one particular game as applicable to a course in linear data structures. While the authors of [1, 2] provide strong motivation for students to implement their software solutions, we provide motivation for common, yet often unmotivated operations like swapping two elements in a list or reversing a linked list.

Lee, et al. [6] focus on enhancing the computational capabilities of elementary school children using an 'unplugged' approach. It can be argued that this work is not related to our work; however, the beauty of board games is that they foster an offline, hands-on approach to computational thinking. As [6] implies, much can be gained from an offline analysis of a game and game strategies. In the case of *Guillotine*, we would ask students to analyze the appropriateness of a particular linear data structure in the face of use cases that can be clearly constructed and analyzed in an 'unplugged' setting. There is also nothing more powerful to algorithm development for reversing a linked list than using manipulatives: for example, cards in the noble line. Another benefit of an unplugged approach is the ability for students to construct clear and precise states for software testing, a powerful and often mundane task.

Drake and Sung [4] provide a survey of games correlated with concepts from CS1 and CS2 courses (e.g., primitives and control structures, one-dimensional arrays, etc.) This work is most similar to our work in that it maps games onto

course content. Contrasting, our work provides a more thorough analysis of one game with a focus on linear data structures.

8 Conclusions

We believe *Guillotine* is a meaningful setting to discuss nuanced choices among linear data structures, including: arrays, singly- and doubly-linked lists, stacks, and queues. These choices can have a dramatic impact on the efficiency and complexity of a software solution and are reminiscent of the considerations students would encounter in a non-academic setting. *Guillotine* represents a viable, all-encompassing project for a course in linear data structures.

References

[1] Ivona Bezáková, James E. Heliotis, and Sean Strout. Board game strategies in introductory computer science. In Tracy Camp, Paul T. Tymann, J. D. Dougherty, and Kris Nagel, editors, *SIGCSE 2013, Denver, CO, USA*, pages 17–22. ACM, 2013.

[2] Ivona Bezáková, James E. Heliotis, and Sean Strout. On the efficacy of board game strategy development as a first-year CS project. In J. D. Dougherty, Kris Nagel, Adrienne Decker, and Kurt Eiselt, editors, *SIGCSE 2014, Atlanta, GA, USA*, pages 283–288. ACM, 2014.

[3] Contributors to BoardGameGeek. Boardgamegeek: Guillotine. `https://boardgamegeek.com/boardgame/116/guillotine`.

[4] Peter Drake and Kelvin Sung. Teaching introductory programming with popular board games. In Thomas J. Cortina, Ellen Lowenfeld Walker, Laurie A. Smith King, and David R. Musicant, editors, *SIGCSE 2011, Dallas, TX, USA*, pages 619–624. ACM, 2011.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[6] Victor R. Lee, Frederick Poole, Jody Clarke-Midura, Mimi Recker, and Melissa Rasmussen. Introducing coding through tabletop board games and their digital instantiations across elementary classrooms and school libraries. In Jian Zhang, Mark Sherriff, Sarah Heckman, Pamela A. Cutter, and Alvaro E. Monge, editors, *SIGCSE 2020, Portland, OR, USA*, pages 787–793. ACM, 2020.

[7] Wizards of the Coast. Magic: The gathering. https://magic.wizards.com/en.

[8] Paul Peterson and Wizards of the Coast. Guillotine. https://en.wikipedia.org/wiki/Guillotine_(game).

[9] Stephen Bowden and et al. Thread discussing master spy. https://boardgamegeek.com/thread/5140/ded-elusive-master-spy.

Designing a Security System Administration Course for Cybersecurity with a Companion Project*

Fei Zuo, Junghwan Rhee, Myungah Park, Gang Qian
Department of Computer Science
University of Central Oklahoma, Edmond, OK 73034
{fzuo, jrhee2, mpark5, gqian}@uco.edu

Abstract

In the past few years, an incident response-oriented cybersecurity program has been constructed at University of Central Oklahoma. As a core course in the newly-established curricula, Secure System Administration focuses on the essential knowledge and skill set for system administration. To enrich students with hands-on experience, we also develop a companion coursework project, named PowerGrader. In this paper, we present the course structure as well as the companion project design. Additionally, we survey the pertinent criterion and curriculum requirements from the widely recognized accreditation units. By this means, we demonstrate the importance of a secure system administration course within the context of cybersecurity education.

1 Introduction

In recent years, we have witnessed an immense shortage of cybersecurity professionals and practitioners across the nation. To fill the gap between the low supply and high demand of the cybersecurity workforce, an increasing number of colleges have launched initiatives to establish new cybersecurity programs or courses in their computer science departments.

In 2021, we initiated a cybersecurity degree program and the certificates for both undergraduate and graduate students at our institution [12]. It is worth

noting that our program features *incident response*, which is an essential task to address security incidents and secure infrastructure. This special theme requires practical strength such as certifiable deep knowledge, hands-on skills, and research-involved cybersecurity activities. To this end, we proposed an array of cybersecurity courses to cover the essential knowledge and skill set for the major tasks of incident response.

Within our cybersecurity curriculum, the Secure System Administration (SSA) course plays a significant role. It aims to introduce the core knowledge and skill set for operating and administering systems securely. In particular, students learn and practice system administration techniques to operate a system with script languages in a command-line interface. In this process, the important concepts, components, and terminologies used in policy, regulation, and risk management for secure system administration are also delivered.

Considering that cybersecurity is a practice-oriented discipline, we always emphasize that students should not only understand how everything works theoretically but also be able to apply the related knowledge to solve real-world problems. For this purpose, we have developed a companion coursework project named PowerGrader, which is a *PowerShell*-based system for automatic management and assessment of programming assignments. We anticipate that students could polish their skills in using a script language by developing applications in a concrete scenario. They are also expected to deeply experience how applications of automation and configuration management work through this practical project.

2 Course Construction

We develop this course with the objective that upon successful completion of this course, students will gain a solid understanding of essential concepts, components, and terminologies involved in system administration policies, controls, and risk management. They will also be able to proficiently administrate computer systems via command-line interfaces and other productivity tools. Also, the capability of designing and implementing the automation of computer operational tasks using script languages is a designated learning outcome.

2.1 Course Structure

Students attending the course are juniors in the CS program as well as students in the software engineering program. They should have finished and passed the Programming I and Programming II courses or equivalent. Table 1 shows the course structure, which is divided into four topics. All of them are commonly suggested in a syllabus concerning security system administration geared towards those aforementioned objectives.

Table 1: Course Structure and Knowledge Units (KUs)

Topics	Knowledge Units (KUs)	Student Activities
T1 (3 weeks)	• Administration basics • Terminals and CLI • Windows essentials • UNIX/Linux essentials	Project: Phase 1 Quiz 1
T2 (3 weeks)	• Regular expression • Editors • Software management • Other system tools	Project: Phase 2 Quiz 2
T3 (5 weeks)	• Script Languages - Bash • Script Languages - PowerShell	Project: Phase 3 Quiz 3
T4 (3 weeks)	• Policies and regulations • Risk analysis and management • Security controls and frameworks • System certifications	Project: Report Final exam

T1: Command-line interface and essential commands

System administration is the field of work in which someone manages one or more systems including software, hardware, servers or workstations. Its main goal is to ensure systems are running efficiently and effectively. It is not surprising that command-line interface (CLI) has become a crucial skill for system administrators. Its programmable characteristic provides great convenience for scheduling automation. Therefore, demonstrating proficiency in the usage of CLI is of necessity for professional management.

T2: System tools and productivity applications

Mastering good tools are a prerequisite to the success of any jobs. For instance, a regular expression (or shortened as regex) is an immensely powerful tool that can be used to improve productivity in routine tasks. Besides this, other system tools such as Advanced Package Tool (APT), Windows management instrumentation (WMI), and editors (e.g., Emacs or Vim) are covered.

T3: Scripting languages and automation

Scripting languages, e.g. *PowerShell* and *Bash*, are commonly utilized in system administration tasks due to the convenience provided by them for manipulating and automating operating system facilities. As an open-sourced and cross-platform system administration and configuration framework from Microsoft, *PowerShell* is widely used to automate routine and repetitive tasks. For example, *PowerShell* is leveraged to automate configuration management in computer networking laboratories [11]. Moreover, current cyber-criminals or hackers usually adopt *PowerShell* as a component of their attack tool-chain. The number of penetration tools that use *PowerShell* increased accordingly at

high speed in recent years [5]. That is why *PowerShell* has been integrated into many cybersecurity-related courses [8].

T4: Policies, controls, and risk management

In an organization, avoiding risks is often difficult due to many reasons such as human fallibility and uncontrollable external factors. Sometimes it is even impossible. How to assess and manage risks is thus an important capability for a qualified system administrator. In this course, essential concepts, components, and terminologies used in system administration policies, controls, and risk management are introduced.

2.2 Comparison with Other Similar Courses

Admittedly, system administration courses are broadly provided by sibling universities. Some are particularly designed for IT programs [9], which do not extensively cover security-related topics. Others only skim the surface of automation based on script languages or mainly focus on Bash [14], although they are included in a cybersecurity program. By contrast, our course design differs in three aspects: the hands-on experience involved companion project, the concentration on *PowerShell*-based administration automation, and the industry-leading, certificate-oriented course structure.

3 Companion Project

The prototype of PowerGrader was inspired by our practical needs in teaching entry-level programming courses [15]. We systematically integrate it into the SSA course as a companion project, so that students can gain a solid understanding of how to make effective use of scripts for automatic system administration. In this section, we present more details about PowerGrader.

3.1 Background and Motivation

Automatic code assessment solutions are a category of widely adopted auxiliary teaching applications. They have been developed to evaluate students' programming assignments for correctness, efficiency, and adherence to the best practices and coding standards. At the very beginning, we initiated our customized code assessment tool PowerGrader in response to the actual demands during our teaching practice. Later, we realized that the development of PowerGrader was suitable to be adapted as a companion project for the SSA course. That was not only because learning how to evaluate the code would be especially beneficial for computer science students, but also because

through this project, they could experience firsthand the significant role of scripting in system administration automation.

We observe that a majority of the existing methods are implemented based on black-box testing, such as CodeAssessor[13] and MOCSIDE[2]. This subcategory of methods are less flexible and unsuitable for introductory-level programming courses, where students are usually required to practise a certain language syntax or programming paradigm, so that a plausibly correct output cannot be simply regarded as the indicator of an acceptable solution.

The "leap year" question can be used as a case study to illustrate this concern. In this assignment, students are expected to implement a program to determine whether a certain year is a leap or common year. As a classical question, this exercise aims at asking students to practice *nested branch statements*. In reality, however, we noticed some students included complicated logical expressions with boolean operators to subconsciously bypass the original requirement, as shown in Figure 1. It was obvious that code assessment based on black-box testing would underperform when handling such a case.

```
if (year%4 != 0 || (year%4 == 0 && year%100 == 0 && year%400 != 0)) {
    cout << "Common␣year" << endl;
}
if ((year%4 == 0 && year%100 !=0 ) ||
    (year%4 == 0 && year%100 == 0 && year%400 == 0)) {
    cout << "Leap␣year" << endl;
}
```

Figure 1: Solving the "leap year" problem without *nested branch statements*.

On the other hand, approaches based on static analysis such as abstract syntax tree or symbolic execution would be more suitable for such intricate tasks, e.g., vulnerabilities discovery and code quality assessment [7, 10]. However, these methods would usually introduce high a false-positives rate that preclude using them for grading purposes [6]. Thus, developing an auto-grader for entry-level programming courses based on their own characteristics appeared to be a more appropriate approach.

3.2 Project Breakdown

The core module of PowerGrader is a hybrid code assessor consisting of a black-box tester and a lexical analyzer. Apart from that, the proposed system can periodically collect the submissions from a file server, and automatically manage them. When the assignments are analyzed, assessment reports and compiling information including error messages are recorded into a log. Based

on these functionalities, we decompose the development of PowerGrader into three phases, as shown in Table 1.

Phase 1: Assignment collection and pre-processing

The assignment collection module is deployed based on a file server where students can submit their work via FTP. A time stamp is automatically attached to each submission. Meanwhile, user credential and access permission are defined accordingly as a security mechanism. In particular, we assume every submission is archived using a ZIP file, and require the file name be in the format as "*FirstName_ LastName_ AssignmentNumber.zip*". Consequently, by parsing the name and meta data of such a ZIP file, we can initialize an assessment report including necessary information such as student name, submission time, etc. After decompressing every ZIP file, the source code will be compiled by invoking a third-party compiler. Any information like error or warning messages during this procedure will be logged.

All of these system administration tasks are required to be conducted by leveraging CLI-based commands. Hence, this phase is to align with the KUs covered by T1, as shown in Table 1.

Phase 2: Regex based lexical analyzer

The lexical analyzer is implemented in a lightweight manner, where the given regular expressions parse the source code to determine whether the student's implementation can fulfill the desired constraints. For example, the regex pattern shown in Figure 2 can be used to verify *nested branch statements*. Considering that PowerGrader is specially designed to handle code submitted as coursework in a programming class, this lightweight approach is not only feasible but also flexible.

```
if\s*\([\s\S]*\)\s*\{[\s\S]*
    if\s*\([\s\S]*\)\s*\{[\s\S]*\}\s*
    else\s*\{[\s\S]*\}\s*\}\s*
else\s*\{[\s\S]*\}
```

Figure 2: A regex pattern for *nested branch statements*.

Upon finishing the lexical analyzer, students can sufficiently polish their skills of utilizing productivity tools like regular expressions and editors. As a result, this phase covers the KUs related to T2 in Table 1.

Phase 3: Black-box test and automation

In the black-box test, expected outputs are literally compared with the counterparts from running a student's code against a test set. In addition to this, all aforementioned functionalities will be integrated into the final product

Table 2: Agreement to the Course Evaluation Questions

Course evaluation question	Average score	Standard error
This course challenged me to think from different angles	4.50/5.00	0.183
I acquired multiple essential system administration skills	4.81/5.00	0.136
The project is well designed and practiced my skills	4.75/5.00	0.112
The techniques delivered by this course are very useful in practice	4.63/5.00	0.155

by making effective use of script languages. According to evaluation results obtained from the hybrid code assessment, a report will be generated as a reference for each submission.

To achieve automation, script language programming is highlighted in this phase, which covers the KUs related to T3 in Table 1.

Project report

On completion of all the phases above, a closing report will be required to conclude the whole project, where the newly developed application will be considered as a case study. Students apply the knowledge covered by T4 as shown in Table 1 to analyze this coding assignment management and assessment system. Through this exercise, they can sharpen their understanding of security policies, controls, and risk management-related concepts in practice.

3.3 Evaluation

Engaging in this hands-on project can make students deeply experience how an automatic system administration and configuration application works in a concrete scenario. Plus, a majority of knowledge units required in the course are covered by this comprehensive project.

To evaluate the proposed course including the companion project, we collected student responses to the course evaluation at the end of 2022. Based on 16 valid responses, Table 2 summarizes their degree of agreement to the questions, which demonstrates a positive feedback.

In addition, we compared students' performances according to their exam scores in two consecutive academic years. The distribution of scores is shown in Figure 3. The average score was promoted from 83 in the year 2021 to 88 in 2022. Hence, a notable performance improvement has been observed after the course project's implementation in 2022.

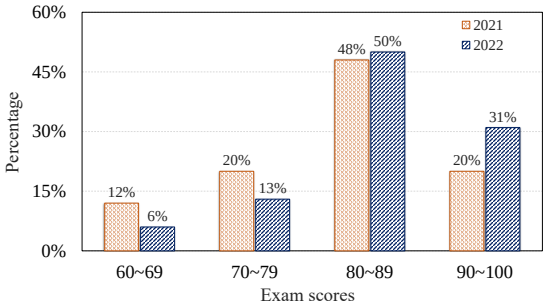


Figure 3: Comparison of exam score distributions in different years

4 Survey on Reputable Accreditation Criteria

4.1 NCAE-C Program

The National Centers of Academic Excellence in Cybersecurity program, managed by National Security Agency (NSA), accredits universities based on their ability to meet rigorous academic criteria and offer top-notch cybersecurity education and training to students [1]. The program has established standards for cybersecurity curriculum and academic excellence towards two specialization tracks, which are Cyber Defense (CAE-CD) and Cyber Operations (CAE-CO). It is noteworthy that the CAE-CD designates five technical core knowledge units (KUs) and another five non-technical core KUs as the curriculum requirements. The proposed SSA course covers two non-technical core KUs, i.e., "security risk analysis", and "policy, legal, ethics, and compliance", in addition to one technical core KU, i.e., "basic scripting and programming".

4.2 ABET Accreditation for Cybersecurity

ABET is a well recognized organization that accredits university programs in natural science, computing, engineering and engineering technology. In particular, the Computing Accreditation Commission of ABET has developed the accreditation criteria for cybersecurity programs [3]. The criteria of ABET explicitly state that curriculum requirements do not prescribe specific courses. Instead, the program needs to involve eight fundamental topics in total, and specifies what course(s) each topic is covered. Our SSA course provides coverage on multiple cybersecurity topics as required by ABET, including data security, human security, organizational security, and societal security. These required topics overlap with T4 of our course structure. Our investigation shows that other ABET accredited cybersecurity programs also consider the

SSA course as a necessity [14].

4.3 CompTIA Certification

To meet the needs of the industry, we also refer to criteria and credentials expected in the job market as the curricular guidance when creating our cybersecurity program. In particular, our department became a partner of CompTIA [4], which is a well-known trade association that issues professional certifications for the cybersecurity and IT industry. Our curriculum is closely aligned with CompTIA certificates. Especially, the proposed SSA course covers the majority of the topics related to the CompTIA Linux+ certificate and some topics for the CompTIA Security+ certificate.

5 Conclusion

As a core course in many cybersecurity-related curricula, secure system administration focuses on not only the essential concepts used in security policies, controls, and risk management, but also the necessary techniques that are used for automating system administration and configuration. Our SSA course has all these suggested topics combined in the development of the PowerGrader course project to enrich students with hands-on experience in automated system administration and further improve their learning outcomes. This paper also surveys the widely recognized accreditation criteria to demonstrate the significance of the SSA course in a cybersecurity program. More importantly, this work shares our experience and insights into teaching SSA and provides an example of its curriculum construction.

References

[1] National Security Agency. National centers of academic excellence in cybersecurity. `https://www.nsa.gov/Academics/Centers-of-Academic-Excellence/`. Accessed: 2023-05-01.

[2] Max Barlow, Ibraheem Cazalas, Chase Robinson, and Jonathan Cazalas. MOCSIDE: an open-source and scalable online ide and auto-grader for introductory programming courses. *Journal of Computing Sciences in Colleges*, 37(5):11–20, 2021.

[3] ABET Computing Accreditation Commission. Criteria for accrediting computing programs, 2022. `https://www.abet.org/wp-content/uploads/2022/03/2022-23-CAC-Criteria.pdf`. Accessed: 2023-05-01.

[4] CompTIA. Computing technology industry association (CompTIA). `https://www.comptia.org/`. Accessed: 2023-04-01.

[5] Danny Hendler, Shay Kels, and Amir Rubin. Detecting malicious powershell commands using deep neural networks. In *ASIACCS*, 2018.

[6] Karen H Jin and Michel Charpentier. Automatic programming assignment assessment beyond black-box testing. *Journal of Computing Sciences in Colleges*, 35(8):116–125, 2020.

[7] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Code quality issues in student programs. In *ITiCSE*, 2017.

[8] Qiang Liu, Wentao Zhao, Minghui Wu, and Chengzhang Zhu. Web security education in a multidisciplinary learning context. In *the ACM Turing Celebration Conference*, 2020.

[9] Mohamed Lotfy and Christian Fredrickson. The structure and delivery of an advanced systems administration it course. *Journal of Computing Sciences in Colleges*, 38(2):33–44, 2022.

[10] Lannan Luo, Qiang Zeng, Bokai Yang, Fei Zuo, and Junzhe Wang. Westworld: Fuzzing-assisted remote dynamic symbolic execution of smart apps on iot cloud platforms. In *ACSAC*, 2021.

[11] Neville Palmer, Warren Earle, and Jomo Batola. Automating the configuration management and assessment of practical outcomes in computer networking laboratories. In *the Science and Information Conference: Intelligent Computing*, 2019.

[12] Junghwan Rhee, Myungah Park, Fei Zuo, Shuai Zhang, Gang Qian, Goutam Mylavarapu, Hong Sung, and Thomas Turner. Developing incident response-focused cybersecurity undergraduate curricula. *Journal of Computing Sciences in Colleges*, 38(7):65–74, 2023.

[13] Brad Vander Zanden and Michael W Berry. Improving automatic code assessment. *Journal of Computing Sciences in Colleges*, 29(2):162–168, 2013.

[14] Xiaodong Yue, Belinda Copus, and Hyungbae Park. How to secure ABET accreditation for a cybersecurity program: a case study. *Journal of Computing Sciences in Colleges*, 37(6):15–24, 2022.

[15] Fei Zuo, Junghwan Rhee, Myungah Park, and Gang Qian. PowerGrader: Automating code assessment based on PowerShell for programming courses. In *the 21st IEEE/ACIS International Conference on Software Engineering, Management and Applications*, 2023.

A Social Good Challenge for Teaching Undergraduate Affective Computing *

Gloria Washington[1], Marion Mejias[2]
[1]Electrical Engineering and Computer Science
Howard University
Washington, DC 20059
gloria.washington@howard.edu
[2]Software and Information Systems
University of North Carolina Charlotte
Charlotte, NC 28262
mmejias@charlotte.edu

Abstract

This paper describes how a social good innovation challenge was used to teach 27 undergraduate and 6 graduate students affective computing techniques. The innovation challenge addressed the UN Sustainable Development Goals (SDGs) and allowed students to create technology solutions to solve problems related to clean water, poverty, and hunger, protecting the planet, human prosperity, and inclusive societies. Course activities were taught using project-based learning and required students to understand their potential users through motivational design thinking techniques. Emotional theory based on emotional artificial intelligence techniques were also taught. Students filled out course evaluations and provided observations on their experience in the course. Student feedback primarily related to motivation to win the design challenge and ability to connect theories taught in class with affective techniques. Challenges related to connecting existing emotional AI software libraries with web-based prototypes. Future iterations of the course will allow more time for integration of existing tools with prototype software.

1 Introduction

University computing students, particularly Black and Brown computing students, often talk about creating software that helps their community and ultimately contributes to the greater good of society. Additionally, most university students across the United States are becoming more aware of how technology may negatively impact already marginalized or underrepresented groups. Unfortunately, computing curriculum taught at most universities do not give students the opportunity to build technology for social good or reflect over how their technology may create a better tomorrow. The United Nation's Sustainable Development Goals are a "shared blueprint for peace and prosperity for people and the planet" [1] that may be leveraged by computing faculty in existing project-based courses to give students opportunities to create technology for social good. These seventeen goals include ending poverty, while spurring economic growth, improving health and education, and reducing racial inequality. Giving students parameters to design and create technology surrounding the UN sustainable development goals can help computing courses teach students how to create meaningful technology around these topics.

Teaching affective computing theory and techniques to undergraduate students is difficult because most concepts are beyond the average undergraduate computing curriculum. Prerequisite knowledge like image processing techniques, machine learning models, and computer vision are usually too difficult to teach in one semester are taught in their own separate courses. Additionally, most affective computing courses are graduate courses that rely on teaching the artificial intelligence and psychological theory that forms the basis for the field. Undergraduate students are becoming more aware of emotionally intelligent tools are interested in creating software and tools that not only help the world but are also emotionally aware. Tools like facial expression recognition and applications use human physiological information to guess mood, stress, and anxiety are becoming commonplace. Students want to work on interesting problems that they dream-up.

In this paper, we explore if teaching applied affective computing techniques using a social innovation challenge to motivate students would improve student engagement in the course. To determine this, analysis was performed over end-of-the-semester student questionnaires and grades on the final project.

The rest of this paper is organized by describing the common topics and theories taught in most affective computing courses. Next, briefly described is the prior use of social innovation challenges employed in computing courses listing the challenges and weaknesses of these approaches to improve student performance. Then, the methodology is described and how this innovation challenge requirements was used to tailor semester deliverables for the course. Finally, we describe the results of study and study conclusions.

2 Background

2.1 Emotion vs. Affect

An emotion is a feeling that occurs in humans that results sometimes in physical or psychological changes that influence behavior and thought [5]. Affect is the underlying experience of a feeling, emotion, or mood[5]. It is the reaction of the human, be it physical through movement of body muscles or parts or the physiological reaction that occurs in the human autonomic nervous system[5]. Often these words are intertwined when software developers are speaking about the emotions of the user in designing or evaluating experiences for human-computer interactions.

Assessing emotion by software and user experience experts is usually done through self-report in a survey. However, assessing the affect of the user involves both the self-report of the emotion and noticing a change in the human autonomic system (if any). Common changes to the human body are increased heart rate for excitement, increased skin temperature for stress, anger, and nervousness. In the next sections we describe the psychological theories of emotion that pertain to how emotion is displayed outwardly to other humans during interactions.

2.2 Basic Theory of Emotions

The Basic Theory of Emotions states that humans experience six basic emotions: happiness, sadness, surprise, anger, disgust, fear [3]. Through this theory, human emotion is binary as it either occurs or it doesn't occur. The premise is that these six emotions all have a signal or input that causes the emotion, a physiological response, and an action or response that immediately follows the behavior. Sometimes this following behavior is masked or hidden by the human. The emotion also has a duration and many emotions can lead to a change in mood by a human. Example getting a ticket while driving to work and getting written-up for lateness contributes to a disgusted mood. Facial expressions are one obvious way in which the six basic emotions are displayed.

Undergraduate students are very aware that more than six emotions can occur within a human, however, they should begin to understand the complexity surrounding emotion, mood, and intensities of emotions experienced by humans. This is difficult, as sometimes different cultures display emotions uniquely.

2.3 Appraisal Theory of Emotions

Appraisal Theory of Emotions says that human emotions are experienced and reacted to differently according to appraisals or assessments directly relating

to a human's values or beliefs [9]. Additionally, in every situation a person finds themselves in, they weigh these values against their expectations and try to assess how hard or easy it will be to accomplish the goal in the particular situation. When faced with goal-blocking events that were unexpectedly encountered, humans often experience more intense emotional reactions[5]. An example of this is a person is given a ticket for driving below the speed limit that causes him/her to receive a demotion at work for lateness. The unexpectedness of the ticket may contribute to a more intense reaction of disgust experienced by the driver because they did not expect to receive a ticket for slow driving which is often associated with safe driving. The affect towards the future driving experience may be hesitation, nervousness, disgust, with a change in elevated heart rate and respiration.

Wearables and their connection to the study of human emotions and affect can be introduced during appraisal theory lessons since physiological data, context surrounding the task, and participant feedback can communicate the basic constructs of appraisal theory. Wearable sensors like fitness trackers gather physiological data like heart rate, skin temperature, blood oxygen levels, and sometimes blood pressure.

2.3.1 Other Emotional Theories of Emotions

The OCC theory of emotions examines the cognitive structure of emotions so that they can be modeled by artificial intelligence for emotion generation or emotion synthesis [6]. The OCC theory of emotions builds upon Basic Emotion Theory, Appraisal Theory, and breaks down emotions so that AI can began to identify what causes humans to experience emotions, what makes emotions vary in intensity, how emotions are similar to each other, and how behavior can be predicted from learning about the experience preceding the emotion.

OCC Theory can be confusing to undergraduate students as its difficult to understand how constructs like "consequences of events, actions of agents, and aspects of objects translate into code. Often the terms agent is associated with an AI agent and objects are associated with object-oriented programming. Additionally, appealing-ness, desirability, and praiseworthiness are difficult to describe without concrete examples. Undergraduate need hands-on ways to see this theory in action. Activities or lessons about OCC theory can include students finding technology based on the theory and how the behavior of the technology reacts due to user interactions and input.

Compound emotional theories found in Schachter et al [8, 7] are too complex to delve into within a normal one-semester undergraduate affective computing course because the examples surrounding their application are usually artificial humans, emotional agents, and deep learning techniques.

2.3.2 Social Innovation Challenges

Innovation challenges like the CMD-IT Innovation Challenge, the Verizon & CGI U Social Innovation Challenge, the Clinton Foundation and Verizon Social Innovation Challenge, and the Law School Admission Council's Social Justice Innovation Challenge are all geared to creating new technological ideas to solve racism, prejudice, reducing climate change and poverty, and decreasing racial bias and in in the United States. Challenges have been used in prior studies in computing and business to teach problem-solving, project management, and giving students the ability to confront real-world problems[4].

3 Methodology

Twenty-six undergraduate and six graduate students worked on projects to address global social ills as outlined in the UN 2030 Sustainable Development Goals. Students went through motivational design thinking activities to quickly create "how might we" statements to address a UN sustainable development goal. Then students quickly thought of technology solutions that addressed the "how might we" statements. The technology that had the most votes for that sustainable goal was developed for the team-based project. There were seven teams, and each project was given the guidelines that the overall motivation behind the application must also amplify or reduce a particular emotion. Additionally, each group must pick an emotional theory that is the underlying basis for how the students assumed human emotions would be experienced or expressed by humans using the prototype.

Graduate students in the course were allowed to pick their own groups even though they usually were the persons with the most programming experience in the course. There was no policy that graduate students could not create a group with only graduate students either. The six graduate students in the course were intermingled throughout the groups and chose their group membership strictly based on the topic they wanted to address in the innovation challenge. Additionally, the graduate student could also mentor the undergrads through the software engineering process.

3.1 Social Innovation Challenge

The challenge used in this study was sponsored by an organization that works directly with studying how technology can help Black and Brown persons, Indigenous populations, and persons with disabilities. The primary goal of the CMD-IT challenge was to tear down structural and institutional barriers so all humans can build a more positive society. All submissions required the teams pitch their idea in 5-minutes or less. Each submission was voted on

by peers or outside persons visiting the challenge site. Judging criteria was devoted to creativity, innovation, usefulness, impact, scalability, sustainability, and design. Prizes included $500, $300, and $100 for first, second, and third place submissions. Also, teams could win a t-shirt for fourth place.

3.2 Human Subjects Protection

All students were required to complete Social and Behavioral Subjects Research from Citi.org. Exempt approval was given for the course. They were also told that they could not share any pics or videos of students they test using their software.

3.3 Applied Affective Computing Techniques

Applied affective computing techniques were used to teach the concepts in the course. Students were taught about how emotions are exhibited by the human autonomic system and through human physical behavior. They also had to concretely tie an emotional theory (Basic Emotional Theory or Appraisal Theory, etc.) to tasks that users would perform using their prototype solution. Additionally, students were taught how to measure these physiological changes using wearable devices that tracked heart rate, skin temperature, and respiration. Verification of the human emotional response was taught as well through questionnaires and surveys. These are described in the next sections.

3.3.1 Facial Expression Software and Wearables

Facial expression software from companies like Affectiva, Amazon Rekognition, Microsoft Azure Cognition, MorphCast, etc. can be used to identify the six basic emotions. Students within the course are shown the functions to include in their tools for using facial expression systems. A web-browser based implementation of a facial expression tool was the preferred method shown by the instructor and usable without much coding experience.

Wearable devices like Fitbit can help students detect stress, anxiety, physical activity, and mood. Getting the data off these devices are as simple as plugging them into a laptop or downloading it from a cloud-based account. Students were taught how to interpret the output of the fitness trackers noting timestamps, mood indicators, stress levels, and other physical information.

3.3.2 Non-technical Affective Computing Techniques

Students were taught about the PANAS, SUS, and high-level data analysis for reporting. The Positive and Negative Affect Schedule (PANAS) was used to verify the emotion of the users. The Positive and Negative Affect Schedule is a

quick-and-dirty way for students to capture the emotions of the users in their study through self-report. Students in the course were taught how to properly convert the self-report measures in the PANAS into readable results using the short form of the PANAS [10]. Students were also taught how to compare the emotions identified in the PANAS to both facial expression output and wearable output. Context of the user's task was also noted by a student observer.

System Usability Scale [2] or SUS, is a quick way to measure user satisfaction as defined by the reliability and learnability of an application. Students were taught how to normalize their scores and interpret SUS output for analysis. A score of around 68 was encouraged. Note: most of the undergraduates did not have statistics or data science experience prior to enrolling. Therefore, analysis of results was limited to descriptive statistics like mean, median, mode, and data visualizations from spreadsheets. Undergraduate that already took Probability and Statistics I at the University were encouraged to perform hypothesis testing using the data they gathered.

Pitching of ideas was a requirement by the innovation challenge. Therefore, students were taught how to pitch their ideas according to four main questions: What's the problem according to the UN SDGs?, Why should you care (in terms of the global problem)?, What's your proposed solution?, and How is it different?. Additionally, students introduced themselves in their pitching video before giving a short demo of the solution. These videos were uploaded along with a link to their prototype solutions to the innovation challenge website.

3.3.3 Technical Non-affective Computing Techniques

Rapid prototyping tools like proto.io, Adobe XD, Thunkable were used by students to quickly create their solutions with little coding. Proto.io and other tools that allow for outside calling of facial expression and audio emotional APIs were easier to leverage in the student projects. Most student groups used Thunkable and Proto.io rapid prototyping tools.

4 Results

Project prototypes were built addressing climate change shown in Figure 1, economic growth and human prosperity, inclusive societies, and addressing chronic disease for improved health and wellbeing. Seven project videos and project prototypes were uploaded to the innovation challenge website. Students could ask their friends and family to vote for their solutions in the challenge, however the final winner for the contest was not determined only by outside votes. A team of external judges evaluated the pitch videos and the materials uploaded for each project. The 1st place submission was submitted from the affective computing course described in this paper. The topic for the winning

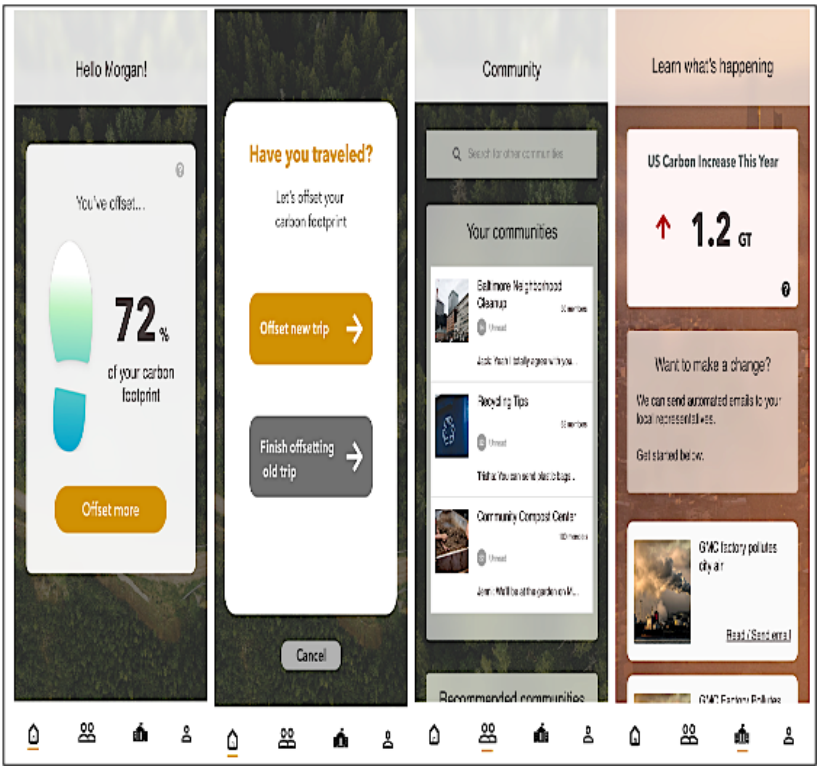


Figure 1: Team project from affective computing course related to climate change.

solution was a support community for persons with chronic disease to share their stories of microaggressions and bias that they experienced in healthcare settings. The project implemented facial expressions API to help understand the emotions of the persons that uploaded their stories. Encouraging words could be sent to those users. Also, the 2nd place submission came from this affective computing course and the topic related to persons with disabilities and how they feel left out of class discussions. The emotional component used emoji, likes, and a mental health break for the disabled students.

The average grade earned in the course was 95% with 29 students receiving and A, two students receiving a C, and one student receiving a B. Due the pandemic some students did not turn in some assignments in the course, however the final project was turned in by all students as it included the results of testing their solutions from the Social Innovations Challenge. Thirty students

received an A on the team project, while one person received a C due to not helping with the demo and usability testing of the final project.

5 Discussion

Students were taught that often mood impacts the experience of users and were also told to use a pre- and post-test PANAS to determine the emotions of the users before and after using their prototypes. Some students forgot this and their results were flawed. Only one student group, the group of two undergrads, used a wearable to verify the emotions of their users. The other groups used the facial expressions software output to verify the emotion. Many of the student groups gathered data that showed that the users enjoyed their tool, but the emotion of the users did not correspond positively to the enjoyable experience of the user as the SUS was higher than 68 showed the learnability of the tool was not well received by user. Only one student noted that the mood of the user was negative coming into the experience with disgust, fear, apprehension, and stress as they had just finished a final before participating in the usability study of the prototype. Many students did not know how to interpret the PANAS so that they could improve the experience of the tool. One reason could be that whimsy, laughter, and attributing an interaction in the real-world with an interaction in the tool was not explained in this class.

Students did not readily connect their tool creation with an emotion to induce, increase or decrease in the user. An example is the climate change group asked users to input daily activities to determine how they are reducing their carbon footprint. However, a metaphor connection to the real-world that included human footprints and their devastation to the rainforests or their local environment would have amplified the need for climate reduction strategies and created more of an emotional reaction in the users. Students in a future iteration of the class will have a topic discussion on use of whimsy, laughter, and metaphors with the real-world to encourage user behavior.

Often students used facial expressions when it was not appropriate for the subject matter or necessarily useful for the tool. Perhaps this is due to familiarity with facial expressions tools.

6 Conclusion

User emotion provides needed context and can be leveraged to evaluate design decisions by software developers. Innovation challenges centered around social good topics can be leveraged to excite undergraduate students to design software for good. Teaching about leveraging emotional AI tools for existing technology innovations is difficult, but motivating students through an inno-

vation challenge and the thought of winning prize money can help to excite students about learning affective computing.

References

[1] Frank Biermann, Norichika Kanie, and Rakhyun E Kim. Global governance by goal-setting: the novel approach of the un sustainable development goals. *Current Opinion in Environmental Sustainability*, 26:26–31, 2017.

[2] John Brooke. System usability scale (sus): a quick-and-dirty method of system evaluation user information. *Reading, UK: Digital Equipment Co Ltd*, 43:1–7, 1986.

[3] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992.

[4] Jan Kietzmann and Herbert H Tsang. Minding the gap: Bridging computing science and business studies with an interdisciplinary innovation challenge. In *Proceedings of the 15th Western Canadian Conference on Computing Education*, pages 1–5, 2010.

[5] Randy J Larsen and Ed Diener. Affect intensity as an individual difference characteristic: A review. *Journal of Research in personality*, 21(1):1–39, 1987.

[6] Andrew Ortony, Gerald L Clore, and Allan Collins. *The cognitive structure of emotions*. Cambridge university press, 2022.

[7] Rainer Reisenzein. The schachter theory of emotion: two decades later. *Psychological bulletin*, 94(2):239, 1983.

[8] Daniel L Schacter and Peter Graf. Effects of elaborative processing on implicit and explicit memory for new associations. *Journal of experimental psychology: learning, memory, and cognition*, 12(3):432, 1986.

[9] Klaus R Scherer. Appraisal theory. 1999.

[10] David Watson, Lee Anna Clark, and Auke Tellegen. Development and validation of brief measures of positive and negative affect: the panas scales. *Journal of personality and social psychology*, 54(6):1063, 1988.

Checkpoint Classifier for CNN Image Classification*

Jackson H. Paul[1] *and Andy D. Digh*[2]
[1]*Electrical and Computer Engineering Department*
Mercer University
Macon, GA 31207
jackson.h.paul@live.mercer.edu
[2]*Computer Science Department*
Mercer University
Macon, GA 31207
digh_ad@mercer.edu

Abstract

Given a large convolutional neural network (CNN) with hundreds of layers, when can the input data be correctly classified? How many layers does each image require? We propose an architecture with a mid-network classifier to classify certain images at earlier points in the model. When the network is very confident about an image, having high activations, then that individual image will be classified early. The number of computations and the average number of convolutions will be reduced if certain images can be classified earlier. In addition, the mid-network classification task is more difficult because fewer features have been extracted at earlier points in the network. Thus, the output and mid-network classifier will work together to correctly classify each image as fast as possible while preserving the accuracy. This proposed method has been implemented into well known computer vision architectures, like ResNet and GoogLeNet Inception. We have achieved large runtime improvements while limiting the accuracy degradation.

1 Introduction

Convolutional neural networks (CNN) have vast amounts of parameters in order to learn hundreds of feature maps for each layer [19, 23]. Although the number of parameters can be large, convolutional models can perform extremely well and have progressed from accurate handwritten digit classification to real-time, efficient object detection [14, 7]. This paper [7] shows the general trend of how the size of the model increases as the accuracy increases. The winning submissions of the famous ImageNet Large Scale Visual Recognition Challenge have increased in size as well, producing much larger networks each year to achieve better performance [18, 3].

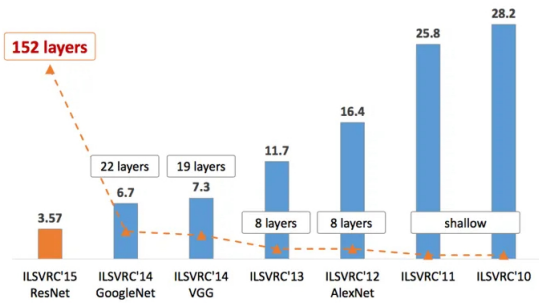


Figure 1: Number of layers (dotted line) and test error (bars) for each winning model of ImageNet Large Scale Visual Recognition Challenge from 2010 to 2015 [3].

The motivation for this paper comes from numerical analysis; we have iterative techniques to solve all types of complex mathematical operations, including integration and differential equations [2]. Iterative techniques are great for modern computers because they can iterate forever. When using these numerical algorithms, thresholds are used to determine when the answer is 'good enough'. Once the answer is below a certain threshold of precision, the algorithm is finished and the answer is returned. Using these concepts, the same question was asked in the context of CNNs; "When is the input data processed enough to be classified correctly?"

We propose a CNN architecture with a checkpoint to attempt to classify the input data earlier in the model. If the input data can be classified, then the model will stop, and output the results. If not, the model will continue and use the remaining layers as intended. The network will naturally be more and less confident about certain images, so only the most confident images will be classified early. In addition, fewer features have been extracted at earlier points in the network, so the checkpoint classifier should perform worse than

the overall output classifier. Thus, the output classifier and the checkpoint classifier will work together to classify images faster while retaining accuracy. Theoretically, if certain inputs could be classified earlier in the network, then the total average number of convolutional layers would be reduced, and the runtime would be improved while minimizing accuracy degradation.

2 Related Works With CNNs

Starting with LeNet, with some exceptions, CNNs usually have a basic, predictable structure [4]. First, a set of convolutional layers to extract features, then fully connected layers to classify the input based on the features, in that specific order [23]. This proposed architecture would differ by having a variable number of layers and a second fully connected classifier network that interrupts the series of convolutional layers.

In [5], they describe using two independent classifiers to have redundancy in case one classifier is incorrect. They both perform the classification after the convolutional layers using the same data but utilize two different approaches to protect against an incorrect classification method. In addition, one of the most famous papers, "Going Deeper with Convolutions," describes the GoogLeNet Inception network [22]. This network uses two mid-network *auxiliary classifiers* to protect against vanishing or exploding gradients and provide some regularization during training [12]. The network has a traditional set of fully connected layers for classification but aggregates all output losses to perform backpropagation. This proposed architecture has an additional set of fully connected layers but they perform a separate classification that does not combine or contribute to the original output classifier.

3 Methods

Overview — First, the general context and overview will be discussed and then each step will be described. Python and PyTorch were used with the built-in CIFAR10 and CIFAR100 datasets. These image classification datasets have ten and 100 output classes, respectively, and have 50,000 training images and 10,000 test images [10]. For the following sections, the base model refers to the unchanged, published architecture, whereas the custom model is the published base model with the extra mid-network classification layer added.

Published Architectures — The first step is training the base model on the given dataset and saving that model to be used for testing. PyTorch has several trained architectures built-in, but saves models as a dictionary. So, the

models need to be trained from scratch to add new layers after training and for layers to be added to specific places in the sequential order.

Table 1: Base model testset accuracies with both datasets, CIFAR10/100.

Model	CIFAR10	CIFAR100
AlexNet	82.47%	48.24%
VGG-16	92.28%	69.34%
ResNet34	82.25%	52.47%
ResNet50	84.46%	56.78%
InceptionV1	86.00%	61.62%

Looking at Table 1, the accuracy of the published networks were not particularly optimized using learning rate, batch size, or other training hyperparameters. Firstly, the graphics card used had memory limitations, so the larger models needed a small batch size to fit on the card. This prevented the models from generalizing the classification and resulted in overfitting and high losses on the testset. In addition, the parameters learned during the base model training are used for both models because the base model layers are unchanged; only a new layer with the mid-network classification is added to the custom model. Therefore, any accuracy and runtime changes are relative to the base model and that difference is the important metric. The base models could have been trained better, but all base model parameters are shared with the custom model so the results will still show the validity of this proposed method.

Mid-network Classification Model — This model was used as a separate classifier and trained using mid-network activations extracted from the implementation layer. After the base model training finished, a custom layer was added to the base model to write the mid-network activations and correct labels to a binary file. Multiple layers were tested for each base model and dataset so activations from each specific layer were collected and stored for training. The layers were chosen to be roughly 40% to 60% of the overall network such that some features have been extracted for a successful classification, but enough layers remain to improve the runtime.

This model was trained on all 50,000 trainset inputs and used a $90\% - 5\% - 5\%$ train, development, and testset split, respectively. The network used two fully connected layers, sizes 2,048 and 1,024, and each layer used a dropout of 70% and batch normalization [21, 20, 8, 12]. The second layer outputs ten or 100 classes depending on which dataset was being used [11]. For the input, this architecture was used for all base models, so the feature map sizes will change

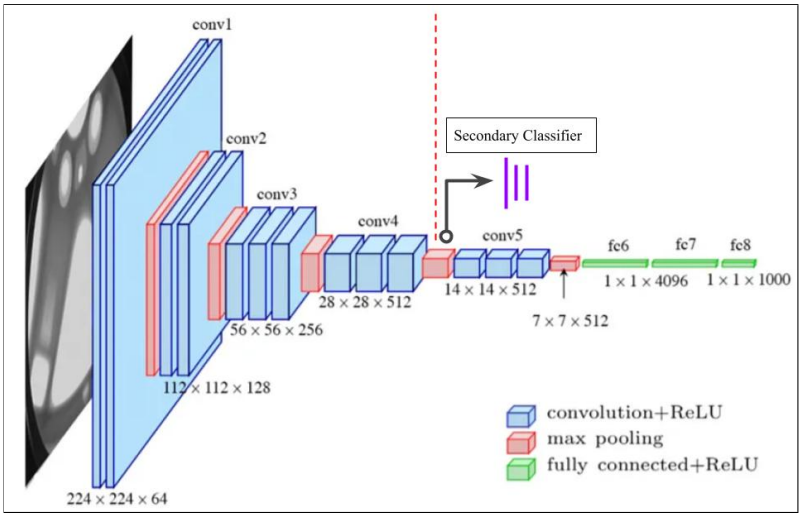


Figure 2: Checkpoint classifier implementation of Mid-network classification model into the VGG-16 base architecture [13]. For the functions, the activation function was Rectified linear unit (ReLU); Adam optimizer with a weight decay of $4 * 10^{-5}$; the loss function was Cross Entropy Loss [15, 1, 9]. Using the Adam optimizer helped the model converge faster than traditional stochastic gradient descent [17]. The final output layer did not use a softmax function because the unrestricted activation values were used to find the threshold value [16].

accordingly. Thus, average pooling was used to downsize the feature maps and choose a pooling size to produce about 3,000 to 4,000 values after flattening the feature maps. Using about 3,000 to 4,000 input values was enough information to classify the inputs but still retain a fairly small overall network. Although, there were one or two cases that used 4,608 as the input size because due to the number of feature maps, ∼4,600 was the closest to the desired range. Although marginally better classification could be achieved with individual optimization for each base model and dataset combination, this architecture performed well and we decided to keep it constant for more consistent results.

Parameter Search — A maximum value threshold is used to determine if the mid-network classification will be kept or not. The mid-network classification model will attempt to classify every image, but only activations above the max threshold will be classified early. In addition to the max threshold, the number of inputs let in and processed early was also optimized as a threshold, this percentage is the let-in threshold. The best max value threshold is found for each let-in threshold for the given layer and base model. At a given let-in

threshold, the “best" max value was found by iterating through values, at a specified step, and finding the percentage of images above the max threshold and the percentage correct and above the max threshold. Those values were divided to find the relative accuracy of inputs above the let-in threshold.

Although, for a max threshold to be optimized for each let-in threshold, the percentage of images above the max threshold is constrained between the let-in threshold and the let-in threshold plus 10%. Thus, as less images are let in, the mid-network classification model can be more selective so relative accuracy increases. The network chooses the highest activations, and only classifies images it is the most confident about. Conversely, as more images are let in, the network becomes less confident and the relative accuracy decreases. This is a helpful property because the let-in threshold will converge to letting in fewer images, thus optimizing down to the let-in threshold for each constrained range.

Timed Testing — For the final results, the base and custom models were tested against each other to observe the accuracy and runtime changes. The custom layer added includes the mid-network classification network and max value threshold to determine if the given image will be classified early. The models were tested using the CIFAR10/100 testset and ran alternating sets of epochs to ensure the computer switches tasks. A batch size of one was used for all testing to simulate real-world scenarios. Also, two warm up epochs of each model are run before testing to warm-up the GPU and system. The data was collected using two sets of five epochs, alternating between each set. Some of the smaller models were tested with three sets of five epochs because the faster runtimes were less consistent.

Implementation — At the specified layer, the image tensor is first average pooled down to the desired image size, passed through the classification model, and the max activation is checked against the max value threshold. The custom layer will return the input image tensor, the output of the classification model, and a boolean value. If the max activation is higher than the threshold, the boolean value will toggle to true. In the forward function of the model, the boolean is checked and if true, the model will return the output. If not, the input image tensor is unchanged and is passed into the next layer like usual.

4 Results and Discussion

In this section, the runtime and accuracy changes will be shown from all models to display the validity of the proposed method. Multiple layers were tested for each model on both datasets, ranging from various points in the networks to find the best position for a classification checkpoint. Sections of this data will

be presented and an online appendix of all layers and models is available by request. The accuracy changes were measured by subtracting from the base model accuracy to observe difference and runtime changes were measured by dividing the base model and custom model time to observe percent change.

CIFAR10/100 results, shown in Table 2 below, for a single layer in AlexNet, VGG-16, and ResNet50 [11, 19, 6]. The progression of runtime and accuracy changes can be observed for a series of let-in thresholds. Looking at AlexNet, the accuracy drops off as more images are let in on CIFAR10, but the accuracy is very stable on CIFAR100 with slight accuracy improvements. The runtime did not improve very much because it is a very small model, it takes more than 80% of images to be classified early to provide a benefit. Although, with only five convolutional layers, the mid-network classification performed well with limited accuracy degradation on both datasets.

Table 2: Runtime and accuracy changes with AlexNet, VGG-16, and ResNet50 on both datasets on select layers. For accuracy changes, a positive value is an increase in overall accuracy, whereas a negative value is a decrease.

	AlexNet Layer 3			**VGG-1 Layer 6**			**ResNet50 Layer 18**		
Dataset	Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change
CIFAR10	0.600	0.480	-4.19%	0.200	-0.180	-0.60%	0.102	-0.020	-0.36%
	0.700	0.310	-0.91%	0.300	-0.580	**3.46%**	0.201	-0.190	**5.54%**
	0.800	0.190	**1.70%**	0.400	-1.010	**6.90%**	0.300	-0.590	**10.66%**
	0.900	-0.060	**4.63%**	0.502	-1.710	**11.16%**	0.400	-1.010	**15.80%**
	1.000	-1.080	**7.84%**	0.601	-2.800	**15.36%**	0.501	-1.760	**20.68%**
CIFAR100	0.600	0.880	-6.75%	0.200	-1.610	-1.04%	0.102	-1.070	-0.71%
	0.700	1.040	-3.56%	0.300	-2.890	**2.76%**	0.200	-2.830	**4.72%**
	0.800	1.220	-0.32%	0.400	-4.520	**6.60%**	0.300	-5.110	**10.22%**
	0.901	0.920	**3.50%**	0.500	-6.640	**10.88%**	0.401	-7.680	**15.23%**
	1.000	0.240	**7.60%**	0.600	-9.110	**15.54%**	0.501	-10.81	**20.74%**

Comparatively, VGG-16 and ResNet50 are larger networks, with thirteen and 49 convolutional layers respectively. Thus, there is a greater potential for runtime improvements, with both models needing only 20–30% of images to observe runtime gains. A general trend can be seen, as more images are classified early, the overall accuracy decreases. This is expected because the output classifier is larger and more features have been extracted after more convolutional layers.

With VGG-16, the runtime has much larger improvements but the accuracy degrades much faster compared to AlexNet. The accuracy degradation is very steep in CIFAR100, with almost 10% accuracy loss with only 60% classified early. Although, larger runtime improvements can also be observed; VGG-16 achieves 15% faster results compared to the base model. Overall, VGG-16 did not perform well due to the large accuracy degradation, but there are large

potential runtime improvements.

Similar to VGG-16, ResNet50 displayed very promising runtime improvements with over 20% faster on both datasets with only 50% classified early. Although, the classification was not very accurate and produced significant amounts of accuracy degradation. On the CIFAR10 dataset the degradation was substantially less, but became very high on CIFAR100. For VGG-16 and ResNet50, due to graphics card limitations, small batch sizes were used: 64 & 100, respectively. Thus, base models did not train very accurately and negatively impacted the mid-network classification performance as a result. Nevertheless, both still demonstrate potentially large runtime improvements of a mid-network classifier.

Table 3: Runtime and accuracy changes with AlexNet, VGG-16, and ResNet50 on both datasets on select layers. For accuracy changes, a positive value is an increase in overall accuracy, whereas a negative value is a decrease.

ResNet34 CIFAR100 Results Per Layer

Layer 14			Layer 20			Layer 26		
Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change	Let-In Thres.	Accuracy Change	Runtime Change
0.000	0.000	-7.76%	0.000	0.000	-7.12%	0.000	0.000	-8.99%
0.136	0.070	-3.87%	0.135	-0.040	-4.02%	0.102	-0.010	-6.27%
0.218	0.180	-1.02%	0.203	-0.100	-2.42%	0.255	0.030	-6.00%
0.301	0.160	**2.38%**	0.303	-0.060	-1.22%	0.348	0.050	-4.49%
0.416	0.170	**5.783%**	0.408	0.000	**0.059%**	0.415	0.020	-3.94%
0.501	0.300	**7.21%**	0.505	-0.060	**2.43%**	0.514	0.020	-2.00%
0.604	0.430	**11.46%**	0.601	-0.070	**3.57%**	0.612	0.060	-2.18%
0.701	0.480	**14.75%**	0.701	-0.130	**3.79%**	0.701	0.010	-0.09%
0.801	0.740	**18.83%**	0.800	-0.080	**7.44%**	0.801	0.160	**0.02%**
0.901	0.640	**24.33%**	0.900	-0.070	**9.54%**	0.901	0.180	**1.93%**
1.000	-0.680	**36.52%**	1.000	-0.380	**21.60%**	1.000	-0.290	**10.43%**

With 33 convolutional layers, ResNet34 provides a deep enough network to accurately train and provide sufficient depth to improve runtime [6]. The mid-network classification was implemented after the fourteenth, twentieth, and 26th convolutional layers. Table 3 above shows the full results of each layer tested at each ∼10% let-in threshold interval. The sequence of the runtimes can be seen, with each layer requiring more images to overcome the classification cost. Moreover, each layer has progressively faster runtimes as the secondary clarifier is implemented earlier in the network.

Starting from the right, layer 26 exhibited adequate classification with no accuracy loss. Although, it was too far into the network to produce major runtime improvements, needing about 80% of early classification to offset the cost of the secondary classifier. Layer 20 addressed these issues with negligible

accuracy loss while providing much larger runtime improvements. With a let-in threshold of 90%, layer 20 had a 0.07% loss of accuracy and a 9.5% faster runtime. Lastly, layer 14 was still able to perform an excellent classification while greatly increasing the runtime. In addition, the accuracy did not degrade until the let-in threshold was 100%; with all images being classified early, the network is 36% faster and 0.68% lower accuracy.

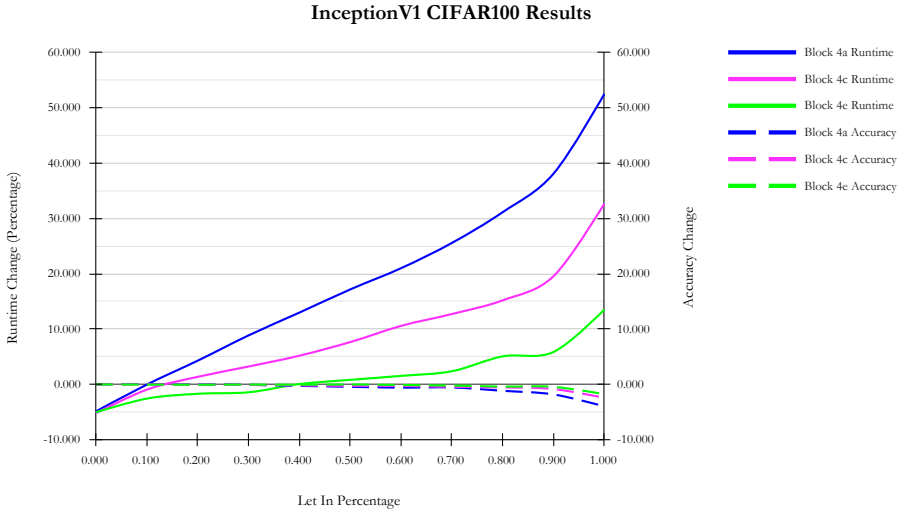


Figure 3: InceptionV1 results on CIFAR100 dataset after the 4a, 4c, and 4e inception blocks. The InceptionV1 network utilized blocks with multiple convolutions known as an *inception block*. The base architecture has a total of nine inception blocks; reference paper for architecture diagram [22].

Similar to ResNet34, the InceptionV1 network performed exceptionally well providing large runtime improvements and very small accuracy degradation [22]. Three separate points were tested, after the 4a inception block, 4c inception block, and 4e inception block. Figure 3 above presents the results for each layer. For the runtime, the results show a progressively greater improvement as the images are classified at earlier points in the network. Each block produces a faster runtime than the later blocks because there are more convolutions to be skipped. In addition, the runtime decreases as more images are let in because the average number of convolutions is reduced; all lines trend upwards as the let-in threshold increases. As seen on the other models, the cost of adding a mid-network classification to a model is about 5% of the runtime and about ten to 40% of images need to be classified early to overcome this cost.

For the accuracy, the trends are also as expected in which each block's accu-

racy degrades as more images are let in and the later blocks degrade less than the earlier blocks. InceptionV1 performed very well overall with a maximum of 15% to 55% faster runtimes and less than 4% accuracy degradation at the highest let-in threshold. Like AlexNet and ResNet34, the mid-network classification was very accurate and the accuracy loss remained negligible, or even slightly positive, until very high let-in thresholds. At around 0.5% of accuracy loss, block 4a's runtime improved by 25%, block 4c's by 15%, and block 4e's by 6%, at 70%, 80%, and 90% let-in threshold respectively.

5 Conclusion

We have investigated the use of an independent, secondary classifier implemented into published architectures for faster classification on CIFAR10/100 datasets. These findings indicate that mid-network classification can give a marginal increase in accuracy, while at the same time, significantly reducing the overall runtime of the model. The specific implementation in the network does need to be surveyed to produce the best results, providing enough information for the classification and enough layers to reduce convolutions. Further investigation should be conducted to assess which types of images or classes benefit from the secondary classifier and if there is any pattern or predictability to the classification improvement. Although these concepts were not verified using the expansive ImageNet dataset, there is sufficient evidence to suggest similar results will be yielded. Therefore, additional experiments should be carried out to investigate the utilization of multiple independent classifiers.

References

[1] Abien Fred Agarap. *Deep Learning using Rectified Linear Units (ReLU).* 2019.

[2] Richard L Burden, J Douglas Faires, and Annette M Burden. *Numerical analysis, 10th.* Cengage learning, 2016.

[3] Siddharth Das. CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more..., 11 2017. `https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5`.

[4] Jeffrey Dean, Greg S Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V Le, Mark Z Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, et al. Large scale distributed deep networks. In *Proceedings of the*

25th International Conference on Neural Information Processing Systems-Volume 1, pages 1223–1231, 2012.

[5] Shiv Gehlot, Anubha Gupta, and Ritu Gupta. SDCT-AuxNetθ: DCT augmented stain deconvolutional CNN with auxiliary classifier for cancer diagnosis. *Medical image analysis*, 61:101661, 2020.

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

[7] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[9] Diederik P Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[12] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[13] Khuyen Le. An overview of VGG16 and NiN models, 3 2021. `https://medium.com/mlearning-ai/an-overview-of-vgg16-and-nin-models-96e4bf398484`.

[14] Y LeCun, K Kavukcuoglu, and C Farabet. Nano-bio circuit fabrics and systems. *IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010.

[15] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[16] Tim Pearce, Alexandra Brintrup, and Jun Zhu. Understanding softmax confidence and uncertainty. *arXiv preprint arXiv:2106.04972*, 2021.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2017.

[18] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[19] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015.

[20] Nitish Srivastava. Improving neural networks with dropout. *University of Toronto*, 182(566):7, 2013.

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

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[23] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

UAV Path Planning using Aerially Obtained Point Clouds*

Alec Pugh[1]
[1]*Department of Computer Science*
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599
alecjp@unc.edu
Luke Bower[2]
[2]*Department of Computer Science*
The University of Alabama in Huntsville
Huntsville, AL 35899
lcb0035@uah.edu
Saad Biaz[3]*, Richard Chapman*[3]
[3]*Department of Computer Science*
Auburn University
Auburn, AL 36849
{biazsaa,chapmro}@auburn.edu

Abstract

With the growing use of unmanned aerial vehicles (**UAVs**) for commercial and military operations, path efficiency remains an utmost concern for battery and time preservation. This paper presents a method for three-dimensional (**3D**) path planning using point clouds obtained from the USGS 3DEP (**United States Geological Survey 3D Elevation Program**) dataset via OpenTopography. The path itself is obtained using the *A* algorithm*, with modifications implemented to account for path smoothing, UAV size, and energy consumption. We also introduce a collision avoidance method using the precomputed data to account for unforeseen obstacles not rendered within the point cloud.

The method presented is designed specifically for an autonomous flight from point clouds obtained via LiDAR (**Light Detection and Ranging**) scans from aircraft, where cavities may be present underneath the surface layer. We use simulations to show the validity of this method.

1 Introduction

The importance of research towards efficient and reliable path finding algorithms has steadily increased with the growing popularity and use of autonomous UAVs. There are many benefits around the development and use of UAV path finding, and each process developed mainly strives to satisfy each of the following requirements: time preservation, path optimality, and danger avoidance. Technological developments have allowed humans to power cars with batteries for hundreds of miles at a time. While such batteries are viable for larger vehicles, versatile UAVs are compact and cannot carry large batteries or payloads due to their size and thrust output. As such, a typical commercially viable UAV has an average flight time of approximately 15-25 minutes [6]. With flight times this short, being able to accomplish more during a mission without recharging is critical to success.

The USGS 3D Elevation Program aims to provide topographical data for the entire United States. Beginning production in 2016, 84% of the nation has topographical data that has been obtained from aircraft LiDAR scans. USGS plans to have the entire US mapped by the end of 2023. These LiDAR scans are displayed in the form of a point cloud, which is a set of 3D points (containing X, Y, and Z coordinates, among other types of data) in space relating to their physical GPS location. The data available through USGS 3DEP can provide accurate path-finding for UAV operations, especially for locations that may be inaccessible to humans or ordinary UAV flight operation due to dangerous conditions, Federal Aviation Administration (**FAA**) regulations, or various other impractical reasons. Furthermore, if this information is confined to a specific operating environment and is obtained prior to a mission launch, optimal paths can be found with less computation.

To this end, the main objective of this paper is to provide a method of path finding using the LiDAR point cloud data from USGS 3DEP. In addition, a secondary objective is to present a method of collision avoidance from the precomputed data, and to automate the dataset retrieval, creating an all-inclusive program.

2 Problem Statement and Motivation

There are many advantages that UAVs hold over manned aerial vehicles, however, there are also many limitations with current technology. One main disadvantage comes from the limited battery life that UAVs possess, so research into battery improvements for UAVs has been vast. While various methods for increasing battery life have been tested, such as unlimited endurance (laser-beam in-flight recharging, tethering, and swapping), along with alternative fuel sources (such as hydrogen, methanol, and hydrocarbons) [1], few of these methods or alternatives have become common in commercial and consumer UAVs due to the limited weight tolerance. Therefore, off-board optimization techniques can be viewed as a solution to help optimize the use of the limited battery life.

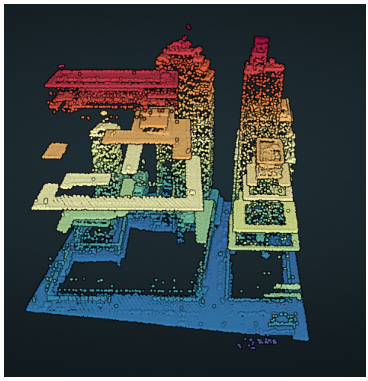


Figure 1: Example of point cloud with cavities present. [4]

The primary objective of our research is to create a reliable and efficient 3D path that works seamlessly with 3DEP scans. Due to these scans being obtained from aircraft, only the surface points of an environment may be scanned accurately. Thus, features that exist directly perpendicular to a surface (such as building faces underneath a roof, or steep cliffs) may be represented as empty space in a 3DEP scan, as shown in Figure 1. The process presented accounts for this problem. Our research simulates a multi-rotor UAV for its quick movement and directional freedom. Due to the path being generated off board from the UAV, on-board equipment can be limited to a 2D LiDAR sensor, GPS, and transmitter as shown in our simulations.

3 Approach

3.1 Matrix Generation

To generate the 3D matrix M that is used for path planning, we first must import the .las file that contains the desired point cloud into *laspy*, a Python library. Additionally, we use *laspy* commands to extract the maximum and minimum point cloud points from the data.

Figure 2: Two cross-sectional 2D grids displaying different heights generated using Algorithm 1. [2]

The difference between the maximum and minimum Z point (Z_{max} and Z_{min}) defines the depth of the matrix. Similarly, the difference between the maximum and minimum Y (Y_{max} and Y_{min}) and X points (X_{max} and X_{min}) define the rows and columns, respectively. In this 2D grid, the X and Y coordinates represent the relative point cloud location, and the value within that cell represents the height of that section in meters (1 cubic meter per cell by default). This is done using code modified from [7], and contains an interpolation gradient to account for data loss from the point cloud data. To convert the 2D grid into a 3D occupancy grid, we loop using height values that start from the minimum Z point to the maximum Z point in the point cloud, creating a temporary 2D grid and assigning a value of 1 or 0 depending on whether or not the cell is an obstacle or not, respectively. This is determined by subtracting the current height value from the height found in the elevation grid at the specified pixel. A threshold value is used to create a *vertical* buffer for object classification within the matrix. The algorithm is displayed as Algorithm 1.

A *horizontal* buffer is created using an implementation of the Breadth First Search (**BFS**) algorithm. Obstacle groupings are identified and expanded outwards by a predefined amount of meters in an effort to prevent potential collisions and account for the UAV size.

3.2 Path Planning and Conversions

To develop the path planning portion of this project, we needed something easily modifiable that would work smoothly with the matrix generation technique discussed in the previous section. For these reasons, the A* path-finding algorithm was selected and used as a base for further modification. To account

Algorithm 1 create3DMatrix(*height_values*, *buffer_size*, *threshold*)

```
M ← matrix
height_values ← H
for cur_height = Z_min to Z_max do
    G ← grid
    for x, y ∈ G do
        if cur_height − H[x, y] ≤ threshold then
            G[x, y] ← obstacle
        else
            G[x, y] ← free
        end if
    end for
    G ← bfs(G, buffer_size)
    M ← append(G)
end for
return M
```

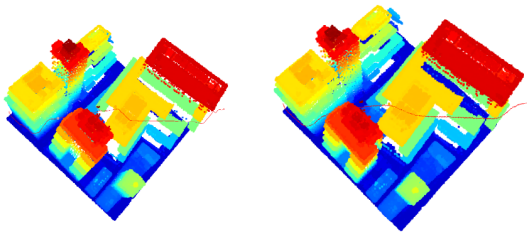


Figure 3: Comparison of non-smoothed and smoothed path. [4]

for all directions in a cubic grid, our A* will evaluate 26 neighbors at every current node. This accounts for the four cardinal directions and diagonals on the same Z plane, as well as the same for one unit above and below, and straight up/straight down. The path obtained from A* consists of 3D points in space in relation to the matrix, with the origin at $(0, 0, 0)$. This path cannot be directly related to the point cloud, as the coordinates of the point cloud are encoded using some reference system (such as WGS84 Web Mercator), so it must first be transformed. This is done by using the maximum/minimum point values from the .las file. With the best path now encoded in WGS84 to coincide the point cloud data, we then apply path smoothing using Bézier curves, which will produce smooth paths through n control points. By using an n-order Bézier curve, we create a smoothed line using each point from the best path as a control point. This provides the most accurate representation of the non-smoothed path, which is shown in Figure 3. We also use this curve to produce a more realistic flight path of a UAV within the simulation, and this in turn shortens the overall distance of the path. However, with the Bézier path smoothing applied, the number of points within the path increases dra-

matically, the higher amount of points results in a more accurate path to be generated due to greater amount of interpolation between each point. Because the focus of this project is for real-life operation, additional conversions must be made to the best path coordinates. Namely, the previously transformed path must be converted to latitude and longitude so that a UAV may receive the path using an onboard GPS. To do this, the Python library *utm* is used. The conversion function must be supplied with the UTM zone of the physical location containing the point cloud scan and whether it exists in the southern or northern hemisphere both gathered from the evaluation of the starting node.

The final path is a 2D list that contains the latitude and longitude coordinates, and the unconverted height (Z value) for each point in the smoothed path. This path is now able to be sent to a UAV, and the accuracy of the conversions from point cloud to real-world GPS waypoints.

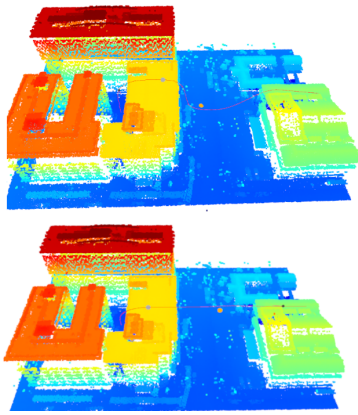


Figure 4: Comparison of path before and after energy model. [3]

Controlling the energy consumption of UAVs is also an important part of battery and resource management, and a primitive model for this is included within our path planning algorithm based off the research done in [5]. The weight of each node is now calculated with an energy consumption factor $e(n)$, which weights neighboring nodes biased on their direction as seen in the formula: $\mathbf{F(n)} = \mathrm{g(n)} + \mathrm{e(n)} + \mathrm{h(n)}$. Table 6 shows the values used for this model. When including this model, the generated path does not make unneeded upwards movement and will only move upwards if absolutely necessary to reach the goal node, as shown in Figure 4.

3.3 Collision Avoidance

The collision avoidance procedure is currently simulated by measuring the distance between the UAV and an obstacle using euclidean distance. We assume a working sensor is onboard the UAV. After the best path has been found using the methodology discussed in Section 3.2, an avoidance trajectory is calculated for each way point in the best path. This is done by evaluating the 3D obstacle matrix in a set of predefined directions by continuously extending a line until an obstacle is met, or a distance threshold is reached. These directions are all oriented the same regardless of UAV heading except for horizontal movement, which is based on the heading H of the UAV (obtained from the best path it is following) and calculated as follows: $H = \arctan \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$. This calculation is done for each waypoint, and the lines that meet the distance threshold are inserted into a priority queue which sorts based on the euclidean distance between the avoidance trajectory and the ending (goal) node. The line with the shortest distance to the end is selected as it will be at the front of the priority queue. The idea behind pre-computing the avoidance trajectory is to reduce the computation time spent on maneuvering during the event of collision avoidance and allow more time to recalculate the path. If an obstacle is met, the general flow of operation is: move UAV to precomputed avoidance trajectory, then recalculate A* from that avoidance trajectory. Algorithm 2 shows how the trajectory calculation is done.

Algorithm 2 findAvoidances(*matrix*, *best_path*, *threshold*)

```
A ← all_avoidance
for p ∈ best_path do
    Q ← priority_queue()
    for d ∈ directions do
        t ← threshold
        while in_bounds(d, matrix) and threshold > 0 do
            p_x ← p_x + d_x
            p_y ← p_y + d_y
            p_z ← p_z + d_z
            t ← t − 1
        end while
        if t == 0 then
            Q ← push(euclidean([p_x, p_y, p_z], end_node), [p_x, p_y, p_z])
        end if
    end for
    A ← append(Q_front)
end for
return A
```

After the avoidance trajectories have been generated, they are run through the same transformations and conversions discussed in Section 3.2. Therefore, the avoidance trajectory and the new path can be sent to the drone with no

issues. To combat being stuck with limited movements, we use a hash table that stores an avoidance direction for each obstacle encountered. If a direction has already been selected and used, and the obstacle still needs to be avoided, the same direction will be used again. If there hasn't been a successful avoidance after x tries, the key is deleted and a new direction is picked and tried.

3.4 Automation using API

A final goal of this research project is to create a seamless process from the input of GPS coordinates to the UAV flying the generated path.

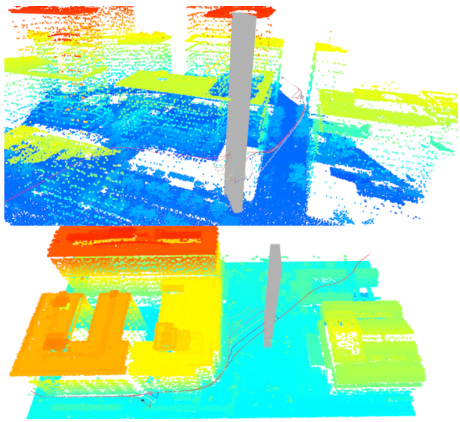


Figure 5: Example of collision avoidance for two obstacle types. [3]

Thus, we needed to utilize OpenTopography's API to allow for remote file downloading. This API allows users to input a search polygon (in the form of minimum and maximum longitude/latitude coordinates) and returns a list of datasets and other information in .json format that contain points in the search area. This .json file also holds critical information regarding each dataset that is extracted from the used dataset. However, we found that not every dataset returned by the API contained points within the search area, thus additional code was used to prune returned datasets from the parsed .json until only the working ones remained. Using *Selenium*, we then automated the process of logging into and downloading a .las file obtained from the API and using that file for the rest of the program remotely. The testing area used was New York, (202,476.83 square feet, coordinates: 40.70502297, -74.01328363 to 40.70357673, -74.01165960) with a maximum height of 156 m. The starting point and ending point used were the top left corner and the bottom right corner and started from the lowest non-obstructed height and ended 10 m above

the starting point. The path was generated by our adapted A* algorithm with a buffer size of 3. Each path was generated using the following settings: 26 nodes (3D movement including diagonals in the Z directions) or 10 nodes of freedom (3D movement without diagonals in the Z directions). In addition, we tested the effects of path smoothing and using euclidean distance for the $g(n)$ cost as well as the $h(n)$ cost (called sqrt in the tables).

4 Testing

4.1 Simulation

Direction	Energy Weight
horizontal	1.0
horizontal angle	1.3
down	1.3
down cardinal	1.5
down angle	1.7
up	8.0
up cardinal	8.4
up angle	8.8

Figure 6: Energy weights used in A* algorithm.

We run these tests with and without the energy function to compare the difference. Path length was measured in units using euclidean distance between each waypoint. All paths generated were possible to fly without collisions.

4.1.1 Energy Function Added

For our energy consumption model testing, we compared the distance of each path, the computational time for the path generation, and the change of elevation in that path for 26-node or 10-node paths with various levels of path smoothing added. The energy model derived from [5] was also included in the path planning algorithm for these tests. Our modifications to the weight of the nodes based on their respective direction is as shown in Table 6. The energy function was implemented using a hash table where the direction is the key, and the weight is the value. Each neighboring node is associated with a direction and the corresponding energy weight value is added when calculating the $F(n)$ cost of the node. This addition was created to try and limit the amount of strenuous maneuvers the UAV preforms in the flight and optimise battery life.

4.1.2 Collision Avoidance

To test collision avoidance, we performed 20 tests each on various point clouds obtained from USGS 3DEP scans in urban environments (cities, downtown areas, etc.). Each simulation contained three different types of obstacles (static sphere, dynamic sphere, and vertical); the purpose of including all at once is to account for potential worst-case scenarios. The obstacle positions were randomly placed along the generated path for each test.

5 Results

After our testing, a few observations can be made about or generated paths. With and without the energy model the 26 node paths outperformed the 10 nodes in all testing in distance at the expense of computation time. The path smoothing did work to shorten the path and avoid obstacles. The energy function eliminated the need for added square root function and generated slightly longer paths than the path without it due to the lack of height change making them less strenuous on the UAV. Our proposed collision detection algorithm successfully avoided all obstacles on 50% to 75% of the tests performed, thus the average of success from all tests is 59%. The higher success rates were attributed to larger point clouds with more flat terrain and less building density. Avoidance failures were most commonly the result of an avoidance needing to go out of bounds, no valid avoidance trajectory generated at a necessary waypoint, and sub-optimal direction selection between two open choices.

6 Conclusion

With the growing topographical data available through programs like USGS 3DEP, path planning for UAVs is more accessible than ever before. We have developed and shown the validity of our process for 3D path planning using aerially obtained point clouds from USGS 3DEP databases with various simulations. In addition, we have added necessary modifications to account for UAV size, added path smoothing, and reduced energy consumption. We have also shown a primitive collision detection algorithm that uses the point cloud data to pre-compute avoidance trajectories along the flight path to be used if needed. The process for obtaining point clouds for a particular area has been fully automated with the use of Selenium and the OpenTopography API. Through testing different settings of our path generation process, we have found the settings (path smoothing, sqrt function, # of nodes checked) that will most likely lead to the shortest path being generated with and without the use of our energy function. Finally, we have shown the precision of our

method in converting the path into latitude and longitude to be used with a GPS-equipped UAV via ARDU Pilot.

While our process works well in simulation, real-life testing and the continuation of testing different factors to improve optimality should be tested. Further work to improve the collision detection algorithm must be done to increase avoidance success rates. The addition of changing some values like grid resolution should also be implemented in the future. For data tables and a more comprehensive literary analysis, refer to our full technical paper at: https://www.eng.auburn.edu/files/acad_depts/csse/csse_technical_reports/csse22-02.pdf

References

[1] A critical review on unmanned aerial vehicles power supply and energy management: Solutions, strategies, and prospects. *Applied Energy*, 255:113823, 2019.

[2] USGS 3DEP (2021). Ga central 2 2018. u.s. geological survey 3d elevation program, distributed by opentopography.

[3] USGS 3DEP (2021). Wa kingco 1 2021. u.s. geological survey 3d elevation program, distributed by opentopography.

[4] USGS 3DEP. Ny newyorkcity. u.s. geological survey 3d elevation program, distributed by opentopography.

[5] Hasini Viranga Abeywickrama, Beeshanga Abewardana Jayawickrama, Ying He, and Eryk Dutkiewicz. Comprehensive energy consumption model for unmanned aerial vehicles, based on empirical studies of battery performance. *IEEE Access*, 6:58383–58394, 2018.

[6] Boris Galkin, Jacek Kibilda, and Luiz A. DaSilva. Uavs as mobile infrastructure: Addressing battery lifetime. *IEEE Communications Magazine*, 57(6):132–137, 2019.

[7] Joel Lawhead. *Learning Geospatial Analysis with Python*. Packt Publishing, Birmingham, United Kingdom, 2019.

Improving Student Motivation Through an Alternative Grading System*

Ryan Stephen Mattfeld
Computer Science Department
Elon University
Elon, NC 27244
rmattfeld@elon.edu

Abstract

The traditional grading system has significant shortcomings in effectiveness and reliability. In addition, the traditional grading system encourages extrinsic motivation in students rather than intrinsic motivation. This reliance on external goals and factors for motivation tends to increase students' anxiety and inclination to cheat. This problem is a growing concern as students gain access to new tools which make cheating quicker and easier. Several ideas for alternatives to the traditional grading system include contract grading, mastery grading, specifications grading, and ungrading. In this study, an alternative grading system combining many of these concepts was applied to three sections of an upper level computer science elective course. To evaluate the effectiveness of this system, questions from the Motivated Strategies for Learning Questionnaire (MSLQ) were asked in a pre- and post- term survey. Statistically significant improvements were found in Self-Efficacy and Learning Performance, Task Value, and Control of Learning Beliefs. These findings support the idea that alternative grading systems may be an effective course adjustment to improve student motivation and learning in an upper level computer science course.

1 Introduction and Related Work

Since its inception, the traditional numeric grading system has received criticisms regarding its effectiveness and reliability [2]. Grading has been a hall-

mark of the education system and is widely seen as a standard to indicate how well a student has learned the material covered in the course. However, the traditional grading system's ability to accurately provide this information has been debated [11]. The reliability of determining grades, both among different instructors as well as repeated grading by one instructor, have been evaluated [13]. These studies have shown inconsistent results among the reliability of grades, the techniques to measure reliability, or the ways to improve grading reliability.

In addition to concerns with the effectiveness, reliability, accuracy, and necessity of grading, intrinsic motivation, or the desire to learn for its own sake, has been shown to be adversely affected by our grading systems [3]. Conversely, an increase in extrinsic motivation, or focus on external rewards and punishments, has been seen in students as a result of grading, which, in turn, increases student anxiety and motivation to cheat [6]. It is possible that this emphasis on extrinsic motivation and its resulting increase in cheating inclination may become even more problematic with the development of new tools like ChatGPT. Thus, encouraging use of grading systems which increase intrinsic motivation and decrease extrinsic motivation may be essential in creating an environment that fosters learning.

Traditional grading systems appear to produce a negative effect on equity, causing particular harm on traditionally underrepresented groups. Instead, current assessments reflect factors such as language proficiency, cultural background, or skills in test taking [11]. Changes in traditional grading practices, like using blind grading and using grading rubrics may minimize some of the inequitable effects of grading. In addition, there are a number of other alternative grading strategies which may reduce inequity and provide equal opportunity for all students [4].

Some alternative grading systems have already been developed. Contract grading is a system that allows students to make choices about what, how, and when to learn [5]. Specifications grading is a system which de-emphasizes numeric grades by giving credit to a student only for work when it fully satisfies a given set of requirements [7]. This results in grades of satisfactory or unsatisfactory, and it is typically paired with the ability to resubmit assignments. There are a number of other ideas for de-emphasizing grades or changing grading practices to de-emphasize grades [1, 4]. Experiences, details, and feedback on the results of implementing these systems in a variety of courses is valuable in developing systems which can be widely adopted.

Several recent studies have examined the impact of alternative grading systems on grades [10] and student motivation [12]. This work evaluates how a modified version of ideas from specifications grading and Grading for Equity impacts student motivation in three sections of an upper level computer science

elective. The contributions of this research are:

1. To describe a concrete implementation of an alternative grading system in three sections of an upper level undergraduate computer science course
2. To report results from changes in student attitude and motivation as measured by surveys given at the start and end of the term
3. To provide student and instructor feedback with respect to the alternative grading system implemented

2 Alternative Grading System

2.1 Grading System Description

The grading system used combines concepts from specifications grading [7] and grading for equity [4] to achieve goals of allowing for re-submission of assignments, retaking of examinations, and reducing numeric grades. Ideally, all numeric grades would be eliminated, instead replaced with two categories: complete/incomplete. However, some of the activities in the course include automated assessment from the online textbook and from quizzes in the course management software which report results as numeric values. So, numerical thresholds for complete/incomplete tags were determined and included in the evaluation system provided to the students as seen in Table 1.

Table 1: Mapping of assignment goals to letter grade targets

Assignment (qty)	Goal	A	B	C
Readings (15)	Complete ($\geq 85\%$)	All but 2	All but 3	All but 4
Labs (23)	Complete ($\geq 85\%$)	All but 2	All but 4	All but 6
Regrades	Use no more than	5	6	7
Exams (4)	Complete ($\geq 85\%$)	All	All but 1	All but 2
Exam Retakes	Use no more than	2	3	4
Final Exam	Score at least	$\geq 90\%$	$\geq 80\%$	$\geq 70\%$

For numeric grades provided by automated grading tools, a threshold of 85% was selected as an indicator that a student had a reasonable understanding of the assessed material and were classified as Complete. Manually graded assignments did not include a numeric grade and instead were identified as complete or incomplete and included detailed feedback for student improvement. Readings were small out of class assignments given in an on-line textbook. Labs were larger assignments which were started in class, and if necessary, completed by students outside of class. A typical class session included 45

minutes (out of 100 total minutes) for work on labs. Exams were given in class at the end of each major unit of content (Database basics, Complex queries, Database design and configuration, and Database programming in Python). The final exam was comprehensive and included content from each of the four units completed throughout the course.

2.1.1 Regrade requests

Philosophically, it would be ideal to allow unlimited regrades; however, there were concerns about the number and quality of re-submissions that may result from this practice. With a goal of reducing this burden, categories were created to link the number of regrades used to a grade letter category. Students were encouraged to redo any work that did not meet the threshold for complete, making use of the feedback provided. The guidelines for the number of regrade requests were included in Table 1. For example, students could submit a re-grade request for up to 5 labs and/or reading activities while still meeting the A level goal for the regrade category. In addition to reading and lab regrade requests, exams could be retaken with versions covering the same content but with different questions and/or datasets. Exam retakes could be requested and could be taken within a set window of time which doubled as office hours each week.

2.2 Survey Development

Selections from the Motivated Strategies for Learning Questionnaire (MSLQ) were included in a survey provided at the beginning and end of the term [8]. The MSLQ has been extensively used in higher education as a tool to measure the components of reflective learning. In addition, usage of this questionnaire has shown strong internal consistency and reliability [9]. The MSLQ uses a 7-point Likert scale to measure a variety of categories of student motivation and learning. It is designed to be modular so that specific topics of interest can be evaluated individually. In this study, six scales which focus on student motivation (with 31 total questions) were selected for the survey:

- Control of Learning Beliefs (4 questions): how much does a student feel that their own efforts will enable them to learn the course material?
- Extrinsic Goal Orientation (4 questions): to what extent is a student motivated by external factors such as grades or competition?
- Intrinsic Goal Orientation (4 questions): to what extent is a student motivated by course tsks themselves as opposed to external factors?
- Self-efficacy for Learning and Performance (8 questions): how confident is a student in their ability to learn the course material and to be successful and perform well in the course?

- Task Value (6 questions): how valuable does a student find the subject matter?
- Test Anxiety (5 questions): how much do negative thoughts and emotions affect a student while taking a test?

In addition to the MSLQ questions, students were asked several questions specifically about the new grading system including an open ended response question soliciting their opinions.

3 Results

This study examined three different sections of Database Systems, an upper-level Computer Science elective course during the 2022-2023 academic year at a liberal arts institution in the southeastern United States. These three sections included a total of 67 students. Students included primarily those in their 3rd and 4th years with a mix of Computer Science majors, Computer Science minors, and Data Science minors. Classes met for 100 minutes twice per week, and class sessions employed an active learning approach including short lectures, interactive demonstrations, and daily hands-on time for working on labs and other course activities. Pre- and post- term surveys including the questions described in Subsection 2.2 were provided in the first and last weeks of the term. The data from the 57 students who completed both surveys were retained for this analysis.

3.1 Changes in Student Motivation

Changes in student motivation were assessed by examining results in each of the 6 categories of student motivation described in Section 2.2. Figure 1 shows the average change among all students in each category. Three of these differences showed statistically significant changes using a dependent t-test for paired samples: Self-efficacy for Learning and Performance ($p<0.001$), Task Value ($p=0.005$), and Control of Learning Beliefs ($p=0.027$).

The largest change in student responses is in the category of MSLQ questions assessing self-efficacy for learning and performance. This category assesses student expectancy for success and self-efficacy. The expectancy for success indicates higher performance expectations within the course. Self-efficacy is a self-appraisal of one's ability to accomplish a task and one's confidence in the skills to perform that task. It is possible that the improvement was driven by providing opportunities for students to redo assignments and learn from their initial mistakes. This may have allowed students to gain more confidence in learning the material in the course and have a greater sense of self-confidence through analysis and correction of mistakes.

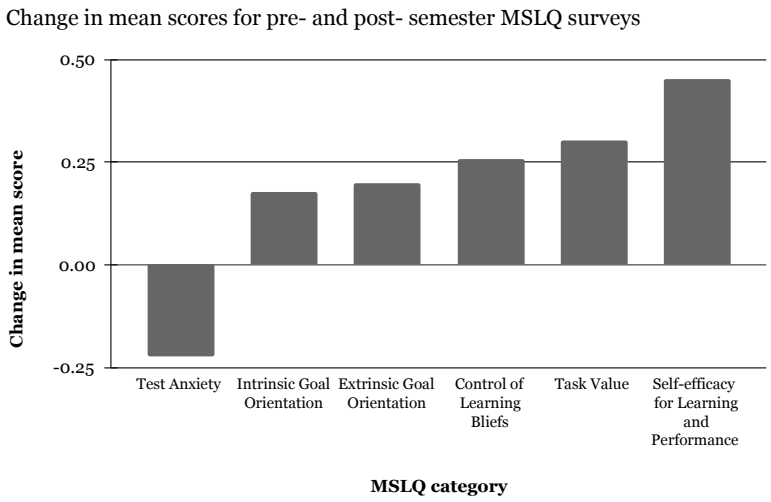


Figure 1: Change in student motivation. A positive number indicates a higher average score on the post-term survey as compared to the pre-term survey

The task value category identifies how interesting, how useful, or how important the course is to the student. This specific finding may be more directly related to course material and less related to the grading system than the other categories. However, it is possible that this was influenced by the grading system and that positive results here are driven by the increased attention that students gave to the material.

The Control of Learning Beliefs category seeks to assess how strongly the student believes that their efforts to learn the material will result in positive outcomes. It is worth noting that student comments about the course indicate that the well-defined grade targets and the option to use regrades or retakes of exams may have contributed to this increase. The question with the largest increase in average response within this category was "It is my own fault if I don't learn the material in this course". This, combined with student feedback may indicate that within this alternative grading system, students felt that they had many opportunities to demonstrate their knowledge.

The changes in the other three categories were not statistically significant. The desired result in Extrinsic Goal Orientation would be a decrease; however, the increase seen was small enough to not be statistically significant. Another desire result would be to see a statistically significant increase in Intrinsic Goal Orientation. In both cases, it is possible that the retention of numerical

($\geq$ 85%) rather than purely categorical assessment (utilized as a concession to available course management software) could have retained too much emphasis on traditional grading thoughts and ideologies. In addition, throughout the course, much more time than is typical was spent discussing the grading system itself (due to its novelty). Having these additional discussions may have emphasized grades to a greater degree rather than de-emphasizing them. The mean responses evaluating Test Anxiety decreased; however, the change fell short of being statistically significant. However, multiple students also specifically highlighted the reduction in stress due to the grading system. Here is one such student quote: "There are a lot of options to ensure students learn and pass at the same time while avoiding unnecessary stress.". It is clear through student feedback that at least some of the students felt reduced stress and anxiety due to the retake and regrade policies.

3.2 Regrades and Exam retakes

The number of regrades and retakes utilized by students is shown in Figure 2. Although there was significant concern before beginning the course that there would be an overwhelming number of regrades and retakes, 48% of students never used a single regrade or retake opportunity. Additionally, 80% of students used 2 or fewer. In 46.9% of cases, if an exam was returned as Incomplete with feedback, it was retaken. However, if retaken, only in 26.7% of cases were students able to improve enough to receive a result of Complete. Although multiple retakes were allowed for any exam, there was only 1 case where a student opted to retake the same exam twice, and a result of Incomplete was received for each attempt. Overall, including regrades and exam retakes appeared to be effective at providing opportunities for students to demonstrate their knowledge despite challenges faced outside of the classroom that may cause one or many assignments to be a poor reflection of the student's actual ability.

4 Discussion and Conclusions

An alternative grading system combining elements of specifications grading and ungrading was implemented in three sections of an upper level computer science course. The grading system provides categorical feedback (Complete/Incomplete) on assignments with grades identified based on the number of assignments meeting the threshold for Complete. Students were provided opportunities to have assignments regraded and retake exams. In order to assess changes in student motivation, students were surveyed at the start and end of the term using questions from the MSLQ. The survey results indicated that

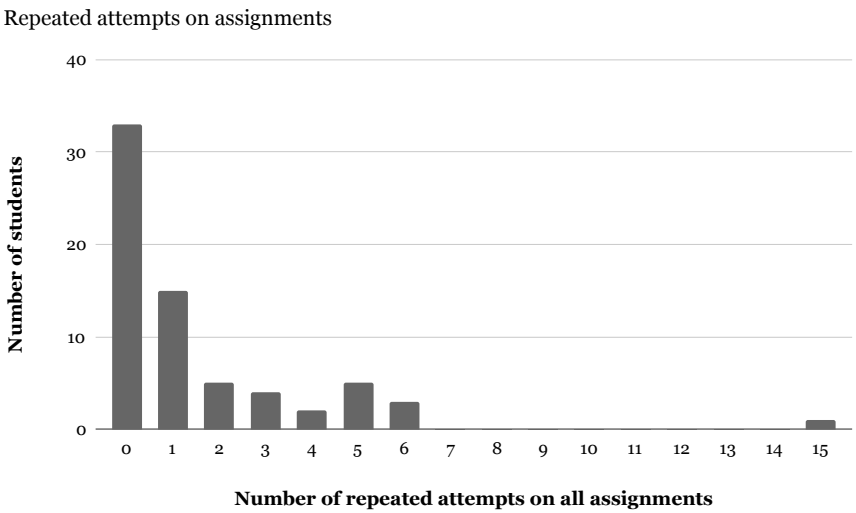


Figure 2: Histogram of the number of combined regrades and exam retakes utilized by students.

Self-Efficacy for Learning and Performance, Task Value, and Control of Learning Beliefs showed statistically significant increases.

There were some limitations which may have impacted results. The course management software was unable to entirely hide grade percentages, and software used to autonomously grade some assignments included traditional percentages. Because of this, a grade threshold was included to allow conversion between a traditional percentage grade and a result of Complete or Incomplete. This link to traditional grading may have re-emphasized the importance of grades. Alternative solutions which allow this link to be removed will be examined for future implementations of this grading system.

The number of regrades and exam retakes was lower than feared before the term began. In all cases, they provided an opportunity for students to learn from feedback and demonstrate greater knowledge of the course content. However, in the case of exam retakes, there were a large number of retakes which did not display improvement. It was noticed that students who came to office hours to discuss their results and review areas in which they struggled generally seemed to be more successful, though this data was not recorded. In future implementations of this system, it will likely be a requirement that a student meet with the instructor prior to retaking an exam in order to ensure

that the student has the opportunity to fully understand their feedback and improve.

The instructor and student experiences from using this grading system was positive. There were noticeably more conversations during office hours about course content and reviewing material to gain a better understanding. There were also fewer conversations about grades. Students indicated reduced anxiety, both verbally and through written survey responses. This decrease in anxiety generally seemed to be tied to the option to retake or redo exams and assignments. It seemed as though this decrease in anxiety generally led students to be less inclined to cheat and more inclined to discuss errors in their work with the instructor, though this is exceptionally difficult to measure in a quantitative way.

There were, however, significant extra burdens in developing alternative versions of exams, proctoring exam retakes, and regrading readings and labs. To mitigate these challenges in the future, time will be spent investigating and possibly developing methods of automating generation of varied test questions. In addition, methods for generating labs and readings that will provide categorical rather than numeric feedback and track re-submissions will be investigated. Ideally, no reference to traditional numerical grades will be included in this grading system. Finally, there was one instance in which having a grading category tied to minimizing regrades caused a student to avoid resubmitting some missed material in order to maximize their grade. This situation is directly opposed to the intent for providing regrades. Future iterations should be designed such that a student does not have to choose between improving their grade and completing the exercises which foster learning. Despite the limitations, this grading system improved the student's and instructor's experiences.

References

[1] Susan Debra Blum. *Ungrading: Why rating students undermines learning (and what to do instead).* West Virginia University Press, Morgantown, West Virginia, 2020.

[2] Susan M. Brookhart, Thomas R. Guskey, Alex J. Bowers, James H. McMillan, Jeffrey K. Smith, Lisa F. Smith, Michael T. Stevens, and Megan E. Welsh. A century of grading research: Meaning and value in the most common educational measure. *Review of Educational Research*, 86(4):803–848, 2016.

[3] Ruth Butler. Enhancing and undermining intrinsic motivation: The effects of task-involving and ego-involving evaluation of interest and performance. *British Journal of Educational Psychology*, 58(1):1–14, 1988.

[4] Joe Feldman. *Grading for equity: What it is, why it matters, and how it can transform schools and classrooms.* Corwin Press, Thousand Oaks, California, 2018.

[5] Tammy Bunn Hiller and Amy B Hietapelto. Contract grading: Encouraging commitment to the learning process through voice in the evaluation process. *Journal of Management Education*, 25(6):660–684, 2001.

[6] Tamera B. Murdock and Eric M. Anderman. Motivational perspectives on student cheating: Toward an integrated model of academic dishonesty. *Educational Psychologist*, 41(3):129–145, 2006.

[7] Linda B Nilson. *Specifications grading: Restoring rigor, motivating students, and saving faculty time.* Stylus Publishing, LLC, Sterling, Virginia, 2015.

[8] Paul R Pintrich et al. A manual for the use of the motivated strategies for learning questionnaire (mslq). 1991.

[9] Paul R Pintrich, David AF Smith, Teresa Garcia, and Wilbert J McKeachie. Reliability and predictive validity of the motivated strategies for learning questionnaire (mslq). *Educational and psychological measurement*, 53(3):801–813, 1993.

[10] Kevin R. Sanft, Brian Drawert, and Adam Whitley. Modified specifications grading in computer science: Preliminary assessment and experience across five undergraduate courses. *J. Comput. Sci. Coll.*, 36(5):34–46, 2021.

[11] Jeffrey Schinske and Kimberly Tanner. Teaching more by grading less (or differently). *CBE—Life Sciences Education*, 13(2):159–166, 2014.

[12] Scott Spurlock. Improving student motivation by ungrading. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, page 631–637, New York, NY, USA, 2023. Association for Computing Machinery.

[13] Jo Tisi, Gillian Whitehouse, Sarah Maughan, and Newman Burdett. A review of literature on marking reliability research. *Ofqual/13/5285*, 2013.

A Comparison of Machine Learning Code Quality in Python Scripts and Jupyter Notebooks*

Kyle Adams[1] *and Aleksei Vilkomir*[2] *and Mark Hills*[3]
[1]*Moravian University*
Bethlehem, PA
adamsk04@moravian.edu
[2]*Department of Computer Science*
East Carolina University
Greenville, NC
vilkomira21@ecu.edu
[3]*Department of Computer Science*
Appalachian State University
Boone, NC
hillsma@appstate.edu

Abstract

Jupyter notebooks are currently one of the most popular environments for Python development, especially in domains such as data science. Existing studies have shown that notebooks may promote bad coding habits, leading to poor code quality and challenges with replicating notebook results. In this paper, we compare the code quality of Python machine learning code found in Jupyter notebooks to that found in regular Python scripts. The goal of this work is to better understand how the machine learning code created in Jupyter notebooks differs both from machine learning code provided in scripts and from the larger body of Python code, with the aim of creating tools to better support both data science students and practitioners.

1 Introduction

According to a study in 2018 [10], Python is the most used language for machine learning on GitHub, as well as the third most used programming language on GitHub overall. One of the most popular environments for Python programming is Jupyter notebooks [5]. Jupyter notebooks provide a literate programming environment, where Python code can be combined with descriptive text and computational results. These results can be in text, but formats such as images and even video are also supported. While Jupyter notebooks can be used for a wide variety of different application domains, they were created with a focus on supporting applications in data science and scientific computing.

With the growth of interest in data science, millions of Jupyter notebooks have been created and shared on sites such as GitHub and Kaggle, a website that hosts data science competitions. Recent work (see Section 2) has explored issues with the quality of the code in these notebooks. For instance, the work of Grotov et al. [11] compared the quality of the code found in Jupyter notebooks to that of Python scripts. This comparison was done across all domains, while noting that results may differ if focused on a specific domain. In this paper, we narrow this prior comparison to focus specifically on machine learning code, while also comparing our results to those found in this earlier work.

The rest of this paper is organized as follows. First, Section 2 discusses related work. Section 3 then details the corpus of Python scripts and Jupyter notebooks used to conduct this study. Section 4 describes the process used to extract metrics and style errors from the code and to compute the results reported in Section 5. Section 5 evaluates our initial results, including a comparison to results found in earlier work. Lastly, Section 6 concludes the paper and describes possible future applications of our results. All code and data sets used in this paper are available in a replication package on Zenodo [6].

2 Related Work

Some of the prior work on Jupyter notebooks focuses on how notebooks are used in practice, including for collaboration and reproducibility. Kery et al. [15] interviewed 21 professional data scientists to learn more about how they use Jupyter notebooks, and also conducted a survey of 45 data scientists to learn more about potential tool support for allowing data scientists to interact with a history of notebook changes. Chattopadhyay et al. [9] used a combination of a survey and interviews to identify "pain points" encountered by users of computational notebooks, including Jupyter notebooks. Pimentel at al. [17] studied the use of both good and bad coding practices in Jupyter notebooks, with a focus on the impact of these practices on reproducibility. They used the results of

this study to recommend best practices for developing notebooks. Pimintel et al. [16] then extended this work with additional analyses and the introduction of a tool, Julynter, that suggests notebook modifications for improving reproducibility. Quaranta et al. [18] also focused on best practices, specifically for collaboration. Wang et al. [26] looked at notebook documentation, evaluating a system named Themisto that automatically generates notebook documentation based on best practices gleaned from a collection of 80 top-rated notebooks from Kaggle. Additional work on collaboration with notebooks includes that of Zhang et al. [30], which explored collaborative roles and practices; and Rule et al. [20], which evaluated the impact of a cell folding extension to Jupyter notebooks on notebook understandability and reuse. Beyond this, papers by Rule et al. [21] and Amershi et al. [7] focused on how notebooks are used for different purposes and on how teams collaborate on machine learning tasks.

Other prior work focuses specifically on tools and analyses for understanding and improving the use of notebooks. Kery and Meyers [14] described and evaluated Verdant, a system for versioning notebook artifacts. Head at al. [12] described an analysis that uses program slicing over execution logs to compute notebook dependencies, along with *code gathering* tools that track notebook history. Wenskovitch et al. [28] designed a visualization tool, Albireo, that allows for multiple levels of visualization as well as exploration of cell dependencies. Wang et al. [27] focused on the need for improved tools for analyzing Jupyter notebooks, showing that notebooks include substantial amounts of poor-quality code. Yang et al. [29] used code summarization techniques to show the impact of *data wrangling* code by showing the effect of the code on the data being analyzed by the notebook. Venkatesh and Bodden [24] presented an analysis and tool that generates cell headers for notebook cells based on the position of the cell in a broader machine learning workflow. Titov et al. [22] described an analysis and tool, ReSplit, that refactors existing notebook cells. Jiang et al. [13] presented an algorithm for generating a dependency graph between cells in a Jupyter notebook and labeling these cells with the relevant machine learning state, based on the API calls used in the cell. Quaranta et al. [19] introduced Pynblint, a linter for Jupyter notebooks that checks for violations of best practices and provides fix recommendations.

Grotov et al. [11] conducted a comparison of Python code in Python scripts and in Jupyter notebooks. Scripts and notebooks were compared using a combination of structural metrics (e.g., SLOC, cyclomatic complexity) and code style violations. The corpus included all Jupyter notebooks publicly available on GitHub, as well as all Python scripts from the 10,000 most starred Python repositories on GitHub, narrowed to use only those notebooks and scripts released under a permissive software license. This resulted in 847,881 notebooks and 465,776 scripts. To detect style errors, the authors used Hyperstyle [8], a

tool for detecting style violations in student programming assignments. A second tool, named Matroskin, was created to work with notebooks, including to compute structural metrics. One specific style issue, that of excessive imports, was also noted by Vilkomir [25]. Our work builds upon, and is compared with, the earlier work of Grotov et al. and Vilkomir, focusing on differences in code quality between scripts and notebooks used specifically for machine learning.

3 Corpus

To carry out our comparison, a corpus of Python scripts and Jupyter notebooks, focused specifically on machine learning, was needed. Kaggle [2] is a website dedicated to machine learning and data science. It holds data science competitions, some with cash prizes, and makes some of the solutions (those the authors allow to be released publicly) available for public inspection and download. Some of the published solutions are created using Jupyter notebooks, while some are created using Python scripts. Some are also created using other languages (e.g., R), but those are ignored here.

To get the initial corpus of data, we used Meta Kaggle [4]. Meta Kaggle is a public dataset, published by Kaggle, that contains information about competitions and the public solutions posted as part of these competitions. Using this data set, a Python script was used to extract the information needed to download each solution. The projects were sorted by the user rating (stars) they received on Kaggle in descending order. We decided to limit our corpus to the top 100,000 projects. Next, the Kaggle API was used to download the project code corresponding to each project. This download process was executed as part of a Bash script for repeatability. The result of executing this script was a collection of Python script files, Jupyter notebooks, and potentially other files that may have been included with the solution.

Using the Kaggle API, a total of 69,858 machine learning files were obtained. Only files written in Python (either notebooks or scripts) were processed further. These files were split into two datasets, one for Python scripts, and one for notebooks. In total, there are 12,136 Python scripts (with a .py extension), and 57,722 Jupyter notebooks (with a .ipynb extension) in these two datasets. The number of files in the final corpus is different from the initial size we planned to obtain because some files were unavailable for download.

4 Methodology

We have used a methodology that mirrors that found in the paper by Grotov et al. [11] to allow for a comparison of our results, focused just on machine learning code, with their results, which looked at notebooks and scripts in

general (including machine learning code). This includes using and reporting the same structural metrics they used over their dataset.

Matroskin [3, 11] is a library designed for analyzing Juptyer notebooks. It focuses on computing structural metrics for the Python code included in the notebooks. Hyperstyle [8, 1] is a tool that is able to analyze code for style errors. It works for code written in Python, Java, or Kotlin, and can follow different style guides, such as PEP-8 [23]. To compute metrics over the code in our corpus, several scripts were developed to automate the use of Hyperstyle and Matroskin. These scripts extract the results of running these tools and aid in the analysis of the metrics and style errors. Other scripts were developed to convert between notebook and script formats, which allowed the scripts (after conversion to notebook format) to be processed using Matroskin and the notebooks (after conversion to script format) to by analyzed using Hyperstyle.

4.1 Matroskin Methodology

Matroskin was developed to handle a large number of notebooks at once, therefore we were able to run the tool on our entire corpus of notebooks by running it on the directory containing our dataset of downloaded notebooks. The script that runs the tool is a variant of a script provided with Matroskin, edited in order to output the results in a format more amenable to further processing. Results of running the script were stored in a single JSON file for each processed notebook. To avoid problems with name collisions in cases where multiple notebooks used the same file name, the results were stored in files that were each given a unique number (e.g., 0.json, 1.json), with a separate CSV file mapping from the original file name of the processed notebook to the relevant JSON file containing the analysis results.

This process was then repeated on the dataset of Python scripts. The analysis first converted each script into a notebook with a single cell containing the script code. The same process described previously was then used to process these converted files: information on each notebook was extracted and saved into a JSON file, with information to link these files back to the related notebook (and thus, the original script).

A separate script was created to analyze the results stored in these JSON files. This script reads the results saved in the JSON files, computing the results shown in Section 5. During this process, we could not process 385 of the JSON files generated for the notebooks and 154 of the JSON files generated for the scripts due to an error in the result format. This error appears to be in the information generated by Matroskin, but further work is needed to determine the actual source of the error.

4.2 Hyperstyle Methodology

Hyperstyle was run individually on each Python script in the corpus, with results output to individual files for later processing. A bug in Hyperstyle, which we have not yet isolated, would cause it to get stuck at random points during the analysis if run repeatedly on a sequence of input scripts. This required us to process these files outside of the script we developed to automate this process. In this way, it was possible to process all scripts in the dataset. While Hyperstyle also includes an option to process an entire directory, the information returned by Hyperstyle in that case differs from the per-file results, generally reporting many fewer style issues. The process used for the scripts was then repeated on the dataset of notebooks. A script was used to first convert each notebook into a script file, which was then processed by Hyperstyle.

A separate analysis script then processes the Hyperstyle result files, generating the results seen in Section 5. During processing, for the results from the collection of scripts, one result file was unable to be processed due to a UnicodeDecoder Error and thirteen results files were unable to be processed due to syntax errors. These are based on characteristics of the original analyzed files. For the results from the collection of notebooks, three results files were unable to be processed due to syntax errors.

5 Evaluation

In this section, we present the results of both the structural and style analyses described in Section 4. For the structural results, the mean (written as M) and standard deviation (written as STD) are shown. Using the mean of the results allows a better comparison of the two datasets due to their difference in size.

5.1 Evaluation of Structural Metrics

The results of comparing the structural metrics between scripts and notebooks can be found in Table 1. We found that the notebooks (M = 174.48, STD = 216.45), on average, contained more source lines of code than the scripts (M = 105.02, STD = 180.11). Notebooks (M = 27.93, STD = 48.51) and scripts (M = 26.96, STD = 57.13) have a similar number of commented lines unless markdown cells ("extended comments", which may be actual comments on the code or descriptive text to be shown as part of the notebook) are included, in which case notebooks (M = 75.40, STD = 113.45) are shown to have almost three times as many comment lines as scripts.

While they tend to have fewer lines of code, the scripts (M = 12.00, STD = 23.71) are shown to have a higher cyclomatic complexity as compared to notebooks (M = 6.18, STD = 9.28). Also, scripts (M = 1.00, STD = 0.24) have,

Table 1: Structural Metrics, Notebooks and Scripts

Metric	Notebook Mean (STD)	Script Mean (STD)
SLOC	174.48 (216.45)	105.02 (180.11)
Comments LOC	27.93 (48.51)	26.96 (57.13)
Extended Comments LOC	75.40 (113.45)	N/A (N/A)
Blank Lines Count	30.42 (46.97)	32.53 (46.69)
Cyclomatic Complexity	6.18 (9.28)	12.00 (23.71)
NPAVG	0.71 (0.27)	1.00 (0.24)
API Functions Count	6.95 (6.69)	10.20 (10.06)
API Functions Uses	13.17 (20.24)	23.29 (81.93)
Defined Functions Count	3.45 (6.00)	3.39 (6.86)
Defined Functions Uses	5.83 (13.31)	4.72 (12.91)
Built in Functions Count	4.83 (3.30)	3.80 (3.20)
Built in Functions Uses	22.51 (35.88)	14.78 (25.26)
Other Functions Uses	86.02 (99.11)	28.73 (54.03)
Cell Coupling	48.44 (358.84)	N/A (N/A)
Function Coupling	10.49 (101.02)	12.59 (105.80)

on average, a higher number of inputs per function (NPAVG) than notebooks ($\mathsf{M} = 0.71$, $\mathsf{STD} = 0.27$). When it comes to function usage, scripts ($\mathsf{M} = 10.20$, $\mathsf{STD} = 10.06$) include more unique API function imports than notebooks ($\mathsf{M} = 6.95$, $\mathsf{STD} = 6.69$). Scripts ($\mathsf{M} = 23.29$, $\mathsf{STD} = 81.93$) also use those API functions more than the notebooks ($\mathsf{M} = 13.17$, $\mathsf{STD} = 20.24$). For built-in functions, notebooks ($\mathsf{M} = 4.83$, $\mathsf{STD} = 3.30$) take advantage of a larger number of different functions than the scripts ($\mathsf{M} = 3.80$, $\mathsf{STD} = 3.20$). The notebooks ($\mathsf{M} = 22.51$, $\mathsf{STD} = 35.88$) also call the built-in functions much more than scripts ($\mathsf{M} = 14.78$, $\mathsf{STD} = 25.26$). When it comes to user-defined functions, notebooks ($\mathsf{M} = 3.45$, $\mathsf{STD} = 6.00$) have about the same amount as scripts ($\mathsf{M} = 3.39$, $\mathsf{STD} = 6.86$). For number of uses of the user-defined functions, notebooks ($\mathsf{M} = 5.83$, $\mathsf{STD} = 13.31$) again have more uses than scripts ($\mathsf{M} = 4.72$, $\mathsf{STD} = 12.91$). For any other functions, the number of uses in notebooks ($\mathsf{M} = 86.02$, $\mathsf{STD} = 99.11$) is significantly greater than in scripts ($\mathsf{M} = 28.73$, $\mathsf{STD} = 54.03$). Lastly, coupling (use of shared/common elements) between functions is greater in scripts ($\mathsf{M} = 12.59$, $\mathsf{STD} = 105.80$) than notebooks ($\mathsf{M} = 10.49$, $\mathsf{STD} = 101.02$) unless we consider cells to be functions (cell coupling), in which case coupling is much higher in notebooks ($\mathsf{M} = 48.44$, $\mathsf{STD} = 358.84$), showing how connected code in notebooks is, even though it is less complex.

Comparing with Prior Results: While most of the metrics computed by Grotov et al. [11] (referenced just as "prior results" below) have been nor-

Table 2: Top Style Errors, Notebooks and Scripts.

Error Code	Error Description	Category	Notebooks %	Scripts %
W0611	Import module or variable is not used	Best Practices	41	64
W0621	Redefining name from outer scope	Best Practices	22	27
W0404	Re-imported module	Best Practices	16	13
W0612	Unused variable name	Best Practices	8	14
W0613	Unused argument	Best Practices	4	9
C0411	Import order not followed	Code Style	38	57
C0412	Imports not grouped by package	Code Style	21	19
C0305	Trailing newlines	Code Style	20	15
W0301	Unnecessary semicolon	Code Style	7	4
W0311	Bad indentation	Code Style	5	11
E1101	Variable accessed for nonexistent member	Error Proneness	47	59
E0001	Syntax error	Error Proneness	47	3
W0104	Statement seems to have no effect	Error Proneness	33	6
E0611	No name in module	Error Proneness	29	54
W0106	Expression is assigned to nothing	Error Proneness	9	1

malized to the source lines of code, we can still compare certain metrics as well as general trends. We found that both notebooks and scripts in our data contain more SLOC on average (64 for notebooks, 23 for scripts) than those from the prior results. The results for functions are very similar except for built-in function uses. In our corpus, notebooks have a higher mean number of uses compared to scripts, while the prior results show both having a similar number. Interestingly, our results show both notebooks and scripts having higher cyclomatic complexity than the prior results, although in both cases scripts are more complex than notebooks.

5.2 Evaluation of Style Metrics

Next we compare the style metrics gathered using Hyperstyle. Although the notebooks had a much higher number of source lines of code, they have a much lower average of warnings per file—for notebooks, a mean of 14.21, versus a mean of 23.77 for scripts. In order to evaluate the warnings, we have taken the ones that appear most frequently from three different categories and found the percentage of scripts and notebooks in which they appear in each dataset. The results of this are found in Table 2. Note that the error codes are based on the codes used in Pylint, a popular linting tool for Python.

Of the top five issues in Best Practices, two are related to imports and two are related to variables. For these four, there is a higher frequency of errors in scripts than in notebooks. For both issues with imports, W0611 (import module or variable is not used) and W0404 (re-imported module), it is possible they are caused because of how the code is created, with developers copying code from existing samples and pasting it (including imports) in to

their own code. Error code W0621 (redefining name from outer scope) is the issue of duplicate variable names that refer to values in different scopes, while errors W0612 (unused variable name) and W0613 (unused argument) are related to unused variables and parameters. Again, these issues may be created when code is copied in to a program and then edited, keeping declarations of variables and/or parameters but removing lines of code including the uses.

Of the top five issues in Code Style, this time three are more frequent in notebooks while two are more common in scripts. Also, two of them again have to do with imports, while the other two deal with spacing. Both C0411 (import order not followed) and C0412 (imports not grouped by package) are problems with the ordering of imports. PEP8 has a specific import standard which expects imports to be ordered in a certain way and grouped together. This can be an issue if imports are not added until they are needed, and therefore get added to the bottom of the import list. For spacing, issue C0305 (trailing newlines) and issue W0311 (bad indentation) do not impact correctness, although W0311 could lead to maintenance issues later since bad or irregular indenting can lead to confusion about the expected behavior of the code. The last code style issue is W0301 (unnecessary semicolon). Semicolons are rarely used in Python code, but this could imply that the writer of the file has experience in writing in other languages where semicolons are commonly used.

In the last category, Error Proneness, we once again see that three of the issues are more frequent in notebooks while the other two are more frequent in scripts. The most frequent issue for both sets of files is E1101 (variable accessed for nonexistent member). This error can indicate a bug in the code, but can also indicate that the dependencies that provide this member are not available. This may also account for the high frequency of code E0611 (no name in module). E0001 (syntax error) is the code given for syntax errors. This issue is significantly more frequent in notebooks than scripts. This could be due to how code is executed in specific parts of a notebook, meaning that some cells could contain syntax errors. The following issue, W0104 (statement seems to have no effect) could have a similar problem, where the statement does have an effect but only for the user using the notebook (e.g., to display a value). The last issue is W0106 (expression is assigned to nothing). This only occurs in 1% of the scripts as opposed to 9% of the notebooks.

Comparing with Prior Results: Comparing with the prior results, we share 1 style error in the top 5 errors for the Error-Proneness category, 0 style errors in the Code Style category, and 2 style errors for the Best Practices category. Of the 15 style errors from the prior results, 13 of them appear in a higher percentage of notebooks than scripts, while this is only true for 7 out of the 15 of our style errors. Of the 3 shared errors, the prior results show all 3 appearing in a higher percentage of notebooks, however, in our results, only 1

of them appears in a higher percentage of notebooks. Studying the reason for these differences is part of our future work.

6 Conclusions and Future Work

In this paper, we presented an analysis of Python scripts and Jupyter notebooks, comparing structural and style metrics extracted from both. The results show that Jupyter notebooks are generally larger, but less complex. They tend to use built-in functions much more than scripts and have a higher coupling rate. Scripts averaged more style issues per file than notebooks. For the fifteen issues chosen, the scripts had the issues appear more frequently for eight, while the notebooks had the issues appear more frequently for seven.

For future work, we would like to expand the machine learning corpus to include code from other sources beyond Kaggle (e.g., through detecting use of ML APIs in code on GitHub). We would also like to explore what aspects of the code are leading to the differences between ML code (specifically) and Python code (in general) that we reported in Section 5. Beyond this, we also believe it is important to create new analysis tools, including linters/style checkers, that are specifically aimed at notebooks created for machine learning tasks. These tools could better support both students and practitioners, giving targeted advice that would make sense given the medium (notebooks), the domain, and the domain's structural and style characteristics. Given the results, tools that would help to integrate existing code examples (such as snippets of code from other samples, from API documentation, or from Stack Overflow) seem particularly important. These could help to keep a consistent set of properly-ordered imports while avoiding the re-declaration of existing variables or the inclusion of unused variables and/or code, and could ensure consistent spacing based on current style conventions. Navigation and code inspection tools that can help developers navigate the many API calls they use in notebooks could also be important, especially for newer developers learning how to use machine learning APIs. Finally, tools that can detect unused code in notebooks, and that can illustrate dependencies between code blocks, could be valuable for students trying to understand existing notebooks or developing their own.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant CNS-2050883.

References

[1] Hyperstyle. https://github.com/hyperskill/hyperstyle.

[2] Kaggle. https://www.kaggle.com/.

[3] Matroskin: A library for the large scale analysis of Jupyter notebooks. https://github.com/JetBrains-Research/Matroskin.

[4] Meta Kaggle. https://www.kaggle.com/datasets/kaggle/meta-kaggle.

[5] Project Jupyter. https://jupyter.org/.

[6] Zenodo Artifact for A Comparison of Machine Learning Code Quality in Python Scripts and Jupyter Notebooks. https://dx.doi.org/10.5281/zenodo.8122385.

[7] 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 ICSE-SEIP 2019*, pages 291–300, 2019.

[8] Anastasiia Birillo, Ilya Vlasov, Artyom Burylov, Vitalii Selishchev, Artyom Goncharov, Elena Tikhomirova, Nikolay Vyahhi, and Timofey Bryksin. Hyperstyle: A Tool for Assessing the Code Quality of Solutions to Programming Assignments. In *Proceedings of SIGCSE 2022*, page 307–313. ACM, 2022.

[9] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities. In *Proceedings of CHI 2020*, page 1–12. ACM, 2020.

[10] Thomas Elliott. The State of the Octoverse: Machine Learning, 1 2019. https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/.

[11] Konstantin Grotov, Sergey Titov, Vladimir Sotnikov, Yaroslav Golubev, and Timofey Bryksin. A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts. In *Proceedings of MSR 2022*, page 353–364. ACM, 2022.

[12] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. Managing Messes in Computational Notebooks. In *Proceedings of CHI 2019*, page 1–12. ACM, 2019.

[13] Yuan Jiang, Christian Kästner, and Shurui Zhou. Elevating Jupyter Notebook Maintenance Tooling by Identifying and Extracting Notebook Structures. In *Proceedings of ICSME 2022*. IEEE, 2022.

[14] Mary Beth Kery and Brad A. Myers. Interactions for Untangling Messy History in a Computational Notebook. In *Proceedings of VL/HCC 2018*, pages 147–155, 2018.

[15] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool. In *Proceedings of CHI 2018*, page 1–11. ACM, 2018.

[16] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. Understanding and improving the quality and reproducibility of Jupyter notebooks. *Empirical Software Engineering*, 26(4):1–55, 2021.

[17] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks. In *Proceedings of MSR 2019*, pages 507–517, 2019.

[18] Luigi Quaranta, Fabio Calefato, and Filippo Lanubile. Eliciting Best Practices for Collaboration with Computational Notebooks. *Proc. ACM Hum.-Comput. Interact.*, 6(CSCW1), 4 2022.

[19] Luigi Quaranta, Fabio Calefato, and Filippo Lanubile. Pynblint: a Static Analyzer for Python Jupyter Notebooks. In *Proceedings of CAIN 2022*, pages 48–49. ACM, 2022.

[20] Adam Rule, Ian Drosos, Aurélien Tabard, and James D. Hollan. Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), 11 2018.

[21] Adam Rule, Aurélien Tabard, and James D. Hollan. Exploration and Explanation in Computational Notebooks. In *Proceedings of CHI 2018*, page 1–12. ACM, 2018.

[22] Sergey Titov, Yaroslav Golubev, and Timofey Bryksin. ReSplit: Improving the Structure of Jupyter Notebooks by Re-Splitting Their Cells. In *Proceedings of SANER 2022*, pages 492–496, 2022.

[23] Guido Van Rossum, Barry Warsaw, and Nick Coghlan. PEP 8—Style Guide for Python Code, 2001. `https://peps.python.org/pep-0008/`.

[24] Ashwin Prasad Shivarpatna Venkatesh and Eric Bodden. Automated Cell Header Generator for Jupyter Notebooks. In *Proceedings of AISTA 2021*, page 17–20. ACM, 2021.

[25] Aleksei Vilkomir. *An Empirical Exploration of Python Machine Learning API Usage*. East Carolina University, 2020.

[26] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks. *ACM Trans. Comput.-Hum. Interact.*, 29(2), 1 2022.

[27] Jiawei Wang, Li Li, and Andreas Zeller. Better Code, Better Sharing: On the Need of Analyzing Jupyter Notebooks. In *Proceedings of ICSE-NEIR 2020*, page 53–56. ACM, 2020.

[28] John Wenskovitch, Jian Zhao, Scott Carter, Matthew Cooper, and Chris North. Albireo: An Interactive Tool for Visually Summarizing Computational Notebook Structure. In *Proceedings of VDS 2019*, pages 1–10, 2019.

[29] Chenyang Yang, Shurui Zhou, Jin L.C. Guo, and Christian Kästner. Subtle Bugs Everywhere: Generating Documentation for Data Wrangling Code. In *Proceedings of ASE 2021*, pages 304–316, 2021.

[30] Amy X. Zhang, Michael Muller, and Dakuo Wang. How Do Data Science Workers Collaborate? Roles, Workflows, and Tools. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW1), 5 2020.

The Shrinking Slice of Women and Black Students in a Growing Computer Science Pie: A Preliminary Spatiotemporal Analysis of Longitudinal Completions Data*

Syed Fahad Sultan[1], Chris Alvin[1], Rebecca Drucker[1,2]
[1]Department of Computer Science
Furman University
Greenville, SC 29613
[2]Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
{syedfahad.sultan†, chris.alvin, rebecca.drucker}@furman.edu

Abstract

In an era where Computer Science plays a transformative role in society, the need to address Diversity, Equity, and Inclusion within the field has become increasingly imperative. Using a data-driven approach, this work shows how Women and Black students, as a percentage of total completions of CS related programs, are decreasing in recent years. This trend is particularly alarming in the Southeast region of the United States.

1 Introduction

As the field of computer science continues to innovate and grow, we must also ensure that this progress is inclusive, equitable, and reflects the diverse

†Corresponding author

perspectives and talents of all individuals. Diversity, Equity, and Inclusion (DEI) in Computer Science (CS) is a topic of growing interest in the United States [2, 7, 4]. The National Science Foundation (NSF) has been collecting data on CS completions [3]; a *completion* is defined as an "award or degree conferred." These data are available through the NSF's National Center for Science and Engineering Statistics (NCSES) and is updated annually [8].

In our review of the literature, we found a dearth of DEI studies in CS that take a data-driven approach. Most of the existing work is qualitative and prescriptive in nature [5, 6], without delving into patterns and structures that may be at the root of the problem. We strongly believe that a data-driven approach is necessary to make progress towards a more equitable and inclusive CS community. Without a quantitative understanding of the underlying dynamics, policy makers and practitioners will be unable to make informed decisions about how and where to allocate resources to address the problem.

To address this gap in Computing Education literature, in this work, we use 18 years of Integrated Post-secondary Education Data System (IPEDS) Completions [8] data to explore the question:

How has the number of CS degrees awarded to students from underrepresented groups changed over time in different regions of United States?

Our results show that the number of CS degrees awarded to students from underrepresented groups has generally increased over time. However, the share of CS degrees awarded to students from underrepresented groups is decreasing. Our results uncover various inter- and intra-group trends that are also spatially dependent. The decrease in share of degrees awarded to Women and Black students is particularly stark in the Southeast. In contrast to other underrepresented groups, the number of degrees awarded to Hispanic/Latino students shows a positive trend across different regions of the country.

Altogether, our results strongly suggest a call to action for more data-driven research on the topic of DEI in CS. Only with a better understanding of the underlying dynamics can we hope to make progress towards a more equitable and inclusive CS community.

2 Methodology

2.1 Data

The NSF has been collecting data on completions for all fields of study since 1966 and updates it annually. IPEDS Completions data is available at the institution level. For each institution, the data includes the number of degrees awarded by field and demographic group.

Table 1: Regions reported in IPEDS data (`variable=OBEREG`) ordered by institutional count.

	Region	States	Institutions Count
1	Southeast	AL AR FL GA KY LA MS NC SC TN VA WV	1536
2	Mid East	DE DC MD NJ NY PA	1056
3	Far West	AK CA HI NV OR WA	923
4	Great Lakes	IL IN MI OH WI	893
5	Southwest	AZ NM OK TX	670
6	Plains	IA KS MN MO NE ND SD	497
7	New England	CT ME MA NH RI VT	332
8	Rocky Mountains	CO ID MT UT WY	245

In this work we analyze 18 years (2002-2020) of Completions survey data from 6307 post-secondary institutions. In IPEDS, this survey data table is titled *Awards/degrees conferred by program, award level, race/ethnicity, and gender*. The programs are classified using 6-digit Classification of Instructional Programs (CIP) code. In our results, we refer to all degrees awarded under CIP code 11 titled *Computer and Information Sciences and Support Services* as simply "Computer Science" for brevity and clarity of presentation. Using the IPEDS *Institutional Characteristics* survey of 2020, we identify the region of each institution; categorization details are provided in Table 1 along with a count of the number of constituent regions.

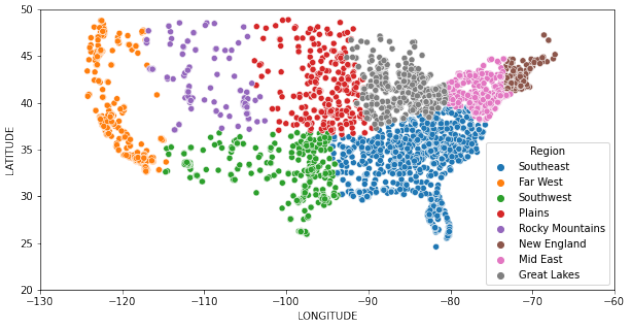
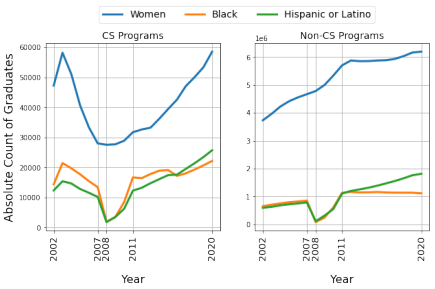
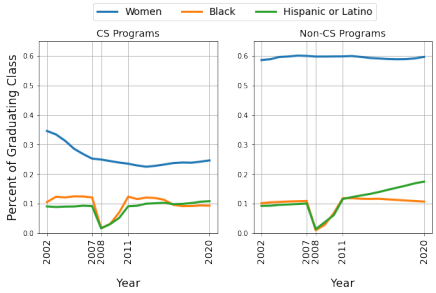


Figure 1: Spatial distribution of institutions in the data, color-coded by region (Alaska and Hawaii are not depicted).



(a) Absolute count of completions; vertical axes use different scales.



(b) Percentage of total completions; vertical axes use the same scale.

Figure 2: Nationwide temporal trends in CS and Non-CS program completions from 2002 to 2020 for Women, Black and Hispanic/Latino students.

2.2 Analysis

Our analysis focuses specifically on the following three underrepresented groups: Women (`variable=CTOTALW`), Black (`variable=CBKAAT`) and Hispanic or Latino (`variable=CHISPT`). All three of these variables are in the data Completions survey table `C<YEAR>`, at the level of institutions, and are publicly available to download from the IPEDS website [8]. Counts were computed using a simple sum of the relevant variable over all institutions for a given year and region. Percentages were computed by dividing relevant sums by `CTOTALT`: Grand total of students awarded degrees with `CIPCODE` in range $[11, 12)$. Data from the IPEDS *Institutional Characteristics* survey table 2020 is used (`variable=OBEREG`) to identify each institution's region. Analysis is done using `pandas` Data Analysis library and plots are made using `matplotlib`. The code is available at `https://github.com/fahadsultan/Education/tree/ccsc`.

3 Results

3.1 Nationwide trends

3.1.1 Absolute count of completions

Post dot-com bubble (2002-2007): In terms of absolute counts (Figure 2b), we see a consistent decline in number of completions in CS between years 2003 and 2008. We believe this is the aftermath of the dot-com bubble burst of 2000 [1]. Note that rising completions from 2002 to 2003 can be explained by the fact that these students *enrolled* before the event. During the same period,

an increase in absolute counts is observed for completions in Non-CS programs across underrepresented groups.

Financial Crisis (2007-2008): We see a sharp decline in completions in 2008 compared to 2007. We believe that this can be explained by dropout rates during the corresponding financial crisis [9]. Notably, this decline is not observed for women in Non-CS programs and in contrast are most acute for Hispanic/Latino and Black students.

Sharp Recovery (2008-2011): During the three years between 2008 and 2011, we observe a sharp recovery across all student groups that underwent a sharp decline between 2007-2008.

Slow Recovery (2011-2020): In terms of absolute counts of completion, in CS as well as in other areas of study, we observe a slow but consistent recovery back to pre-2008 levels. Warranting further investigation, completions for Hispanic/Latino students in Non-CS programs continue to increase while the number for Black students remains the same.

3.1.2 Percentage of total completions

As we observe in Figure 2b, nationwide trends stand in bleak contrast when measured as a percentage of total completions in both CS and Non-CS program completions for underrepresented groups. When we consider percentage of total completions for underrepresented groups, we observe a negative or neutral trend for most underrepresented groups. The only group that exhibits a positive trend is that of Hispanic/Latino students in Non-CS programs.

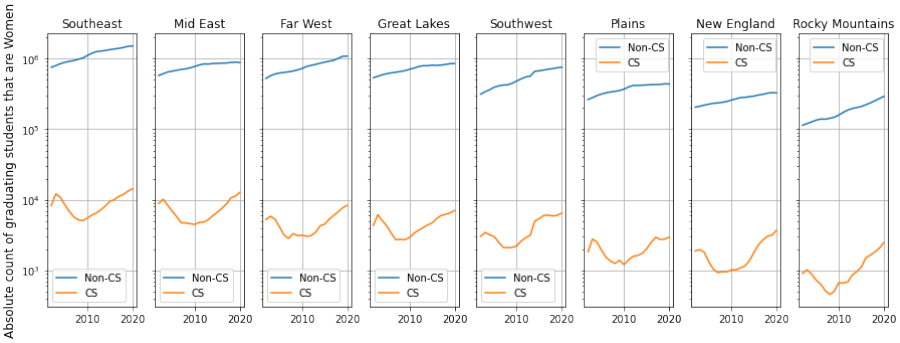
Post dot-com bubble (2002-2007): With the exception of Women in CS, for our three groups, the trend lines are relatively flat during this period.

Financial Crisis (2007-2008): Hispanic/Latino and Black students exhibit a sharp dip in completions in both CS and Non-CS programs. Women in Non-CS programs do not show any change during this period, whereas Women in CS show a less of a dip compared to the counterparts groups in CS.

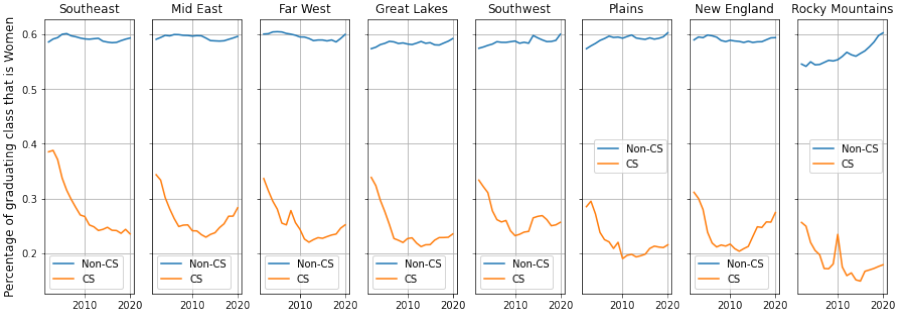
Sharp Recovery (2008-2011): Similar to absolute counts, we observe a sharp V-shaped recovery during 2008-2011 across all student groups that underwent sharp decline in the preceding period.

Lack of Progress (2011-2020): With the exception of Hispanic/Latino students in Non-CS programs, no other groups exhibit a positive trend from 2011-2020.

In summary, while there is a clear increase year to year in absolute counts of CS completions, when we consider the percentage of total completions, Women and Black students show no improvement in recent years.



(a) Absolute count of completions; vertical axes are log-scaled.



(b) Women completions as the percentage of total completions.

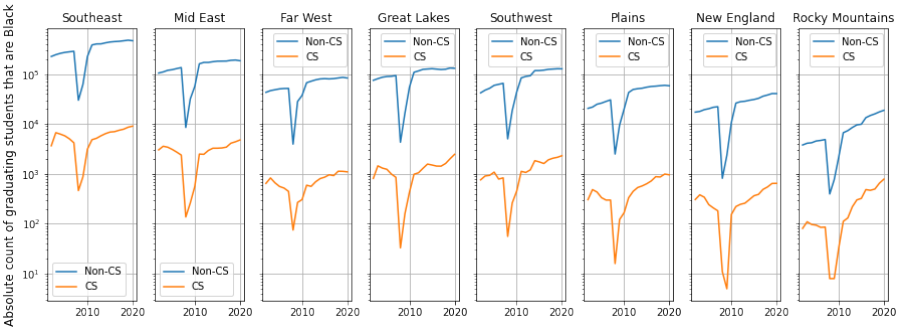
Figure 3: Trends in completion of CS and Non-CS programs by Women from 2002 to 2020 for 8 major regions of the United States.

3.2 Regional trends

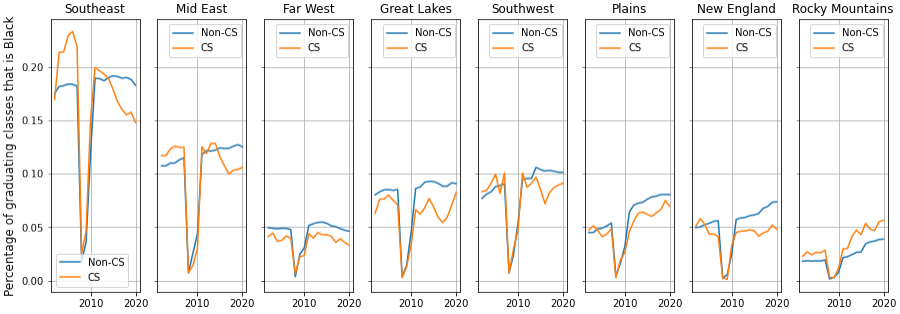
We add a spatial component to our temporal analysis by studying trends for the eight major regions in the United States listed in Table 1.We observe divergent patterns for different student groups across different regions in Figure 3 through Figure 5.

3.2.1 Women students

A deeply noteworthy observation we can make about Figure 3 is the significant differences in trend lines when we compare absolute counts of completions by Women students (Figure 3a) to completions as a percentage of total completions (Figure 3b). Figure 3a shows absolute counts increasing across different



(a) Absolute count of completions; vertical axes are log-scaled.

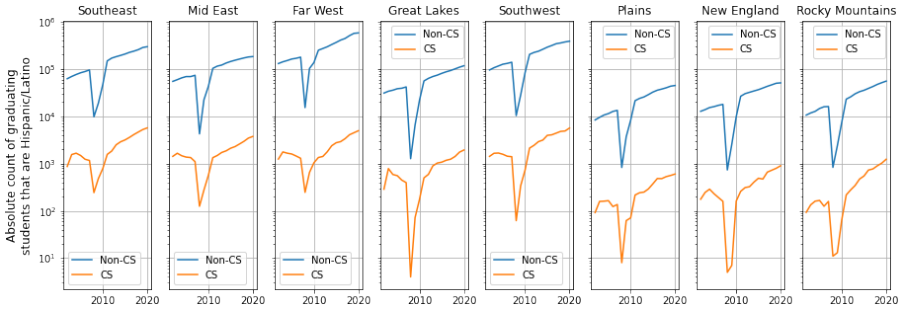


(b) Completions by Black students as the percentage of total completions.

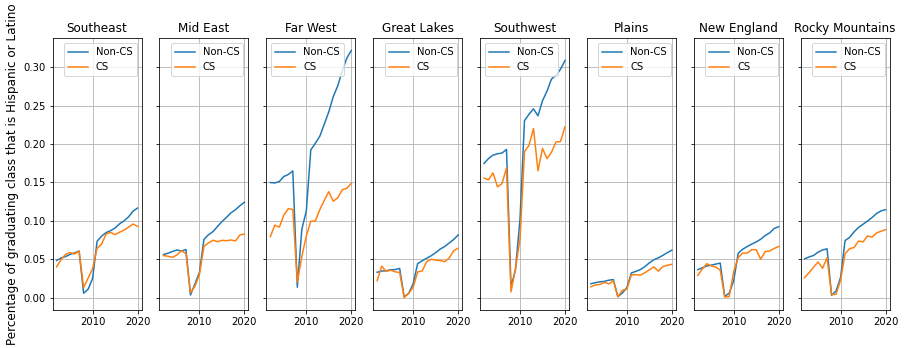
Figure 4: Trends in completion of CS and Non-CS programs by Black students from 2002 to 2020 for 8 major regions of the United States.

regions of the US, at least in recent years. We might conclude that Women in CS evidence an upward trajectory even though they may have a long way to go (note that the vertical axis is log-scale) compared to Non-CS areas of study. However, when we consider percentage of the total completions in CS (Figure 3b), Women's share is decreasing across all regions of the US. For Non-CS, the completion trend lines are either mostly flat or modestly increasing.

This evidence is an example of the phenomenon colloquially described as a *rising tide lifts all boats*. That is, the total CS completions is increasing, including an increase in the number of completions by Women students. However, with each year, the proportion is decreasing for Women in CS. We can thus conclude that while the Computer Science pie is increasing in size, the Women's slice of that pie is decreasing. In Non-CS programs, the trends are modestly positive or flat.



(a) Absolute count of completions; vertical axes are log-scaled.



(b) Completions by Hispanic/Latino students as the percentage of total completions.

Figure 5: Trends in completion of CS and Non-CS programs by Hispanic/Latino students from 2002 to 2020 for 8 major regions of the United States.

Even though the decrease in Women's completions as percentage of total completions is consistent across different regions in the US, the biggest drop is in Southeast. This is of particular concern since the Southeast maintains the most institutions according to IPEDS (Table 1).

3.2.2 Black students

We observe similar trends for Black students that we observed with Women: trend lines for absolute counts are positive (Figure 4a) but trend lines for percentage of total completions are mostly flat or negative (Figure 4b). An alarming result is the sharp decline from 2012 to 2020 (the most recent 8 years in the data) for Black completions in the Southeast while other regions show mixed trends during the same period. This is of tremendous concern since

the Southeast is the region with the greatest number of institutions and where most of the black population is concentrated in the United States, as per 2020 census Data, Summary File 1.

3.2.3 Hispanic/Latino students

Trend lines observed for Hispanic/Latino students stand in positive contrast to those observed for Women and Black students. Unlike other underrepresented groups in our analysis, Hispanic/Latino students show a positive trend in both absolute counts as well as percentage of total completions. These positive trends are consistent across different regions in the US. However, the rate of improvement in CS is not keeping up with the rate of improvement in Non-CS programs (Figure 5b). This trend is most visible in Far West and Southwest regions. These regions are also where most of the Hispanic/Latino populations are concentrated in the United States, as per 2020 census data, Summary File 1.

Not only are more Hispanic/Latino students graduating in all areas of study in the entire country but they are also making up a greater percentage of total graduates in their respective areas. This trend can also be viewed in aggregate in Figure 2. The trends for Hispanic/Latino students are to be celebrated and need to be investigated further for potential causes, so effective policy interventions can be designed to replicate this trend for Women and Black students.

4 Conclusions

From our preliminary spatiotemporal analysis of longitudinal completion data, we can conclude the following:

1. While absolute counts of Women, Black and Hispanic/Latino students are increasing in Computer Science, for Women and Black students, it is due to a general increase in Computer Science students completing the program; i.e., a general increase in size of the Computer Science pie.

2. As percentages of total completions of CS programs, Women and Black students have consistently been decreasing in the recent 5-8 years, across all regions of the United States, but particularly in the Southeast.

3. The only underrepresented group that exhibits robust improvements in completion of Computer Science across different regions of the country is that of Hispanic/Latino students.

Taken together, the results presented in this paper serve as a call to action for the CS Education community to intentionally include more data-driven analyses and approaches to the problem of DEI in CS. The IPEDS data is a rich source of information that can be used to better understand the underlying dynamics plaguing the CS community.

References

[1] Milam Aiken, Bart Garner, Kaushik Ghosh, and Mahesh Vanjani. Dot. com boom and bust effects on mis college enrollments: 1995–2006. *Communications of the IIMA*, 8(1):4, 2008.

[2] Jill Denner and Shannon Campe. Equity and inclusion in computer science education: Research on challenges and opportunities. *Computer Science Education: Perspectives on Teaching and Learning in School*, page 85, 2023.

[3] Sarah Krichels Goan and Alisa F Cunningham. Degree completions in areas of national need, 1996-97 and 2001-tab. nces 2006-154. *National Center for Education Statistics*, 2006.

[4] Nwannediya Ada Ibe, Rebecca Howsmon, Lauren Penney, Nathaniel Granor, Leigh Ann DeLyser, and Kevin Wang. Reflections of a diversity, equity, and inclusion working group based on data from a national cs education program. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 711–716, 2018.

[5] Anagha Kulkarni, Ilmi Yoon, Pleuni S Pennings, Kazunori Okada, and Carmen Domingo. Promoting diversity in computing. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 236–241, 2018.

[6] Elizabeth A Larsen and Margaret L Stubbs. Increasing diversity in computer science: Acknowledging, yet moving beyond, gender. *Journal of women and minorities in science and engineering*, 11(2), 2005.

[7] Linda J Sax, Kathleen J Lehman, Jerry A Jacobs, M Allison Kanny, Gloria Lim, Laura Monje-Paulson, and Hilary B Zimmerman. Anatomy of an enduring gender gap: The evolution of women's participation in computer science. *The Journal of Higher Education*, 88(2):258–293, 2017.

[8] NC Statistics. Integrated postsecondary education data system. 2012. `https://nces.ed.gov/ipeds/`.

[9] Ralph Stinebrickner and Todd Stinebrickner. The effect of credit constraints on the college drop-out decision: A direct approach using a new panel study. *American Economic Review*, 98(5):2163–2184, 2008.

Do We Need to Write? Researching Perceptions of Disciplinary Writing Importance and Skills in an Advanced Computer Science Course*

Elizabeth von Briesen
Department of Computer Science
Elon University
Elon, NC 27244
evonbriesen@elon.edu

Abstract

Our research explores perceptions related to writing in the computer science discipline. It is a common misconception that this skill is not important in the field, and we are motivated to dispel that notion and assist students in gaining experience and confidence in their disciplinary writing skills. To that end, we surveyed undergraduate students at the start and end of term in our Artificial Intelligence course, an advanced computer science elective. Students wrote two blogs-like items, one each for audiences with and without technical knowledge of the field, and also produced technical documentation related to two programming assignments. We found that on average, students agreed that writing in the discipline is important, and that they have some confidence in their writing abilities across audiences. While we did not find a statistically significant difference between perceptions at the start and end of the term, our overall results and open-ended feedback indicate that students find writing in the field to be important, and that there is strong interest in further curricular enhancements in this area.

1 Introduction

"*Who wrote this*, you wonder. With a sinking feeling, you realize you're looking at your own code from fourteen months ago, and you've forgotten almost everything about it" [3, p. xxiii]. Bhatti et al., authors of "Docs for Developers: An Engineer's Field Guide to Technical Writing," address a major issue in software development: the failure to consistently write appropriate documentation. It is sometimes assumed by people, both within and outside the field, that computer scientists neither like nor *need* to write. The above quote is only one example of the potential consequences of failing to prioritize the writing process in this discipline. Beaubouef and McDowell highlight that in addition to traditional code documentation and manuals, computer scientists must communicate with other professionals in the field, and most importantly, with customers and clients [2].

This study is motivated by two areas of interest in undergraduate computer science education: perceptions of the importance of, and preparation for, writing in the discipline. The primary question we are seeking to answer through this research is: *Can disciplinary writing assignments in an advanced computer science course impact students' attitudes toward the value of being a strong writer in the field and improve their perceptions of the effectiveness of their writing?*

Undergraduate students are building a foundation that will support their future work, and for computer science students, technical skills are essential to the strength of that base. However, as Bhatti et al. suggest, those technical skills alone are often insufficient when viewed in the larger context of the profession [3]. The ACM's Computing Curricula 2020 includes a variety of writing and communication competencies, which we detail in Section 2 [1]. While their total number is far fewer than those of technical competencies, their inclusion signifies an importance in the development of future computer scientists.

We conducted this research at an undergraduate, liberal arts, teaching university. In addition to their rich experience in writing in the liberal arts, students have also benefited from extensive enhancements to writing instruction across all departments achieved through the a recent QEP project. As such, the university is very supportive of writing in any discipline, and many students have broad writing experience.

To answer our research question, we administered surveys at the start and end of the term (referred to as pre- and post- from here), and integrated writing assignments designed to be similar to writing students may encounter in their future work. These assignments ranged from a blog-like post for an audience without technical knowledge of the CS field, to a project README file evaluated by their peers. While we did not see a significant change in student perceptions of the importance of writing in the profession, or in student

confidence in their own writing skills, we found that average perceptions for each was weighted toward agreement. We also received informative open-ended feedback. The above, along with three individual case studies, are detailed in Section 4. We are encouraged by our findings that our students see value in developing their disciplinary writing skills, and are looking forward to continued analysis and development of future research in this area.

2 Related Work

The ACM's Computing Curricula 2020 includes competencies that reflect the importance of communication skills in the discipline. Included in the 84 Computer Science Draft Competencies are those that recognize the need for documentation (HCI-E, SDF-E), reports (PL-B), and other types of communication that address industry trends (SP-C) and the impact of technology on society (SP-E). The report's 88 Information Systems Draft Competencies include 7 specific to communication, with 2 focused on written communication (58, 59) [1, pp. 111–117]. It is clear from the inclusion of these competencies in the ACM report that they are important to computer science education.

Faculty have varied perceptions of the importance of integrating writing into their undergraduate STEM courses. Some see it as a skill unrelated to the scientific content of their courses, while others find it an essential component of their own work as scientists and educators [9]. In computer science, researchers have made efforts to introduce writing of different forms into standard computer science courses [4, 5, 6, 7, 12, 13]. They have also examined commonly seen issues in student writing [10], how writing centers can address industry needs through supporting student development in this area [11], and how to better manage grading and assessment of writing assignments [8]. This research is motivated by the above work, and our methodology is partly informed by Grubb's work with blogs in the CS classroom [6], and Digh's communication research in an Artificial Intelligence course [5].

3 Methods

We conducted this research in the computer science department's Artificial Intelligence course, which is an advanced elective for CS majors and minors. This class had 25 students, with one professor and no teaching assistant. To answer our research question, we collected quantitative and qualitative data through pre- and post-survey instruments, and assigned a variety of writing tasks common in the discipline. All enrolled students consented to the collection of the survey data as well as samples of their writing from relevant assignments, with one student withdrawing consent.

3.1 Pre- and Post-Survey Instruments

Participants completed surveys with more than 30 questions. The pre- and post-surveys were identical except for one additional question on the post-survey about interest in follow-up interviews. We collected details about student majors, minors, internships and research experience, and demographics. The survey was a combination of Likert-scale, multiple answer, and open-response questions. These surveys collected data about student writing interests, writing experience within and outside the classroom, perceptions of the importance of writing in the discipline, perceptions of the types of writing common in CS, and interest in further development of disciplinary writing skills.

3.2 Disciplinary Writing Assignments

Students completed four assignments with writing components aligned with types of writing common in the field. Two were low-stakes blog-like posts, and two were reports and documentation produced in conjunction with programming assignments. For each writing task, we provided students with guidance and examples of that genre from a variety of resources. For example, we developed a resource including articles about, and open-source repositories of, design documentation for the associated assignment. In addition to instructor evaluation, every assignment with a writing component included time for giving and receiving peer feedback. This approach was intended to give students the opportunity to practice and improve their disciplinary writing, expose them to different genres and audiences common in the field, and communicate with their peers about the writing process. The assignment design was as follows:

Blog Posts–Low-Stakes[1]

1. **Audience Without Technical Knowledge of CS**[2]**:** Write about an AI technology of your choice and its impact on society.
2. **Audience With Technical Knowledge of CS:** Write about an ethical issue or issues in a specific use of AI, include connections with at least one other field of study.

Documentation–High-Stakes

[1]These assignments were very open-ended with respect to style, including a zine format if desired.

[2]In this paper, this audience is sometimes referred to as "non-technical" and includes lay audiences, and non-CS experts and professionals. The other blog post audience type is sometimes referred to as "technical."

1. **Design Document and Code Readability:** For a coding assignment implementing and comparing algorithmic solutions to the 8-Queens problem, write design documentation for an audience of fellow students that includes an overview of the problem, a design plan, and discussion of alternative approaches. Additionally, review and implement best-practices for code commenting, variable names, overall organization, and readability.
2. **README File and Project Report:** For a self-selected implementation of an AI task, write a README file for a peer briefly describing the project, providing instructions for running the code, and reviewing sample use cases and results. Also produce a report detailing the project topic, the selected algorithm or algorithms implemented, and an analysis of the quality of the solution.

4 Results and Discussion

We are particularly interested in determining if there were changes in student perceptions of the importance of writing in the discipline, and in their own level of confidence when writing for different audiences. To this end, we include here results from 3 survey questions and detail 3 representative case studies.

All 25 enrolled students completed the pre-survey, and 18 completed the post-survey as well. All results presented here, including case studies and examples of open-ended feedback, are from the 18 students who completed both surveys.

4.1 Select Survey Questions

Figure 1 visualizes the pre- and post-average Likert scores for three survey questions asking how strongly students agreed or disagreed (Strongly Agree (5) to Strongly Disagree(1)) with the following statements:

Q1: Good writing skills are important in the computer science profession.
Q2: I can write effectively about computer science for a **technical audience*** in the field.
Q3: I can write effectively about computer science for an audience **without technical knowledge*** of the field (e.g. lay audiences, non-CS experts and professionals).

For all questions, and in both the pre- and post-surveys, the average score was above 3 (Neither Agree nor Disagree). This indicates that preferences were weighted toward agreement with the statements. While we did find an increase

*Bolding included in original survey question for clarity

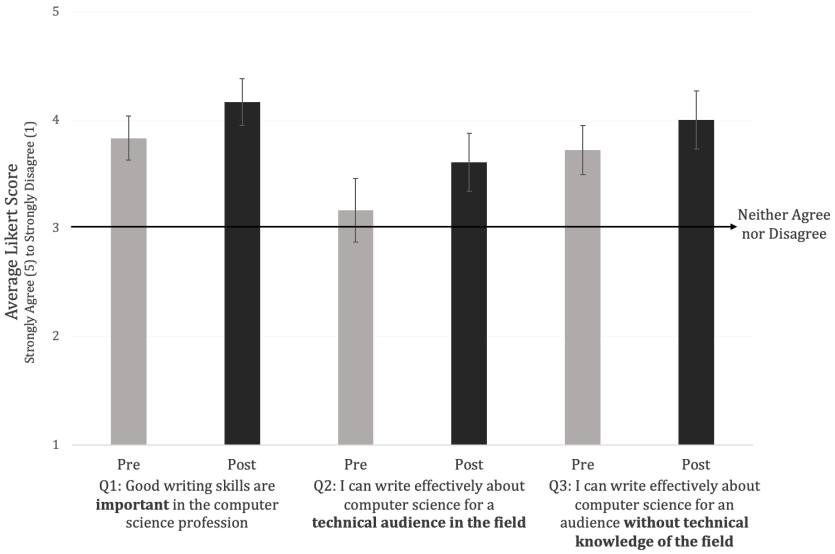


Figure 1: Change in Average Student Perceptions in Pre- and Post-Survey Questions

in all averages in the post-survey results, it was not statistically significant.

Q1 Discussion: As noted in Section 1, the importance of writing in the CS profession is sometimes underestimated [2, 3]. We were encouraged to find that, on average, students agreed that good writing skills are important in the field, with the post-survey result exceeding the "Somewhat Agree" level (4). When asked for general comments about writing in the computer science profession in the post-survey, one of the students noted that *"I think it is a vital skill if you want to have a profession in computer science,"* and another stated *"I think it is useful because you will encounter many people that do not have a computer science background and you need to explain complex topics to them."* These results indicate that our computer science students appreciate the value of writing in the field, which is important as a motivator for skill development.

Q2 & Q3 Discussion: These questions explored students' perceptions of their writing ability for audiences with and without technical knowledge of the field. Again, we found that the average response was weighted toward agreement with the statements, indicating that students felt some degree of confidence when writing for these audiences. When asked in the post-survey about the types of

writing in computer science they were interested in developing, 11 comments communicated interest in writing for a technical audience, such as: *"I am interested in developing writing that can be used in project proposals that convey the details and need for a computer science project."* 4 comments indicated interest in writing for a non-technical audience including: *"Writing for non-technical audiences, especially presentations, memos, and other professional forms of communication"*, and *"I want to write more blog posts describing complex topics to a non-technical audience."*. These results show an overall positive view of disciplinary writing for different audiences and interest in further development of these skills.

4.2 Case Studies

Here we present three participant cases. Table 1 shows the change in their Likert scores for the selected questions, and for context we include samples of open-ended feedback each participant provided in the surveys.

Table 1: Individual Responses to Select Survey Questions: Strongly Agree(5) to Strongly Disagree(1)

Participant #	Q1 Pre/Post	Q2 Pre/Post	Q3 Pre/Post
4	4/5	2/4	4/5
7	3/5	2/4	4/4
13	5/5	4/4	2/4

Case 1 - Participant 4: This participant was a junior, double majoring in accounting and computer science. They had a CS internship prior to taking this course and noted that writing was not a significant part of that experience. They commented in the pre-survey that *"An important part of my planned career is communicating to non-technical people why they should care about technical risks,"* and they seemed to have confidence in their skills in this area (Q3 responses). They also commented that *"Doing blog posts challenged me to think about how to make CS topics relevant to non-CS people. This was a really interesting challenge."* Of additional note here is the 2-point increase in agreement for Q2, showing that this student gained confidence in their ability to write for a technical audience. When asked about examples of writing for this audience type during their studies, they stated that *"The second blog post and the documentation I've done in Dr. von Briesen's class are the main example I can think of CS writing I've done for a technical audience at Elon."*

Case 2 - Participant 7: This participant was a graduating senior and computer science major. He had a CS internship prior to taking this course, but did not comment on any writing he did in that setting. This student represents a participant who increased 2 points in his perception of the importance of writing in CS (Q1), and also in his confidence when writing for a technical audience (Q2). In the post-survey, when asked how much he enjoys writing in a professional setting, he stated that *"I would much rather work on other types of projects than writing. Simply not an engaging task to me."* However, he also commented that he would be interested in developing his *"Documentation. Technical to non technical"* skills.

Case 3 - Participant 13: This participant was a graduating senior and computer science major. He also had a CS internship prior to the course, and this appears to have influenced his view of the importance of writing in the profession. He stated in the pre-survey that *"From my experience of Internship in a professional setting I was always told to write documentation as simple to understand as possible so anyone could look at it and have some understanding as to what is going on."* Additionally, this student's perceptions for Q3 increased by 2 points, indicating increased confidence when writing for a non-technical audience. Overall, his open-ended responses conveyed a strong interest in improving his writing for a technical audience, and we find it interesting that it was instead his confidence in his ability to write effectively for a non-technical audience that increased on the Likert scale.

Discussion:

These three cases represent a variety of perceptions and perspectives from our class. Both participants 4 and 7 felt more confident in their writing for a technical audience at the end of the term, and all three expressed interest in improving their writing for a technical audience in the field. Interestingly, only participant 13 reported an emphasis on writing in his internship, although several other students commented positively on writing in internships during class discussions (10 of 25 students had completed a CS internship). The above gives us confidence that additional research and work to integrate writing that is common in the field into our courses will be well received by most students. We did not receive any open-ended feedback from students stating that the writing tasks were not appropriate or useful, and in fact, 7 of 18 participants expressed a desire for more opportunities to write in these modalities in our computer science courses.

To close this section, we would like to note that it is likely there is some bias in these results due to the nature of the study. Classroom research about disciplinary writing clearly shows the interest of the professor in the topic,

which may then influence students' perceptions, even at the beginning of the semester. Additionally, it is possible that the level of confidence our students had in their writing ability is a result of their broader academic training at our liberal arts institution, as noted in Section 1. This may yield stronger support and appreciation for these types of activities in the CS classroom than at other types of institutions with fewer general writing requirements for CS majors.

5 Conclusions and Future Work

In this paper, we presented the methodology and preliminary results from our research study exploring changes in student perceptions toward the value of writing in the discipline of computer science, and in their confidence in the effectiveness of their writing in the field. Our methodology included pre- and post-surveys, along with multiple disciplinary writing assignments. We found that on average, students scores were weighted toward agreement that good writing skills are important in computer science, and that they have confidence in their abilities to write effectively for technical and non-technical audiences. We did not find a significant change in their perceptions between the pre- and post-survey. Nevertheless, we are optimistic that a continuation of this work will be well received by our students, and their feedback and writing artifacts will be instrumental in improving our efforts to integrate writing into our courses. Our future work in this area will include continued analysis of our quantitative and qualitative results, development of subsequent research studies in both introductory and advanced courses, and appropriate modifications to our approach to explore the use of generative AI models to assist in the writing process.

Acknowledgements

We thank the Center for Writing Excellence at Elon University for providing the Research into Writing Grant to support this research. Thanks also to Alana Evora for her assistance in refining survey questions.

References

[1] Association for Computing Machinery (ACM) and IEEE Computer Society (IEEE-CS). Computing Curricula 2020: Paradigms for Global Computing Education. Technical report, 2020.

[2] Theresa Beaubouef and Patrick McDowell. Computer science: student myths and misconceptions. *Journal of Computing Sciences in Colleges*, 23(6):43–48, 2008.

[3] Jared Bhatti, Zachary Sarah Corleissen, Jen Lambourne, David Nunez, and Heidi Waterhouse. *Docs for Developers: An Engineer's Field Guide to Technical Writing*. Apress, 1 edition, 10 2021.

[4] Charmain B. Cilliers. Student perception of academic writing skills activities in a traditional programming course. *Computers & Education*, 58(4):1028–1041, 5 2012.

[5] Andy D. Digh. Writing and speech instruction in an introductory artificial intelligence course. *Journal of Computing Sciences in Colleges*, 36(5):119–128, 2021.

[6] Alicia M. Grubb. Reflections on Course Blogs in First-Year CS. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–10, 2020.

[7] Nicole Herbert, Kristy de Salas, Tina Acuña, and Erik Wapstra. A methodology to integrate professional skill development throughout an ICT curriculum. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 280–286, 2020.

[8] Grace M. Mirsky. Effectiveness of specifications grading in teaching technical writing to computer science students. *Journal of Computing Sciences in Colleges*, 34(1):104–110, 2018.

[9] Alena Moon, Anne Ruggles Gere, and Ginger V Shultz. Writing in the STEM classroom: Faculty conceptions of writing and its role in the undergraduate classroom. *Science Education*, 102(5):1007–1028, 2018.

[10] Rehmat Munir, Francesco Strafforello, Niveditha Kani, Michael Kaler, Bogdan Simion, and Lisa Zhang. Exploring Common Writing Issues in Upper-Year Computer Science. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 161–167, 2022.

[11] Krista Speicher Sarraf and Ben Rafoth. Don't Forget the End User. *The Writing Center Journal*, 38(1/2):131–164, 2020.

[12] Alistair Willis, Patricia Charlton, and Tony Hirst. Developing students' written communication skills with Jupyter notebooks. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 1089–1095, 2020.

[13] Lisa Zhang, Bogdan Simion, Michael Kaler, Amna Liaqat, Daniel Dick, Andi Bergen, Michael Miljanovic, and Andrew Petersen. Embedding and Scaling Writing Instruction Across First-and Second-Year Computer Science Courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 610–616, 2023.

Explicitly Characterizing Team Structures in Teaching Team-based Project Courses*

Mingxian Jin
Department of Mathematics and Computer Science
Fayetteville State University
Fayetteville, NC 28301
mjin@uncfsu.edu

Abstract

Teamwork has become increasingly vital in computer science programs. While student teams in project courses offer numerous benefits, they also face many challenges. Our focus is to ensure effective team functioning and foster positive team experiences among our diverse student population. In this paper, we present a strategy that explicitly characterizes team structures by appointing student team leaders to make team hierarchy while other teams without hierarchy may still work democratically. By clearly defining the roles and responsibilities of each team member, students gain a better understanding of their individual tasks within the project. This approach provides strong students with opportunities to develop leadership skills while facilitating weaker students in receiving peer assistance, thereby improving their engagement and chances of project success. Preliminary assessment data is included, along with positive student feedback, highlighting the effectiveness of this approach.

1 Introduction

Teamwork has become increasingly vital in computer science programs. In a team-based project course, students obtain valuable team experiences in collaboration, conflict resolution, and group dynamics management. These skills

are highly transferable and prepare students for successful careers in industry or academia.

While student teams in computer science project-based courses offer numerous benefits, they also face common challenges during the team formation process [3, 4, 5, 6, 11]. These challenges include:

- Skill imbalance: Team members with unequal skill levels can make frustration and hinder project progress. Strong students may hesitate to work with weaker members, leading to exclusion.
- Communication issues: Ambiguity, misunderstanding, and unresolved issues can impede collaboration. Communication styles, language barriers, and scheduling conflicts may further hinder effective teamwork.
- Varying commitment levels: Team members may have different levels of motivation and engagement, causing imbalances in effort and potential conflicts within the team.
- Unequal workload: Unequal distribution of workload among team members can create tension and resentment. Without clear expectations for individual contributions, students can be confused about their task assignments.
- Time management: Managing project timelines and deadlines can be challenging when team members have conflicting schedules or varying availability. Poor time management can lead to rushed work, compromised quality, or missed deadlines.

With the rapid growth of computer science enrollments, student diversity in backgrounds, expertise, and perspectives has also increased. While this diversity can enhance problem-solving and innovation, it further exacerbates the above problems in teaching team project-based courses. In teaching-oriented institutions like ours, where there is no graduate program or teaching assistants, the instructor bears the sole responsibility of delivering course materials (excluding the capstone project course which will be explained later) and managing all teams in a way of micromanagement.

Our primary focus is to ensure effective team functioning and positive team experiences for our diverse student population. Additionally, we face the challenges of managing teams, monitoring project progress, and accurately evaluating individual performance, especially as class sizes continue to grow. Student teams typically last for a semester long. Voluntary team formation often results in strong students naturally forming teams together, leaving weaker students to be assigned or hastily grouped, leading to potential team dysfunction. When teams consist of students with significant skill gaps, the above problems are unavoidable. Consequently, these issues result in undesirable learning outcomes, especially for weaker students who would lose interest in project work, ultimately leading to project failure at the end.

To address these challenges, we have implemented changes in team formation and management. We encourage strong students to form teams with weaker students and explicitly define team structures, similar to industry practices, including the roles and responsibilities of each team member. With the agreement of all team members, a team leader is appointed, who takes responsibility for managing the team and may receive a higher grade for project assignments than other team members. These changes provide strong students with opportunities to develop leadership skills and enable weaker students to learn from their peers. Teams become easier to manage, and individual student performance can be more accurately evaluated.

In this paper, we present this new strategy of explicitly characterizing team structures in team-based project courses. Such a course can be a junior/senior level software engineering course or a senior capstone project course. Section 2 reviews related work by other researchers. Section 3 provides a description of the proposed strategy in detail. Section 4 discusses its implementation in course design. Section 5 presents the preliminary data of learning outcomes and includes student feedback. Finally, Section 6 summarizes our work, and suggests future improvements.

2 Related Work

In [1], ABET recognizes the significance of developing students' teaming skills in computer science curricula. One of the six major program learning outcomes identified by ABET is "Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline." ABET emphasizes the need for students to not only work as team members but also as team leaders in their course activities.

Numerous researchers have explored effective team formation, team performance evaluation, and related factors in software engineering and other project-based courses. A snapshot of their work is provided in this section.

Presler-Marshall et al. in [7] conducted interviews with students to summarize factors that contribute to team success or failure. They found the common problems include communicating, setting and adhering to deadlines, and effectively identifying tasks. Team requests cannot predict a team success or dysfunction. Dulanga in [2] describes the skillset to be a good team player from the industry perspective, which includes technical skills, experience, and soft skills. The author points out that, with effective communication tools in place, clear roles and responsibilities of each team member are critical.

To form more effectively functioning teams, various approaches have been experimented. Walker et al. in [11] adopted a special approach in making student teams consisting of both traditional and non-traditional students, allow-

ing them to learn from each other's diverse background across multiple courses. Perez et al in [5] formed small heterogeneous teams where students performed different tasks and faced different challenges through role rotations and documentation transfers. Iacob et al. in [3] presented a mentorship scheme as part of a software engineering course, in which selected undergraduate students with certain experience as professional software engineers served as mentors of student teams. They found that this mentorship approach helped students perform better in teams led by mentors.

Studies on assessment and surveys have further helped our understanding of important factors that make student teams successful. Sims-Knight et al. in [9] developed a self-report assessment tool to evaluate students' teaming practices and provide guidance for improvement. Presler-Marshall et al. in [6] proposed a set of weekly surveys to identify struggling teams, enabling early proactive intervention by teaching staff. Moreover, Oakley et al in [4] describes a guide to the effective design and management of team assignments in a college classroom which includes using various forms and peer ratings to interact frequently with team members. Tafliovich et al. in [10] investigated student preferences in evaluations and provided useful insights to tailor questions for student self and peer evaluations.

In most cases, as in our previous practice, student teams are not formally structured nor explicit guidelines for team structures are given. We decided to make changes by explicitly characterizing team structures and clarifying the roles and responsibilities of each team member, as discussed in the next section.

3 Strategy of Characterizing Team Structures

According to Scratch [8], there are three basic team structures in software engineering:

1. Chief programmer team structure – The chief programmer serves the key person who oversees all team activities, from technical tasks to managerial issues. This individual possesses sufficient knowledge to develop a project plan, assign tasks, and not only writes their own code but also assists others in completing their tasks. This team structure places exceptional expectations on the chief programmer, and the project success heavily relies on the individual.

2. Democratic team structure – In this structure, all team members play equal roles in the development process. There is no hierarchical relationship, and each member takes responsibility for their own code. The team views the entire project as a collective effort, and bugs are shared responsibilities. Team members collaborate in planning the project and

contribute equally to its success. Typically, it requires team members to have similar levels of technical skills and collaborative skills.

3. Hybrid team structure – This team structure combines the advantages and disadvantages of the previous two approaches. It separates technical and managerial responsibilities, with project leader handling technical issues and a project manager managing managerial aspects.

In our practice, since students are working as a relatively small group, typically three members but occasionally two or four, we slightly modify the above team structures to simplify them into two types:

1. With a Team Leader: An individual student is appointed as the team leader, who takes the responsibility for managing project timelines, assigning tasks to other team members, scheduling/presiding team meetings, monitoring each member's work progress to report to the instructor, and providing assistance when needed.

2. Without a Team Leader: The team of this type operates in a democratic manner without a designated leader. All members are expected to make equal contributions to the project and share the same responsibility for its progress. Grades for project assignments are distributed equally among all team members.

The purpose of this change is to mitigate the aforementioned problems and to create more efficient student teams working on a semester-long project. We believe that:

- Strong students not only learn from working on the project but also develop their leadership abilities in a team environment. By assuming the role of a team leader or chief programmer, these students are responsible for project management tasks such as task scheduling and assignment. They can also learn better from helping others.

- Weaker students receive more direct help from their peers. With clearly defined roles and task assignments, their responsibilities are clearer. This has shown noticeable improvements in their engagement and the likelihood of their project failure is decreased significantly.

- For other students who work in a democratic team or without a designated leader have reported greater enjoyment during their work, since they have asserted the initial characterization process and feel more confident about their development process.

4 Implementation in Course Design

Our curriculum includes two required team project-based courses: software engineering and senior (capstone) project. Unlike some other institutions, the capstone project course is not a follow-up to the software engineering course due to resource restrictions. Instead, students choose different project topics from a wide range of research areas representing other faculty's specialities within the department. While a team can select a faculty member as their project advisor, all teams are managed (and/or advised) by the instructor.

In both courses, students are required to introduce themselves at the beginning of the semester, providing individual background, project interests, collaborative personality, availability during the semester, and expectations for their team partners. When forming teams, students' preferences are considered, but strong students are encouraged or sometimes assigned to partner with weaker students, who may be appointed as a team leader. After a brief training on industry team structures, each team is required to reach an agreement on their team structure type and document it for the instructor's reference.

In the software engineering course, the entire class works on the same software development project but in different teams. Students begin working on the project typically from the third or fourth week after forming teams during the initial weeks, concurrently covering fundamental concepts throughout the semester. The instructor provides the project description. The project consists of three submissions representing different stages of software development: requirements/analysis, design, and implementation. At the end of the course, each team presents their work in class.

For teams with a team leader, the team leader takes on responsibilities such as scheduling team meetings, assigning tasks to members, providing overall project architecture design, and managing integration testing during the implementation stage. In teams without a team leader, all members share the workload based on their assigned tasks, which may shift over time and in different cases.

In the senior capstone project course, teams work on different projects selected from an established project pool representing other faculty's specialities, with priority given to grant-supported projects. Students usually complete the team formation process within the first two weeks to allow sufficient time to work on their projects, which often involve learning brand new areas. Project deliverables include a project proposal, midterm and final reports, and a group weekly blog documenting all team activities. At the end of the course, a project presentation is given, with all department faculty invited to participate in the evaluation.

For teams with a team leader in the capstone project course, the team leader assumes a key role in the project. This individual creates the project

plan, assigns tasks to members, sets up team meetings, and manages other team activities. Other members take on tasks assigned by the leader and report their progress. While documentation is not as formalized as in the software engineering course, students are reminded of the importance of documentation through team blogs or group journals. Sometimes a team leader is advised to designate a member specifically responsible for documentation. In some cases, a team leader may focus solely on the technical aspects while assigning managerial roles (such as blogging or meeting recordings) to another team member. This approach resembles the hybrid team structure described in Section 3. Grading each team member's project should reflect their individual contributions with this context.

For a team without a team leader, all members are expected to share the work, similar to the software engineering course. In this type of team, each team member must contribute to the group journal to reflect their individual contributions.

If a team structure is not working out after a few weeks, a democratic team may request to be converted into a leader team. In such cases, the team can meet with the instructor to review their activities, evaluate their progress, and make changes to the team structure if necessary.

5 Preliminary Data and Student Feedback

By utilizing this new strategy to work on a semester long project, we have noticed a positive impact on student performance. Students have shown improved efficiency in team working environments. Strong students have developed leadership skills in addition to achieving better learning outcomes from helping others. Weaker students have become more engaged in all team activities as they may receive quicker help from peers when needed, leading to a decreased likelihood of project failure.

To assess the effectiveness of this approach, we have collected assessment data from three software engineering classes in Table 1. The assessment form is similar to our report for ABET accreditation in PLO 6: Students can work efficiently as a team member or a team leader in a team environment. We have data from the semester before implementing this approach (shown in regular font) and data from the semester afterwards (shown in bold). The data indicates improved passing rates and a decrease in the failing rate.

We did not collect data from the senior capstone course, due to the fact that the course is offered twice a year so the class size is small, making the data less meaningful. However, we can provide some quotes from students' feedback before and after implementing this approach.

Before, more like these –

Table 1: Team Assessment Data for Software Engineering

Semester	# of Teams	Good= exceeds+ meets	Proficient= meets+ marginal	Pass= e+ m+m	Fail= doesn't meet+ not attempt
F20	9	77.78%	40.74%	91.36%	8.64%
F21	7	70%	35%	90%	10%
F22	**8**	**86.67%**	**58.6%**	**94.67%**	**5.33%**

- *"Difficult to work with partners who are not on the same technical level as you and it is difficult managing between teaching them basics and putting together a complex project. Why should I teach them?"*

- *"My teammates brought little contributions to the project. I wish they would have put the same amount of effort into the project as I did. ...They really didn't attempt to do anything until the last couple weeks of the semester and in doing so forced me to finish the project in a rushed manner, that made our project look worse than it could have been."*

After, more like these (clearly, some from a leader team and some from a democratic team)–

- *"She was excellent and knowledgeable on the project. She's a pro. She was our leader! I am glad I got a lot of help from her so I can pass the course."*

- *"XXX was the technical backbone of this project. I mostly had to catch up to him. He is a true project leader."*

- *"I stayed open to suggestions and led the assignment when I needed to. I tried to learn as much as I could so I could pass down my research to other students of the group."*

- *"YYY can hold his own weight when it comes to splitting our large assignments into smaller parts. I would say he did a great job of explaining what kind of content we need to produce using examples from the class textbook. ...Respectively, YYY has good leadership skills when my team and I were working on this project."*

- *"Incredible teammate. We went into this project with no experience towards our project and in the end we gained a lot more hands on experience from working with our team by learning from each other."*

6 Summary and Future Work

In today's computing-related job markets, teamwork skills have become increasingly important. It is crucial to train students to be effective team members or leaders as part of any computer science curriculum. ABET has recognized the significance of team assessment as one of six major program learning outcomes. Therefore, it is essential to emphasize the development of students' collaborative skills in project-based courses, preparing them for the workforce and bridging the gap between classroom experiences and real-world professional settings.

Based on our teaching experience, we propose a strategy to explicitly define team structures that mirror industry practices, enabling student teams to work effectively and efficiently. By clearly defining the roles and responsibilities of team members, students gain a clear understanding of their specific contributions as team players. This approach accommodates students with diverse backgrounds and expectations by tailoring learning objectives accordingly. Strong students are given opportunities to develop their leadership skills through project planning, task management, and mentoring others. Weaker students receive more accessible support from peers or team leaders, resulting in increased engagement in project activities and improved project grades. By explicitly specifying the roles of team members, task assignments become clearer, enabling the instructor to better manage teams, monitor project progress, and evaluate individual performance more accurately. In teams without a designated leader, students practice software development as egoless roles. As they assert their equal roles within the team, they typically enjoy working together and report positive team experiences.

Future work includes continuing implementation of this strategy and collecting additional data for analysis and comparison. As we have observed that team performance may sometimes deviate from the initially defined structure, it is important to explore how to effectively adapt team structures in the middle of the semester. Addressing this aspect warrants further attention and investigation.

References

[1] ABET. ABET criteria for accrediting computing programs, 2023-2024. `https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2023-2024/`.

[2] Chameera Dulanga. How to build a great software engineering team. August 08, 2021, Accessed on June 6, 2023.

[3] Claudia Iacob and Shamal Faily. The impact of undergraduate mentorship on student satisfaction and engagement, teamwork performance, and team dysfunction in a software engineering group project. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, page 128–134, 2020.

[4] Barbara Oakley, Richard M Felder, Rebecca Brent, and Imad Elhajj. Turning student groups into effective teams. *Journal of student centered learning*, 2(1):9–34, 2004.

[5] Beatriz Pérez and Ángel L Rubio. A project-based learning approach for enhancing learning skills and motivation in software engineering. In *Proceedings of the 51st ACM technical symposium on computer science education*, pages 309–315, 2020.

[6] Kai Presler-Marshall, Sarah Heckman, and Kathryn T Stolee. Identifying struggling teams in software engineering courses through weekly surveys. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 126–132, 2022.

[7] Kai Presler-Marshall, Sarah Heckman, and Kathryn T Stolee. What makes team[s] work? a study of team characteristics in software engineering projects. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 177–188, 2022.

[8] Stephen R Schach. *Object-oriented and classical software engineering*, volume 8. McGraw-Hill New York, 2010.

[9] Judith E Sims-Knight, Richard L Upchurch, TA Powers, Sara Haden, and Raluca Topciu. Teams in software engineering education. In *32nd Annual Frontiers in Education*, volume 3, pages S3G–S3G. IEEE, 2002.

[10] Anya Tafliovich, Andrew Petersen, and Jennifer Campbell. Evaluating student teams: Do educators know what students think? In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 181–186, 2016.

[11] Ellen L Walker and Oberta A Slotterbeck. Incorporating realistic teamwork into a small college software engineering curriculum. *Journal of computing sciences in colleges*, 17(6):115–123, 2002.

The CTEEAM Process in Practice: An Evaluation of Its Role in Digital Forensics Education *

Barry Bruster, Joseph Elarde, Mir Hansen
Austin Peay State University
Clarksville, TN 37044
{brusterbg, elardej, hasanm}@apsu.edu

Abstract

Digital forensics investigations often involve processing vast amounts of evidence and navigating the potential impact of cognitive biases. The CTEEAM (Critical Thinking, Ethical, Evidence, Analysis, Management) process presents an approach for structuring and organizing digital forensics investigations, enabling better evidence management, and reducing biases. This paper explores the implementation of the CTEEAM process in two different case studies involving undergraduate and graduate students. It also presents the results of a student feedback survey assessing how the CTEEAM process aided students in organizing evidence, discovering essential evidence items, and mitigating cognitive biases. The findings suggest that the CTEEAM process holds promise for improving digital forensics pedagogy and enhancing the discernment of valid information in the context of cybersecurity.

1 Introduction

Digital forensics has emerged as a key component in law enforcement and corporate incident response teams due to an increasing prevalence of cybercrime and policy violations. The abundance of digital devices and internet use has led to a crime surge, leaving a digital footprint [7], making digital forensics specialists more critical than ever. They are now more than ever tasked with

acquiring, extracting, validating, analyzing, and reporting digital evidence to solve cases and protect organizations from cyber threats. However, this rapid growth of digital evidence poses significant challenges for investigators and students who must process vast, sometimes conflicting, evidence reliably and without bias. Unfortunately, traditional educational resources often lack guidance on effectively organizing and analyzing evidence, potentially leading to confirmation bias and compromised case integrity.

Given these challenges, there is a pressing need for a comprehensive, systematic, and science-driven process that considers ethical aspects and counters cognitive biases [6, 2]. Within an educational context, this paper focuses on enhancing students' ability to organize and analyze evidence through critical thinking effectively. The proposed solution is incorporating the CTEEAM process (Critical Thinking Ethical Evidence Analysis Management) into undergraduate and graduate Digital Forensics course pedagogy. This process will be presented through two classroom case studies in this paper.

The structure of our paper is as follows: Section 2 overviews a proposed process solution called CTEEAM (Critical Thinking Ethical Evidence Analysis Management) that we have incorporated into our undergraduate and graduate Digital Forensics courses' pedagogy. This section also details the implementation of the process and presents a case study example. Section 3 outlines our evaluation methods, while Section 4 presents the results of our evaluation. Section 5 discusses the findings, and finally, Section 5 provides a summary, conclusions, and recommendations for future research.

2 The CTEEAM Process

The CTEEAM (Critical Thinking Ethical Evidence Analysis Management) process is designed to enhance digital forensics case investigations, addressing limitations in traditional course materials. Existing resources often fail to provide adequate guidance on evidence organization and bias reduction, hence the need for a method like CTEEAM. This process involves critical thinking, ethics, analysis, and management. It encourages scientific thinking while acknowledging potential biases strives for a balanced investigation of incriminating and exculpatory evidence, employs a systematic method with assigned metrics, and uses a spreadsheet to collect and organize case information.

CTEEAM enables students to navigate investigations effectively by assigning strength, relevance, and source quality metrics to evidence. This approach mirrors the creation of a qualitative risk assessment matrix, but in CTEEAM, a three-dimensional "magic cube" structure is created from evidence strength, relevance, and source quality metrics. The process includes various evidence organization techniques like timelines [4], social connections

[5, 1], means-motive-opportunity (MMO), and evidence categorization, aiding students in case comprehension. The procedure commences with case administrative information collection, followed by hypothesis development and active evidence-seeking. Key evidence is often highlighted after documenting it in the CTEEAM spreadsheet, revealing critical insights or indicating a need for further evidence collection.

2.1 Process Overview

The CTEEAM process is summarized as follows:

- Identify the main problem, question, and case administrative information.
- Develop a list of hypotheses, probabilities, and associated narratives to explain the main problem or question.
- Identify and list potential cognitive biases.
- Design the overall investigation plan.
- Segment problem/question into subordinate problems/questions if needed.
- Execute the investigation plan by examining and collecting evidence to support and refute hypotheses or subordinate questions.
- Reassess hypothesis probabilities.
- Reassess ethical assessment expectations, reviewing ethical considerations to ensure that incriminating and exculpatory evidence is documented.
- Conduct a root cause assessment.
- Review the investigation plan and ensure essential evidence items are captured, explained, and included in the case report.

In essence, the CTEEAM process offers a systematic method for students to examine digital forensics cases, involving hypothesis development, probability assignment, narrative writing, evidence collection and analysis, and ethical review. This enhances students' case understanding and leads to robust, well-backed conclusions.

2.2 Case Study Class Exercises

In Sections 2.2.1 and 2.2.2, we'll explore how the CTEEAM process was implemented in our Digital Forensics course through two case studies. Case study 1 highlights CTEEAM's use in a lecture and lab activity, demonstrating its practicality. Case study 2 depicts the process's incorporation into a gamified group activity, promoting collaboration and critical thinking. The CTEEAM process was also applied in individual homework assignments, allowing for independent case analyses. These varied applications underscore CTEEAM's versatility and effectiveness across different educational scenarios.

2.2.1 Case Study-1: Jane & Doug Case

In the first case study, students engaged in a multi-week murder mystery, the Jane and Doug Case. It involved physical and digital evidence and was structured with plot twists to keep student engagement high. Students assessed various types of evidence, developed hypotheses, and crafted narratives, learning the importance of critical analysis and avoiding premature conclusions. The aim was to illustrate how the CTEEAM process facilitates comprehensive, bias-free case building. Prior to beginning the case, students received instruction on the types of evidence and standards of proof.

The case unfolded in three parts – The Initial Investigation, Jane's Interrogation, and The Stamp Dealer, each presenting fresh evidence for assessment and discussion. These interactive sessions culminated in students documenting evidence types, crafting event timelines, and evaluating the case's strength. Following these debates, the instructor illustrated the CTEEAM process, showing its effectiveness in revealing pivotal evidence items for a case report. The case study showcased how CTEEAM can be instrumental in organizing, assessing, and evaluating evidence in complex digital forensics cases. Evidence was provided to students clearly and organized, step-by-step, to increase engagement and add drama to the activity. The evidence is as follows:

Part 1: Initial Investigation

- **E1**: Jane and Doug were heard fighting by Doug's neighbor on the morning of 9/25/2021. Jane was observed driving off.
- **E2**: Doug's neighbor, Jack, testified that he saw someone who looked like Jane at Doug's home on the night of 10/1/2021. It was dark and foggy.
- **E3**: Jack couldn't reach Doug after seeing his door open, he called the police.
- **E4**: Doug was found dead on his kitchen floor, poisoned.

- **E5**: Investigators found Jane's fingerprints inside Doug's house.
- **E6**: Investigators discovered an email sent by Jane on 9/26/2021, angrily attacking Doug for cheating on her.
- **E7**: Investigators found an email sent by Jane on 9/27/2021 threatening suicide.
- **E8**: Jane's web search history on 9/27/2021 included a search for poisons.
- **E9**: Investigators discovered a reply from Doug on 9/28/2021, apologizing and expressing his love for Jane and admitting he was ill with the flu.

Part 2: Jane's Interrogation

- **E10**: Under interrogation, Jane stated she visited Doug around 7:00 pm on 10/1, and they reconciled. He was fine when she left at 8:00 pm.
- **E11**: Jane forgot her cell phone at home on 10/1.
- **E12**: Jill, another neighbor, saw someone leave Doug's home at 10:00 pm.
- **E13**: Jane purchased gas at 9:30 pm near her home (35 minutes from Doug's).
- **E14**: Jill contacted investigators a day after being canvassed. She recalled the car was a dark blue Mercedes or Audi, but it was dark and foggy.
- **E15**: Jane drives a black Lexus 440 sedan.
- **E16**: Doug texted Debbie at 8:05 pm, ending their relationship.
- **E17**: Debbie drives a black Audi.
- **E18**: Debbie was found dead from poison, with a hand-printed suicide note.
- **E19**: Debbie's neighbor, Fred, thought he saw a visitor parked in her driveway at 11:00 pm.
- **E20**: Doug's time of death was approximately 9:00 pm.
- **E21**: Debbie's time of death was 11:30 pm.
- **E22**: Both died from the same poison.

- **E23**: Tom, Doug's brother, was stricken with grief upon learning of Doug's death.
- **E24**: Investigators noticed an object missing from Doug's wall - a prized stamp in Doug's collection.

Part 3: The Stamp Dealer

- **E25**: A call came in on Doug's landline from a stamp dealer who managed the sale of a rare Canada Two Cent Large Queen stamp.
- **E26**: The dealer confirmed the bank transfer information for the sale and mentioned a Swiss bank account number (931-340-7301).
- **E27**: The poison was injected into a bottle of 2016 Château Pape Clément Pessac-Léognan (Cabernet Sauvignon).
- **E28**: The wine bottle at Doug's home was wiped clean.
- **E29**: The same type of wine was found at Debbie's home with one print, not Debbie's.
- **E30**: Jack's cell phone records were obtained through a search warrant.
- **E31**: Records showed that Jack was at Debbie's home on the night of her death.
- **E32**: Fred, Debbie's neighbor, reviewed his security camera footage and reported the license plate number, which matched a Hertz rental car rented on the night of Debbie's murder.
- **E33**: A detective confirmed with the Hertz agent that Jack had rented the vehicle.
- **E34**: Searching Jack's home found the same wine and poison used in both deaths.
- **E35**: The fingerprint on the wine bottle found at Debbie's home matched the one found at Jack's home.
- **E36**: A web search on Jack's computer showed he was looking for hotels in Mexico.

Students were provided time to discuss the evidence in small groups and answer case questions for each part. They assessed the type of evidence acquired, the timeline of events, and whether a strong case could be made for the prime suspect (first Jane, then Jack) beyond a reasonable doubt. The CTEEAM process was used to examine the evidence throughout the exercise.

2.2.2 Case Study-2 M57-Jean Case Role Play Exercise

In this case study, students were given the M57-Jean Exfiltration case [3] scenario and divided into three teams. Each team had specific roles and responsibilities, and the exercise was conducted in several timed rounds to simulate the process of a real-world investigation. This exercise aimed to develop student skills in digital forensics investigations, enhance their understanding of team collaboration, and emphasize the importance of proper evidence management.

The M57-Jean Exfiltration case scenario was introduced to the class, with two image files available to investigate. Students were divided into teams, with each team member assigned a specific role:

- **Lead investigator** (manager, decision-maker)
- **Digital Forensics** Investigators (responsible for extracting evidence)
- **Document Specialist** (responsible for publishing the standard report)
- **CTEEAM Specialist** (responsible for updating the CTEEAM spreadsheet)

The Investigation part of the exercise was conducted in several timed rounds, each with a specific purpose.

- **Organization and Planning Round (Round 1)**: In this 15-minute round, teams developed an initial list of hypotheses, created an investigation plan, and documented ethical biases. The deliverables for this round included a professional report with research questions, hypotheses, an investigation plan, and ethical considerations. Points were awarded for the speed and quality of the report (3 points for first place, 2 points for second place, 1 point for third place).
- **Evidence Extraction Rounds (Rounds 2-3)**: In these 15-minute rounds, Digital Forensics Investigators extracted evidence using tools such as Autopsy, OSForensics, and FTK Imager. When notable evidence was discovered, it was tagged, commented on, and captured. Digital Forensics Investigators then emailed the evidence to their team members, with the email timestamp determining the placement of the evidence. Points were awarded for the evidence extracted (3 points for first place, 2 points for second place, 1 point for third place). The lead investigator documented the evidence on the whiteboard while the Document Specialist and CTEEAM Specialist updated their respective documents.
- **Report Generation Round (Round 4)**: In this 15-minute round, teams documented their conclusions and opinions in a standard report.

The goal was to provide a clear and concise summary of the evidence collected and present a well-supported hypothesis. Reports with well-structured timelines [4] and connection diagrams received two additional points each. Points were awarded for the correct solution, with a bonus for finding additional evidence indicating another crime. The report and CTEEAM spreadsheet were emailed to the instructor. The correct solution received 10 points, and the bonus points were 5 points.

The exercise aimed to develop and improve students' skills in digital forensics investigations and enhance their understanding of proper evidence management and teamwork. The engaging and competitive nature of the exercise provided students with an informative and enjoyable learning experience.

3 Methods

To evaluate the effectiveness of the CTEEAM process in digital forensics education, we conducted a two-pronged approach involving both in-class undergraduate students and graduate cybersecurity seminar participants. This evaluation process assessed the applicability and usefulness of the CTEEAM process in different educational settings and student groups by using a Likert survey containing the following questions:

- **Q1**: To what extent do you agree that the CTEEAM process helped you organize your evidence effectively during the case studies?"
- **Q2**: How strongly do you agree that the CTEEAM process assisted you in discovering key evidence items for your case reports?"
- **Q3**: How strongly do you agree that the timeline, social network, and magic quadrant diagrams included in the CTEEAM process helped you visualize important evidence and connections?"
- **Q4**: To what extent do you agree that the CTEEAM process helped you consider and avoid cognitive biases such as confirmation bias during the investigations?
- **Q5**: To what extent do you agree that the CTEEAM process effectively contributed to your overall understanding of digital forensics investigation and evidence management?

The survey aimed to assess various aspects of the CTEEAM process, such as its ability to help organize evidence, discover essential evidence items, consider and avoid cognitive biases, and visualize important evidence through various diagrams.

4 Results

In this section, we present the results of the survey conducted among two distinct groups: in-class undergraduate students and graduate seminar students. For the in-class undergraduate students, the survey results for questions 1-5 (Q1-Q5) are shown in Figure 1.

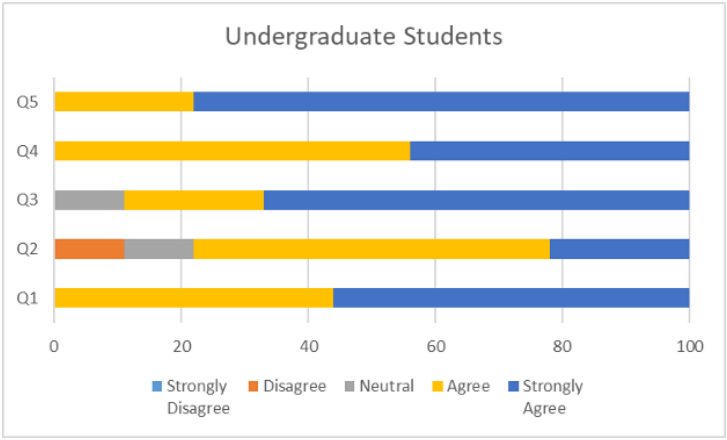


Figure 1: Undergraduate Student CTEEAM Evaluation Survey

The results for question (Q1) indicate that 100% of the students found the CTEEAM process helpful in organizing evidence. At the same time, 78% agreed or strongly agreed that the process assisted in discovering new evidence items (Q2). Regarding the recommended process visualization charts, 89% indicated they helped visualize important evidence items (Q3). When asked if the process helped consider and avoid cognitive biases, 100% of the undergraduates agreed or strongly agreed. Moreover, for the overall effectiveness question (Q5), 78% of the students strongly agreed the process was effective.

Similarly, For the in-class graduate seminar students, the survey results for questions 1-5 (Q1-Q5) are shown in Figure 2. The results for question (Q1) indicate that 100% of the students found the CTEEAM process helped organize evidence. Moreover, all graduate students agreed or strongly agreed the process assisted in discovering new evidence items (Q2), although only 17% strongly agreed. Regarding the recommended process visualization charts, 100% agreed or strongly agreed that the visualizations helped identify important evidence items (Q3). When asked if the process helped consider and avoid cognitive biases, 100% of the graduates agreed or strongly agreed. And finally, for the overall effectiveness question (Q5), 67% of the students strongly agreed that

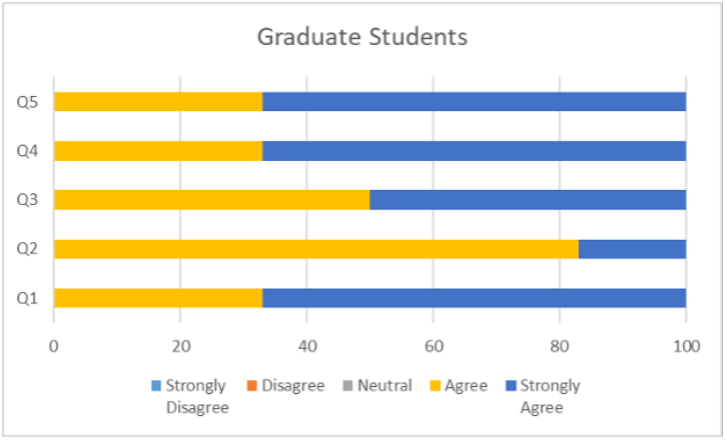


Figure 2: Graduate Students CTEEAM Evaluation Survey

the process was effective.

5 Discussion

This section interprets the CTEEAM process evaluation survey results from 14 undergraduate and 30 graduate seminar students. The findings indicate a positive reception of the CTEEAM process in digital forensics education, though given the small sample size, they are suggestive rather than definitive.

Undergraduate students unanimously agreed that the CTEEAM process helps organize evidence (Q1) and mitigate cognitive biases (Q4), suggesting its effectiveness in structuring evidence and maintaining objectivity. Moreover, 78% found it useful in discovering new evidence items (Q2), and 89% agreed that the process visualization charts helped visualize crucial evidence and connections (Q3), indicating the potential of CTEEAM in unearthing additional key evidence items and understanding complex data structures.

Similarly, graduate students agreed on the utility of the CTEEAM process in organizing evidence (Q1), discovering new evidence items (Q2), and identifying crucial evidence through visual tools (Q3). However, only 17% strongly agreed on the process's role in discovering new evidence items (Q2), and 67% strongly agreed about its overall effectiveness (Q5), suggesting potential areas for improvement for advanced students.

References

[1] Social network analysis. Technical report, Home Office of the United Kingdom of Great Britian, 1 2016.

[2] Digital forensics experts prone to bias, study shows, 5 2021.

[3] Simon Garfinkel. Digital corpora: 2009 m57-jean.

[4] Christa Miller. Timelines in digital forensic investigation: From investiagtion to court, 9 2020.

[5] Martin Mulazzani, Markus Huber, and Edgar Weippl. Social network forensics: Tapping the data pool of social networks. In *Eighth Annual IFIP WG*, volume 11, page 1Ũ20. Citeseer, 2012.

[6] Nina Sunde and Itiel E. Dror. A hierarchy of expert performance (hep) applied to digital forensics: Reliability and biasability in digital forensics decision making. *Forensic Science International: Digital Investigation*, 37:301175, 2021.

[7] David Williams. Discover how technology helps manage the growth in digitial evidence, 9 2022.

Effectiveness of Using Game Development in CS1: Faculty-Led or Peer Created Video-Based?*

Xin Xu, Wei Jin, Hyesung Park, Evelyn Brannock
Department of Information Technology
Georgia Gwinnett College
Lawrenceville, GA 30043
{xxu, wjin, hpark7, ebrannoc}@ggc.edu

Abstract

Many studies have shown that introductory programming courses are challenging and tend to have a low passing rate. In this project, the authors investigated the effectiveness of using peer-recorded game development videos in helping students learn Programming Fundamentals (CS1). The project idea is inspired by the positive result from spring 2021 where four IT faculty members adopted the game development modules in their programming courses. In spring 2023, the video-based workshops were piloted. In this paper, the authors will report the result of this pilot study and compare the result of the video-based workshop to the faculty-led PowerPoint (PPT)-based workshop in spring 2021.

1 Introduction

The current job market remains excellent for IT/CS majors as these careers are among the fastest-growing job categories. According to the US Bureau of Labor Statistics Occupational Outlook Handbook, overall employment in computer and information technology occupations is projected to grow 15% from 2021 to 2031. About 418,500 openings each year, on average, are projected to come from growth and replacement needs [16].

Despite the ever-increasing demands for a workforce with strong programming skills, many students are notably deficient in these skills[3]. The CS1 fundamental programming language course provides foundational knowledge and serves as a gateway course for high-level IT/CS courses at most schools. Unfortunately, this course has a high DFW rate. At the authors' institution, the CS1 course had a DFW rate ranging from 36% to 51% in different semesters. In contrast, the DFW rate was only 22% for other IT/CS courses. It is crucial to provide students with the right motivation in addition to technical skills and to increase the retention rate. This project was developed with the goal of utilizing engaging, active learning teaching methods to improve the student learning experience and lower the student failure rate in the CS1 class. The initial work started in AY 19-20 with the recruiting of peer students to develop games, and conduct workshops to guide other students to build the same game. In summer and fall 2020, faculty revised the workshop materials and adopted the workshop modules in spring 2021. In AY 22-23, video tutorials for these workshops were developed and adopted in spring 2023.

In the rest of the paper, the authors will present related work in Section 2, and share the history and the different implementation approaches of the project in Section 3. Section 4 compares the results of the two workshop approaches: faculty-led PPT-based and video-based. Finally, in Section 5, the advantages and disadvantages of the two approaches were discussed.

2 Related Work

Numerous universities and colleges, including research institutions and open access colleges, are facing declining retention rates, commonly referred to as the "storm" effect on higher education [13, 14]. Thus, improving student retention and motivation has become crucial for academic success and student achievement. Over the past decade, several studies have examined different approaches that focus on engagement, empowerment and adaptive learning environments. Engagement-based strategies, like collaborative learning and real-world applications, have been shown to enhance motivation and Retention [12, 5, 11, 4]. Empowering students is another effective strategy for motivation and retention [8]. This approach involves students in decision-making, promoting autonomy, and providing personalized feedback. Autonomy-supportive classrooms enhance intrinsic motivation and academic achievement [15]. Xu and Recker [17] conducted research on using AI and machine learning to personalize the learning experience and improve retention rates that can adapt to each student's needs.

Game development as a teaching tool for programming has gained traction, providing benefits such as increased engagement, improved problem-solving

skills, and a broader understanding of programming concepts [1, 2, 6, 7, 9, 10]. It motivates students through interactive and entertaining experiences, enhances problem-solving abilities, and prepares students for future careers.

3 Project Description

This project is a continuation of two previous projects. In the first project, students were hired as peer mentors to develop computer games (i.e., Flappy Birds) using Processing, a graphical programming environment based on Java. They also created workshops using step-by-step PowerPoint-based instructions, and conducted them in introductory programming classes taught using Java. However, the project was interrupted by the pandemic after the first in-person workshop. But the preliminary results showed the workshops were effective in motivating students to learn programming.

In the next project, faculty improved the student-created workshop modules utilizing their combined teaching and coding expertise. In spring 2021, each faculty adopted one set of workshops in their section(s) of the programming course. Despite the fact the school was still in the pandemic and some sections were online, the results were very encouraging. Students showed better attitudes toward the programming class, an improved curiosity and motivation for learning. However, it is challenging to promote the workshop modules among faculty since it requires training in Processing and modification of the teaching schedule to include the workshops.

This challenge motivated the authors to expand the project to include peers at one more subsequent level. In fall 2022, four IT peers were recruited to convert the PPT based workshop instructions into video-based workshop tutorials that could be assigned to students outside of the classroom. To ensure their suitability for the task, faculty members recommended students who completed gateway courses, and interviews were conducted to assess their capability to follow directions and successfully complete the project. The peers did all the video recordings and editing, and sometimes modification based on faculty feedback. In spring 2023, students in several sections of CS1 were assigned to develop the Circle Dodge game following a series of video-guided workshops, covering all major CS1 concepts. In each workshop, students were provided starter code. and a video in which the peer mentor explained the related programming concepts, in addition to demonstrating the code to add. This paper presents the pilot study comparing peer-created video tutorials to faculty-led PPT presentation-based workshop.

4 Project Evaluation

In spring 2021, the authors received 159 survey responses, composed of 77.35% male and 17.61% female. For the pilot study in spring 2023, 29 responses were received with 58.62% male and 37.93% female. For the discussion below, the authors will mark our previous project, the faculty-led PowerPoint-based workshops conducted in spring 2021 as Group 1 or G1, and this new project, the video-based peer-led workshop assignment for the second half of the semester, as Group 2 or G2. The race/ethnicity distribution of G1 and G2 is shown in Figure 1. This section shares the survey results of the students' feedback on their workshop experience, and the motivational factors for increased curiosity about programming and IT.

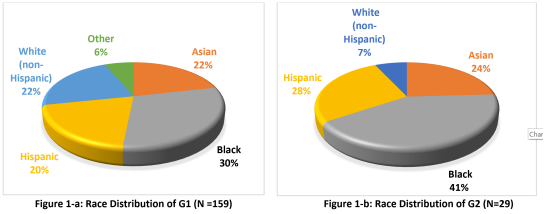


Figure 1: Race Distribution of G1 and G2

4.1 Analysis of Workshop Experience

In both G1 and G2, feedback on the following questions was gathered from students' workshop experience. A 5-point scale was used, with 5 representing strongly agree and 1 representing fully disagree. The parenthesized words are matching with the legend used in the charts:

- Did you find yourself engaged during the game development process? (engaged)
- Do you think the fact that the game was developed by previous students motivates you to develop similar projects? (motivated)
- Did you learn anything new from this workshop? (learned something new)
- Would you like to have more of this type of workshop as part of a standard programming class? (would like to have more)
- Did you enjoy the overall experience of developing a game using Processing? (enjoyed)

As shown in Figure 2, overall, students enjoyed the game development process and learned something new regardless of the workshop format (with

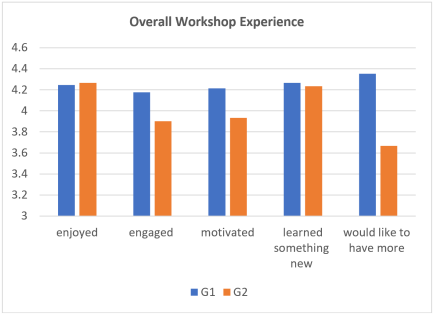


Figure 2: Analysis of Workshop Experience (G1 vs. G2)

both ratings above 4.2). However, G1 has a higher rating in engagement and is better motivated by the fact that peers developed the game and the workshop tutorials. The largest dissimilarity was in the response to having more of this type of workshop. 86% of the students in G1 answered yes (4) or definitely yes (5) to have more workshops, but in G2, about 66% answered so. Overall, students in G1 felt better engaged, more motivated by their peers, and therefore would like to have more game development workshops.

Further analysis on gender revealed that females have almost the same responses in both workshop formats on all the survey questions (see Figure 3-a). But the change of format from faculty-led PPT style workshop (G1) to video-based homework style workshop (G2) has a bigger impact on male students in engagement and motivation. As shown in Figure 3, even though male students agreed that they learned something new (average 4.21 in G1 and 4.12 in G2), and they enjoyed the game development process (4.22 in G1 and 4.18 in G2), in both workshop formats, their engagement dropped from 4.16 in G1 to 3.76 in G2, and motivation dropped from 4.19 in G1 to 3.71 in G2. The average rating for having more such workshops dropped from 4.37 in G1 to 3.35 in G2. Even though there is a decline in engagement and motivation, more than half of the male students in G2 still responded with definitely yes (5) or yes (4) on having more of such workshops. The result of the t-test also showed that the difference between these two workshop formats is not statistically significant($p>0.05$).

Analysis of the workshop experience was also conducted on different race/ethnicity groups. As shown in Figure 3-b, for Asian students, the overall rating for all survey questions is above 4.0 for both workshop formats. For Hispanics (Figure 3-d), an almost identical result was reported on enjoyment, motivation by the peers, and learned something new in both G1 and G2, and a light decrease of engagement (from 4.19 to 3.75). For African Americans, they re-

Figure 3: Workshop Experience by Gender and Race

ported a higher rating for engagement and motivation by the peers in G1 than in G2 (see Figure 3-c). Overall, African American and Hispanics would like to have more faculty-led workshops (average 4.46 and 4.3 in G1) than video-based workshops (average 3.58 and 3.75 in G2) as shown in Figure 3-c and 3-d. The comparison of White (non-Hispanic) group was not presented since only two responses were received from G2.

4.2 Analysis of Motivational Factors

If a student remains curious about programming and IT, it is often true that they will be motivated to continue learning in this field. So, feedback was obtained for the question "Did the workshop help you become more curious about programming and IT" as an indicator for motivation. In this section, the impact of the workshop format on motivation will be investigated. Factors were identified that make students become more curious about programming and IT and hopefully, subsequently, motivated to learn more. Students were asked to rate the effectiveness of the following statements on a scale from 1 to

5 (1 meaning not effective at all and 5 representing very effective).

- This game and workshop were developed by peer students.
- The workshop tutorial was engaging.
- I am able to apply programming concepts in a new environment.
- I am learning a new technology.
- I can be creative with programming.
- It seems not too hard to create a game.
- The workshop tutorial was engaging.

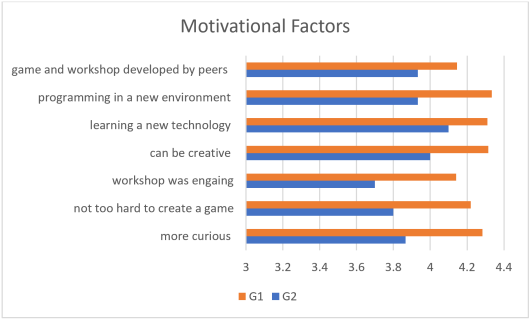


Figure 4: Motivational Analysis (G1 vs. G2)

Figure 4 showed that overall faculty-led workshops have a more positive result in increasing curiosity than video-based workshops (4.28 vs. 3.87), and the same for each factor that contributes to curiosity and motivation. However, being creative and learning a new technology played more important roles in increasing curiosity than other factors as the rating reached 4.0 or above regardless of the delivery format.

Figure 5-a showed that the females responded well in both formats but male students responded better to the faculty-led workshops than video-based workshops. Further Welch-test results showed none of these differences are statistically significant ($p>0.05$).

Analysis on race revealed that Hispanic students have almost the same responses in both workshop formats (Figure 5-d) with an above 4.0 average rating on all factors. In other words, they were motivated regardless of the delivery format. Asian and Black students were better motivated with faculty-led workshops than video tutorials, and the impact is bigger for Black students than Asian students (see Figure 5-b and 5-c) but not statistically significant ($p>0.05$). Again, the result for White students was not presented since there were only two responses.

Figure 5: Motivational Analysis by Gender and Race

5 Discussion and Conclusion

Overall, students reported a more positive workshop experience for faculty-led workshops than video-based workshops. The main reason for the difference could be the immediate feedback supplied in faculty-led workshops. There seems to be a benefit to direct interaction with faculty (and each other) inside the classroom (or virtual classroom for online synchronous sections). This format provides students with immediate responses and explanations to their questions. Even though faculty-led workshops generate more positive feedback, the authors also recognize the challenges of implementing the workshops since they require training for faculty to adopt the workshop modules. Another challenge is for faculty to modify their already time-pressured teaching schedule to add this game development component during class time. These challenges motivated them to develop the video-based workshop tutorials. The primary advantages of video tutorials for students and/or the instructor are:

- Flexibility: The instructor has the option to use the video workshops inside or outside of the classroom; therefore, minimizing the impact on the teaching schedule. The videos can be disseminated repetitively, semester

after semester, inside or outside of the classroom, in-person or online.
- Sustainable and easy to implement: The video workshops require virtually no training for faculty to adopt as all the instructions are provided in the assignment description, with built-in links for the videos and starter code. The videos can be re-used and do not rely on the instructor's specific expertise, teaching methods or experience level. The videos can help disseminate the work, easily allowing sharing with other educators and cross-institutionally to broaden the impact.

For faculty willing to take advantage of both, a hybrid approach is suggested, for example, having one in-class faculty-led workshop using PPT-based instruction or the video, but completing the rest outside of the classroom following the videos. Another factor that might affect the result is the game itself. The authors only had videos for Circle Dodge completed in spring 2023. Two additional sets of videos are now available. They are Flappy Birds and Snake, which happens to be the games used in G1. In the future, different sets of videos could be utilized to determine if the current results are dependent on the particular game deployed and thus affects the outcome.

References

[1] Jessica D Bayliss. Using games in introductory courses: tips from the trenches. In *Proceedings of the 40th ACM technical symposium on Computer science education*, pages 337–341, 2009.

[2] Jessica D Bayliss and Sean Strout. Games as a" flavor" of cs1. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 500–504, 2006.

[3] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, 2005.

[4] Jan Herrington and Ron Oliver. An instructional design framework for authentic learning environments. *Educational technology research and development*, pages 23–48, 2000.

[5] David W Johnson and Roger T Johnson. An educational psychology success story: Social interdependence theory and cooperative learning. *Educational researcher*, 38(5):365–379, 2009.

[6] Yasmin B Kafai. Playing and making games for learning: Instructionist and constructionist perspectives for game studies. *Games and culture*, 1(1):36–40, 2006.

[7] Fengfeng Ke. A qualitative meta-analysis of computer games as learning tools. *Handbook of research on effective electronic gaming in education*, pages 1–32, 2009.

[8] Martin Klein. Self-determination theory: Basic psychological needs in motivation, development, and wellness. *Sociologicky Casopis*, 55(3):412–413, 2019.

[9] Michael J Lee and Amy J Ko. Personifying programming tool feedback improves novice programmers' learning. In *Proceedings of the seventh international workshop on Computing education research*, pages 109–116, 2011.

[10] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15, 2010.

[11] Chris Mayfield, Sukanya Kannan Moudgalya, Aman Yadav, Clif Kussmaul, and Helen H Hu. Pogil in cs1: Evidence for student learning and belonging. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 439–445, 2022.

[12] Laura Mebert, Roy Barnes, Jacqueline Dalley, Leszek Gawarecki, Farnaz Ghazi-Nezami, Gregory Shafer, Jill Slater, and Erin Yezbick. Fostering student engagement through a real-world, collaborative project across disciplines and institutions. *Higher Education Pedagogies*, 5(1):30–51, 2020.

[13] Michael Nietzel. New data reveal more pandemic fallout: A historic drop in college persistence rates, 2021.

[14] Oleg V Pavlov and Evangelos Katsamakas. Will colleges survive the storm of declining enrollments? a computational model. *Plos one*, 15(8):e0236872, 2020.

[15] Johnmarshall Reeve. Why teachers adopt a controlling motivating style toward students and how they can become more autonomy supportive. *Educational psychologist*, 44(3):159–175, 2009.

[16] Bureau of Labor Statistics U.S. Computer and information technology occupations. `https://www.bls.gov/ooh/computer-and-information-technology/home.htm`.

[17] Beijie Xu and Mimi Recker. Teaching analytics: A clustering and triangulation study of digital library user data. *Journal of Educational Technology & Society*, 15(3):103–115, 2012.

Designing a No SQL - Non Traditional Databases Course*

Conference Tutorial

Karen E. Works
Computer Science
Florida State University
Panama City, FL 32405
keworks@fsu.edu

Abstract

NoSQL is becoming an in-demand required skill for data engineers and developers. [2] notes that Internet of Things (IOT) applications require workers to have NOSql skills. In "Why Amazon, Google, Netflix and Facebook Switched to NoSQL?" [3] Dr. Brock answers the fore-mentioned question by highlighting that relational databases are no/t designed to support the gargantuan amount of unstructured data produced nor the exponential growth of such data whereas NoSql databases are designed to support such environments. According to [1], "One could say that non-relational DB's are here to stay, and their popularity means that employers will be looking for those who are skilled in DBMS like it." Hence, to support our students we must teach them NoSQL.

In this tutorial, I will share my experience in designing, developing, and teaching an elective in NoSQL. I will share both the challenges and rewards in the hopes that it encourages others to incorporate NoSQL into a course as well. A variety of resources to develop and teach four different types of NoSQL databases will be presented.

This tutorial session will provide instructors with an introduction to NoSQL databases (non-relational databases). Participants will learn the basics of developing and implementing four types of NoSQL databases (namely, Document-Oriented, Key-Value Pair, Column-Oriented and Graph) as well as instructional approaches on teaching each within a course. This will be an active learning session with demonstrations and discussion activities.

References

[1] This is why you should learn mongodb. https://bootcamp.berkeley.edu/blog/learn-mongodb-database/. Accessed: 2023-05-02.

[2] Top iot job skills to get jobs in tech. https:/mondo.com/insights/in-demand-iot-skills/. Accessed: 2023-05-02.

[3] Shannon Block. Why amazon, google, netflix and facebook switched to nosql? https://www.linkedin.com/pulse/why-amazon-google-netflix-facebook-switched-nosql-shannon-block-cfe. Accessed: 2023-05-02.

Developing Identity-Focused Program-Level Learning Outcomes for Liberal Arts Computing Programs*

Conference Tutorial

Jakob Barnard[1], Grant Braught[2], Janet Davis[3], Amanda Holland-Minkley[4], David Reed[5], Karl Schmitt[6], Andrea Tartaro[7], James Teresco[8]

[1] *University of Jamestown, Jamestown, ND 58405*

Jakob.Barnard@uj.edu

[2] *Dickinson College, Carlisle, PA 17013*

braught@dickinson.edu

[3] *Whitman College, Walla Walla, WA 99362*

davisj@whitman.edu

[4] *Washington & Jefferson College, Washington, PA 15317*

aholl andminkley@washjeff.edu

[5] *Creighton University, Omaha, NE 68178*

DaveReed@creighton.edu

[6] *Trinity Christian College, Palos Heights, IL 60463*

Karl.Schmitt@trnty.edu

[7] *Furman University, Greenville, SC 29690*

andrea.tartaro@furman.edu

[8] *Siena College, Loudonville, NY 12211*

jteresco@siena.edu

Abstract

The SIGCSE Committee on Computing Education in Liberal Arts Colleges (SIGCSE-LAC Committee) has found that liberal arts and small colleges approach design of their computing curricula in unique ways that are driven by

institutional mission or departmental identity. This impacts how faculty at these colleges adopt curricular guidelines such as the current ACM/IEEE-CS CS2013[1]. The committee is developing guidance, informed by its sessions at recent CCSC and SIGCSE conferences, to help with the design and/or revision of CS curricula in liberal arts contexts [1]. This will ultimately be included in the committee's article in the Curricular Practices Volume that will be released as a companion to the new ACM/IEEE-CS/AAAI Computer Science Curricula guidelines (CS2023)[2]. Curricular guidelines like CS2013 or CS2023 inform curriculum design but are balanced with the vision for a program, departmental strengths, locale, student populations and unique academic experiences. The desire to craft distinctive curricula, combined with the size of prior curricular recommendations, requires an assessment of tradeoffs between achieving full coverage of curricular recommendations and a school's other priorities. SIGCSE-LAC's guidance will encourage faculty to reflect on their programs and the role of CS2023, beginning with their institutional and departmental priorities, opportunities and constraints. The specific goal of this session is to help participants develop programlevel learning outcomes that align with the unique features of their programs. Following an overview and brief discussion of the newest CS2023 draft, participants will begin working through a preliminary version of the committee's reflective assessment process. This process is framed by a series of scaffolding questions that begin from institutional and departmental missions, identities, contexts, priorities, initiatives, opportunities, and constraints. From there, participants will be led to identify design principles for guiding their curricular choices including the CS2023 recommendations. Examples gathered from the committee's previous CCSC and SIGCSE sessions will be available to help to articulate identity and program design principles, which will then be used for the identification of identity-focused program-level learning outcomes. Participants will leave the session with a better understanding of how CS2023 can impact their programs and a jumpstart on the entire reflective assessment process. Feedback on the process and this session are welcome and will be used to refine the committee's guidance prior to its publication in the CS2023 Curricular Practices volume.

Presenter Biographies

Two of the eight co-authors of this session plans to serve as presenters.

Grant Braught is a Professor of Computer Science at Dickinson College. He is a facilitating member of the SIGCSE-LAC Committee, has organized committee events focused on curricula and has published widely on issues re-

[1] `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`
[2] `https://csed.acm.org`

lated to CS education, particularly within the liberal arts. **Jim Teresco** is a Professor of Computer Science at Siena College. He has been involved in CCSC Northeastern for almost 20 years and currently serves as board chair, and has been involved with the SIGCSE-LAC Committee for 4 years. His research involves map-based algorithm visualization.

Other Author Biographies

Jakob Barnard is Chair and Assistant Professor of Computer Science & Technology at the University of Jamestown. He is a member of the SIGCSE-LAC Committee and his research involves how curricula has been integrated into Liberal Arts Technology programs. **Janet Davis** is Microsoft Chair and Professor of Computer Science at Whitman College, where she serves as the department's founding chair. She co-organized SIGCSE pre-symposium events in 2020 and 2021 on behalf of the SIGCSE-LAC Committee. **Amanda Holland-Minkley** is a Professor of Computing and Information Studies at Washington & Jefferson College. Her research explores novel applications of problem-based pedagogies to CS education at the course and curricular level. She is a facilitating member of the SIGCSE-LAC Committee. **David Reed** is a Professor of Computer Science and Chair of the Department of Computer Science, Design & Journalism at Creighton University. He has published widely in CS education, including the text *A Balanced Introduction to Computer Science*, and served on the CS2013 Computer Science Curricula Task Force. **Karl Schmitt** is Chair and Associate Professor of Computing and Data Analytics at Trinity Christian College. He has served on the ACM Data Science Task Force and various Computing, Technology, Mathematics Education related committees for the MAA, ASA and SIAM. His interests explore data science education, and interdisciplinary education between computing, mathematics, data, and other fields. **Andrea Tartaro** is an Associate Professor of Computer Science at Furman University. Her computer science education research focuses on the intersections and reciprocal contributions of computer science and the liberal arts, with a focus on broadening participation. She is a member of the SIGCSE-LAC Committee, and has published and presented in venues including the CCSC and the SIGCSE Technical Symposium.

References

[1] Amanda Holland-Minkley, Jakob Barnard, Valerie Barr, Grant Braught, Janet Davis, David Reed, Karl Schmitt, Andrea Tartaro, and James D. Teresco. Computer science curriculum guidelines: A new liberal arts perspective. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, SIGCSE 2023, page 617–623, New York, NY, USA, 2023. ACM.

Introduction to Non-Functional Requirements*

Conference Tutorial

Joe Temple
Department of Computer Sciences
Costal Carolina University
Conway, SC 29528
jtemple@coastal.edu

Abstract

There are three things that motivate this tutorial:

1. I spent the last 2 decades of my time at IBM providing services and remedial training on the non-programming aspects of IT solutions
2. I became aware of Dr Peter Denning's work, which lays out framework for teaching the computing sciences.
3. I became aware of the current work on the next round of curricula guidelines being done by a joint IEEE/ACM/AAII committee.

In this tutorial we will start with Denning's Framework of Practices for Computing Sciences[1]. Then we will discuss the role of Non-Functional Elements (NFEs) in IT. We use the acronym PASSAIC for the NFEs: Performance, Availability, Security, Scalability, Anti-fragility, Integrity, Costs. Then we will show how Performance and Cost combine with functionality (Programming) to create a value proposition. We will show that the "bookends", Performance and Cost, are driven by the rest of the NFEs. Finally we will discuss using NFEs as a vehicle for teaching Denning's Modeling practice. There is also an appendix that includes definitions of all the NFEs represented by PASSAIC[2, 3].

References

[1] Peter J. Denning. Great principles of computing. *Commun. ACM*, 46(11):15–20, 11 2003.

[2] Joe Temple. *Information Technology Performance.* Amazon Publishing, 2023.

[3] Joe Temple. *Nuts and Bolts: The World of Information Technology.* Amazon Publishing, 2023.

Curricular Practices for Computing for Social Good in Education*

Conference Tutorial

Heidi J. C. Ellis[1] and Gregory W. Hislop[2]
[1]Computer Science and Information Technology
Western New England University
Springfield, MA 01119
ellis@wne.edu
[2]College of Computing and Informatics
Drexel University
Philadelphia, PA 19104
hislop@drexel.edu

Abstract

The development of the ACM/IEEE/AAAI CS Curriculum Guidelines, CS2023, includes a parallel and collaborative effort to provide supplemental material that supports the Guidelines. A series of peer-reviewed articles are being created by experts in various aspects of the design and delivery of Computer Science programs. The Computing for Social Good committee is developing an article that provides an international perspective on Computing for Social Good in Education (CSG-Ed).

This workshop will inform the CCSC:SE community about the CSG-Ed effort while also obtaining comments and suggestions about the effort from the computing education community. The workshop will provide an overview of the global efforts in CSG-Ed, description of models for incorporating CSG into curricula, including examples, and recommendations and best practices for including CSG in curricula. The workshop outline will include:

- Introductions: 5 minutes
- Overview of the CSG-Ed efforts: 20 minutes

- Questions: 5 minutes
- Wrap up and Summary: 10 minutes

Breakout groups will be asked to provide comment about the CSG-Ed effort, identify any missing approaches, models, and best practices, and supply any other observations about the effort. The groups will also use a short set of prompting questions to consider Computing for Social Good in their own curriculum. These prompts will support discussion among attendees about opportunities and issues in addressing Computing for Social Good in undergraduate CS curricula.

Teaching the Divide-and-Conquer Closest Pair Algorithm Using a Map-Based Visualization*

Nifty Assignment

James D. Teresco
Department of Computer Science
Siena College
Loudonville, NY 12211
jteresco@siena.edu

This nifty assignment presentation is about an engaging, classroom-tested activity that uses interactive, map-based algorithm visualizations (AVs) of the brute-force and divide-and-conquer (D&C) approaches to the problem of finding the closest pair among a set of points. The D&C algorithm, *e.g.*, *EfficientClosestPair* in Section 5.5 of Levitin [1], is reasonably straightforward to describe, but it is the author's experience that even upper-level undergraduates in an Analysis of Algorithms course gain only a superficial understanding and do not appreciate its efficiency compared to the straightforward quadratic time algorithm. The activity presented here and the associated AVs offer a supplement or alternative to traditional instruction about this algorithm.

The activity and the data and AV tools it uses are part of a larger project called Map-based Educational Tools for Algorithm Learning (METAL) [2][1].

METAL's web-based AVs use its set of graph data, which is based on highways worldwide. Data is displayed in map form (optionally also in tabular form) along with an AV status panel showing psuedocode of the algorithm and values in key variables and data structures. As each line of psuedocode is executed, it is highlighted and the learner can see color-coded changes highlighted on the map and in the AV status panel. The activity briefly summarized here is one of the first of METAL's new easily-adoptable learning modules to be tested in a classroom[2]. The author guided students through the activity to help ensure it could be done within the 60-minute class time, and to be able to

[1] `https://courses.teresco.org/metal/`

[2] Spring 2023 *Analysis of Algorithms* class meeting at Siena College with 34 students

deal more quickly with any glitches in this first test run. Students each completed copies of the activity's instructions document with answers to questions and observations about the AVs and algorithms. The activity could also be completed independently or in small groups in a lab or homework setting, but would likely take longer with less instructor guidance.

In Part 1 of the activity, the AV of Brute-Force Closest Pair algorithm[3] is used to remind students how that algorithm works (they had studied it previously) and to help them appreciate the costs of a quadratic-time algorithm as larger data sets are used. In Part 2, they answer a series of questions designed to help them understand, at a high level, possible ways a set of points might be divided in half to enable a D&C approach. For Part 3 of the activity, the D&C Closest Pair AV[4] is used extensively in a guided manner to demonstrate the challenging ideas of the algorithm and gain an appreciation of how much more efficient this algorithm is in terms of the number of distances computed and compared. Part 4 builds on the last point by using the D&C Closest Pair AV on larger data sets. This has two advantages: being able to see the algorithm work in situations where more levels of recursion are needed, and by seeing how many distance computations are needed, in practice, for the brute-force and D&C algorithms. Finally, they examine the pattern to see that it matches the expected time complexity of each.

Informal feedback on this first use of the newly-implemented D&C Closest Pair AV and the associated activity was very positive. A more formal evaluation was infeasible in the constraints of the semester schedule. All of the METAL data and AVs are freely available, but the learning modules are available only upon request at this time. This learning module is one of several that have been developed and tested so far, and additional modules are in various states of planning or development.

References

[1] Anany Levitin. *Introduction to the Design and Analysis of Algorithms.* Pearson, 3 edition, 2012.

[2] James D. Teresco, Razieh Fathi, Lukasz Ziarek, MariaRose Bamundo, Arjol Pengu, and Clarice F. Tarbay. Map-based algorithm visualization with metal highway data. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 550–555, 2018.

[3] `https://tinyurl.com/wwmzbntx`
[4] `https://tinyurl.com/ymnn2rz9`

Integrating GIS Into CS2*

Nifty Assignment

Evelyn Brannock[1] *and Robert Lutz*[2]
[1]*Dept of Information Technology*
Georgia Gwinnett College, Lawrenceville, GA
ebrannoc@ggc.edu
[2]*Dept of Computer Science*
Piedmont University, Demorest, GA
rjlutz@piedmont.edu

Abstract

Geographic Information Systems (GISs) provide information databases for use in many application areas. OpenStreetMap (OSMap) is a map of the world, created and maintained by nearly 5 million users, using free tools and software [3]. By utilizing the Overpass Application Programming Interface (API) [1], rich GIS data can be integrated into standard assignments making the assignments more localized and familiar. By solving problems and learning CS skills that utilize familiar points of interest and places, student engagement is increased.

Materials

No materials are required beyond the standard tools used in CS2: a programming/runtime environment and an integrated development environment (IDE). The details of the assignment and the starter code will be provided upon request.

Assignment

The student is supplied starter code that retrieves a list of all named streets within a specific geography. In this example, we will process the results

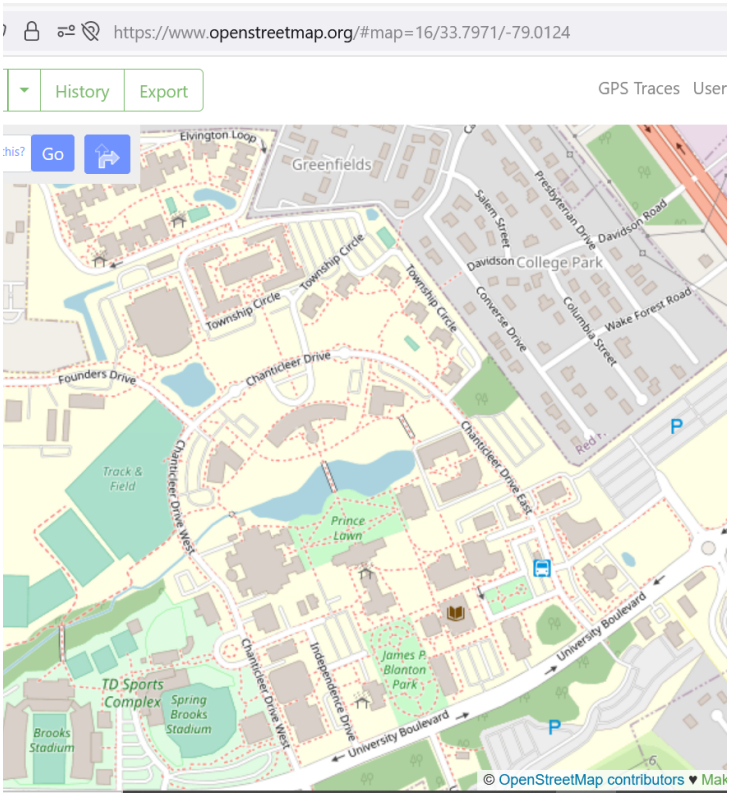


Figure 1: Map of CCU from OpenStreetMap [2]

of retrieving street names from OpenStreetMap. The input list is named *streetnames.txt* and is provided by the instructor. Modify the starter class, *StreetsWithNames.java*, to start FOUR threads. Each thread should count the number of entries that contain the string "peachtree" (case-insensitive) in the list (e.g. Peachtree or peacHTRee are valid occurrences). Each thread should update the provided *AggregateSum* class. Modify the *AggregateSum* class to make it thread-safe. Files supplied: *AggregateSum.java*, *streetnames.txt* *StreetsWithNames.java*.

Metadata

Summary	Students process a text file that contains occurrences of a popular street name in the local area using a real-time open-source GIS data query. Multi-threading is employed to accomplish a divide-and-conquer solution • A starter class and a data file are provided. A list of street names is prepared in advance by querying OpenStreetMap's Overpass API • Students use a reference text file with the street names. Processing is performed with parallel programming and the results are reported • Students make the API calls for a different area and street name
Topics	Text I/O, Multi-threaded programming, ReSTful Programming, JSON Processing, GIS Processing
Audience	Mid CS2, Web Development, Data Analytics / Data Visualization, Mobile Application Development, Multi-threading Processing
Difficulty	**Medium:** The capability to manipulate a new API, parse csv (or JSON) sequential text IO, understand parallel execution, and some (but nominal) knowledge of multi-threading from the student is required.
Strengths	• **Introduces the value of integration to other applications and tools:** Encourages learning another API "on the fly" • **Deep algorithmic and coding knowledge is not required:** Students are not required to understand advanced GIS concepts or algorithms; this knowledge is encapsulated in the OpenStreetMap and Overpass API • **Adaptability to multiple audiences:** Supports a variety of languages and starter code can be provided to scaffold content areas of other courses • **Gentle introduction:** Easy to use restful APIs and csv (or JSON) • **Simplicity in authentication:** No API key required • **Inspires experimentation:** Overpass API offers strong frontend w/queries
Weaknesses	• **Domain Overload:** GIS can offer very rich information data (relational, geographic, geometric, geological, land use, etc.) • **Performance Issues:** Open service throttling possible
Dependencies	OpenStreetMap platform and Overpass API
Variants	Any Overpass API query result can be utilized (walking paths, bike paths, post boxes, water ways, buildings on campus, etc.), multiple programming languages can be introduced, a user interface which includes specific queries of interest can be built, many visualizations can be explored, development of JSON processing skills, application of Java's Streams library.

Rubric

10	Four threads are created. Each thread counts the number of occurrences of peachtree (case-insensitive) in its own sub list, which is one quarter of the entire list.
10	AggregateSum has been updated to make it thread-safe
5	The aggregate sum is printed after all thread have concluded their processing.The aggregate sum is correct!
10	Make API call for city nearest out campus and count occurrences of streets containing Martin Luther King

References

[1] Overpass API Documentation. `https://osmlab.github.io/learnoverpass/en/docs/` [Accessed 1 August 2023].

[2] OpenStreetMap. Map of coastal carolina university. `https://www.openstreetmap.org/#map=16/38.9239/-94.7323` [Accessed 29 July 2023].

[3] OpenStreetMap. What is OpenStreetMap? `https://welcome.openstreetmap.org/what-is-openstreetmap/` [Accessed 1 August 2023].

ChatGPT: To Use or Not To Use, That is the Question*

Panel Discussion

Paul S. Cerkez[1], Joseph Edward Hummel[2], Marlon Mejias[3], William Pruitt[4]
[1] Department of Computer and Information Services
Office of Academic Integrity
Coastal Carolina University
Conway, SC 29528
pcerkez,aiofficier@coastal.edu
[2] Department of Computer Science
Northwestern University
Evanston, IL 60208
joe.hummel@northwestern.edu
[3] College of Computer and Informatics
University of North Carolina - Charlotte
Charlotte, NC 28223
mmejias@charlotte.edu
[4] DCS Corporation
6909 Metro Prark Drive, Suite 500
Alexandria, VA 22310
wpruitt@DCSCorp.com

ChatGPT, from OpenAI (AI – artificial intelligence), and the many similar Large Language Models (LLM) appear to have taken the world by storm with some for it, some against it. In simple terms, these products are a great tool for the experienced domain user, however, precisely because of their capability, there is a lot of controversy surrounding student's use.

There are basically two firmly entrenched camps in this debate. Some educational institutions are taking a "why fight it" approach and are creating classes to teach using it while others are discussing banning its use completely.

There are many variations along the continuum. One middle of the road approach some institutions are proffering is allowing its use as long as the sources used in creating the output can be tied to it. Some "industry technologists" are touting the greatness of it and how much time it saves them when doing some things, especially code generation. On the other side of the fence are those who want students to learn the domain before they can use the tool. As with the views of the academic's, there are many variations in 'acceptability' here also. The primary objective of this panel is to allow for a frank discussion of supporting, or not, ChatGPT use by students in an educational environment.