The Journal of Computing Sciences in Colleges

Papers of the 31st Annual CCSC Mid West Conference

September 20th-21st, 2024
Grand Valley State University
Grand Rapids, MI

Abbas Attarwala, Editor
California State University, Chico

Saleh Alnaeli, Regional Editor
University of Wisconsin-Stout

Volume 40, Number 4

September 2024

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

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
ACM2Y
Code Grade
GitHub

Welcome to the 2024 CCSC Mid West Conference

Welcome to the 31st Annual Midwest Conference at the Robert C. Pew Campus of Grand Valley State University in Grand Rapids, Michigan. This year's conference brings educators, researchers, students, industry professionals, and industry partners from across the region to explore the latest advancements and share insights in computer science. Sessions include a pre-conference workshop, keynote and dinner speakers, refereed papers, panels, tutorials, nifty assignments, works in progress, vendor talks, student showcase, and a student programming contest. We accepted 9 of 13 excellent paper submissions, a 69% acceptance rate. These selected papers represent cutting-edge research and innovative practices across various topics. We look forward to the pre-conference workshop by Michael Rogers and Bill Sevier, covering the basics of Git and GitHub, including using Git for collaborative projects and GitHub for managing assignments and providing feedback. We are excited to feature David Clark, author of Grading for Growth, as our keynote speaker, who will introduce Alternative Grading and cover what works and what does not. We are also honored to have Jonathan "J" Tower of Trailhead Technology Partners as our banquet speaker, sharing insights into the important trends happening in the software industry today. We extend our heartfelt thanks to everyone who made this conference possible: the conference committee, paper reviewers, speakers, presenters, and especially our site chairs, Zach Kurmas and Vijay Bhuse, and Grand Valley State University for their exceptional support in hosting this year's event. We also thank our National Partners ACM2Y, ACM CCECC, Code Grade, GitHub, and Rephactor for their continued support of our activities and UPE for student prizes. We have a full schedule of events and encourage you to make the most of the many opportunities for learning, collaboration, and professional growth. Your participation is critical to the success of this conference, and we hope you find the sessions engaging and inspiring. Thank you for joining us, and we look forward to a productive and enjoyable conference!

Jeff Lehman
Huntington University
2024 Midwest Conference Chair

2024 Steering Committee - CCSC Midwestern Region

Lucy La Hurreau, Registrar (2025)	Ivy Tech Community College, Fort Wayne, IN
Saleh M. Alnaeli, Editor (2024)	University of Wisconsin-Stout, Menomonie, WI
Michael P. Rogers, Webmaster (2025)	University of Wisconsin Oshkosh, Oshkosh, WI
Dominic Wilson, Treasurer (2026)	University of Findlay, Findlay, OH
Karl Schmitt, At-Large (2026)	Trinity Christian College, Palos Heights, IL
Kris Roberts, At-Large (2024)	Ivy Tech Community College, Fort Wayne, IN
David Largent, Regional Representative (2026)	Ball State University, Muncie, IN
Jeff Lehman, Conference Chair (2024)	Huntington University, Huntington, IN
David Largent, Past Conference Chair	Ball State University, Muncie, IN

2024 CCSC Midwestern Conference Committee

Jeff Lehman, Conference Chair (2024)	Huntington University, Huntington, IN
Zaid Althaha, Vice-Chair	University of Wisconsin-Parkside, Somers, WI
Zachary Kurmas, Site Chair	Grand Valley State University, Allendale, MI
Vijay Bhuse, Assistant Site Chair	Grand Valley State University, Allendale, MI
Saleh Alnaeli, Authors (2024)	University of Wisconsin-Stout, Menomonie, WI
Ahmed Elmagrous, WIP, Nifty Tools/Assignments	University of Wisconsin-Stout, Menomonie, WI
Cathy Bareiss, Panels, Tutorials, Workshops	Bethel University, Mishawaka, IN
Imad Al Saeed, Papers	Saint Xavier University, Orland Park, IL
David Largent, Past Conference Chair	Ball State University, Muncie, IN
Paul Talaga, Programming Contest Co-Chair	University of Indianapolis, Indianapolis, IN
Md Haque, Programming Contest (Assistant)	University of Indianapolis, Indianapolis, IN
Nathan Sommer, Programming Contest (Assistant)	Saint Xavier University, Orland Park, IL
David Largent, Publicity	Ball State University, Muncie, IN
Lucy La Hurreau, Registrar (2025)	Ivy Tech Community College, Fort Wayne, IN
Deborah Hwang, Registrar (Assistant)	University of Evansville, Evansville, IN
Stefan Brandle, Speakers Chair	Taylor University, Upland, IN

William Turner, Speakers (Assistant) . . . . Wabash College, Crawfordsville, IN
Dominic Wilson, Treasurer (2026) . . . . University of Findlay, Findlay, OH
David Largent, Student Showcase Chair . . . Ball State University, Muncie, IN
Andy Harris, Past Conference (Assistant) . Ball State University, Muncie, IN
Kris Roberts, Two-year College Liaison Co-Chair . . . . . . Ivy Tech Community College, Fort Wayne, IN
Takako Soma, Vendors . . . . Illinois College, Jacksonville, IL
Michael P. Rogers, Webmaster (2025) . . . . . University of Wisconsin Oshkosh, Oshkosh, WI

Raising the bar: What works, what doesn't, and what to do next with alternative grading [*]

Keynote

David Clark
Department of Mathematics
Grand Valley State University
Allendale, MI 49504
clarkdav@gvsu.edu

Points, partial credit, and weighted averages are so traditional that it's hard to imagine a world without them. But there are some big questions about traditional grading systems: Do they show what a student has (or hasn't) learned? Could students "get by" on partial credit – or fail a class due to the formulas we use, even though they've actually learned everything we could ask? What does a mashed-up average of points and partial credit even mean? The good news is that there are better options: approaches to grading that are grounded in how humans actually learn and better represent that learning, all while upholding high standards. In this keynote, you'll learn the basics of alternative grading, see what works (and what doesn't), identify practical ways to fit new grading practices into your classes, and find resources to help you build a grading system that actively supports learning.

[*]Copyright is held by the author/owner.

The Future Now: The Top Trends In Software Development *

Keynote

Jonathan "J." Tower
Trailhead Technology Partners
Jenison, MI 49428
jonathantower@gmail.com

To prepare students for careers in a rapidly changing industry, it is crucial for computer science instructors to stay updated on the latest trends and innovations in the field. Jonathan "J." Tower, a sought-after industry expert, will provide insights into some of the most important trends happening in the software industry today, from AI-driven development and cloud-native architectures to the rise of low-code platforms. As a ten-time recipient of the Microsoft Most Valuable Professional (MVP) award and the founder of Trailhead Technology Partners, J. has deep industry experience and an understanding of the evolving landscape of software development in business, His thought leadership in the software world uniquely positions him to guide educators through the latest software trends the same way he regularly does for professionals. Be part of the conversation on the forefront of software development and equip yourself with the knowledge to stay ahead of the latest trends in our ever-changing field.

*Copyright is held by the author/owner.

Low Code App Development[*]

Conference Workshop

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

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

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

References

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

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

[*]Copyright is held by the author/owner.

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

Tool Support for Teaching Deductive Databases[*]

Work in Progress

Ramachandra B. Abhyankar
Computer Science Department
Indiana State University
Terre Haute, IN 47809
RB.Abhyankar@indstate.edu

In recent years, new applications have been found for deductive database systems [1]. Consequently, there has been a revival of interest in deductive databases. Several articles and books have appeared in recent years to augment older articles and books . The terms "Deductive Databases", "Datalog and its Extensions", "Logic Databases", "Knowledge Bases" , and "Answer Set Programming" are often used interchangeably. A related topic is "Prolog." While the theory of deductive databases was rapidly developed and disseminated through books and articles, implementations of deductive database systems remained mostly experimental. These experimental systems were often available on only one platform, and were difficult to acquire and install. "Modern" Data log Engines [2] have not seen widespread use either. The difficulty of acquiring a robust deductive database system for use in teaching was acknowledged by the creators of DES (Datalog Educational System). The use of the DES system itself required knowledge of Prolog. After looking at several candidates, it appears that at the current time, perhaps the most promising option is the DLV system. This system is available on all major platforms and is easy to install. It comes with a good tutorial and good documentation. The system is free for educational use. This work in progress presentation will review older and newer experimental Datalog systems and report on the use of DLV in teaching Deductive Database concepts. In future work, it would be interesting to compare the capabilities of the DLV system with the successful Answer Set Solver Clingo.

[*]Copyright is held by the author/owner.

References

[1] Sergio Greco and Cristian Molinaro. *Datalog and Logic Databases.* Springer-Verlag, 2015.

[2] Bas Ketsman and Paraschos Koutris. *Modern Datalog Engines.* Now Publishers, 2022.

Enhancing Student Engagement and Interdisciplinary Research with IoT and Drone Frameworks*

Work in Progress

Saleh M. Alnaeli[1], Ahmed. A. Elmagrous[2]
[1]Math, Stat, Computer Science Department
University of Wisconsin-Stout, Menomonie, WI 54751
{alnaelis, elmagrousa}@uwstout.edu

Presented is a framework that integrates a sequence of hands-on projects using IoT and Scientific drones to increase students' motivation in computer science and engineering and is aimed at supporting interdisciplinary research in colleges. This framework provides a practical approach for engaging students in application development and real-world problem-solving. The initial Key projects of the proposed framework include: 1) Custom-Built Drone: An educational drone designed to address technical challenges and offer solutions. It is equipped to carry a cluster of handheld computers, microcontrollers, and multiple programmable modules programmed in languages typically taught in computer science courses. 2) Smart Weather Stations: Custom-built stations with different sensors. These stations collect real-time data and transmit it to the cloud for processing via cellular networks, primarily for smart farming and agriculture. The project will expose students to various technologies and ideas enhance their problem-solving skills and promote interdisciplinary research. 3) IoT-Based Robotic Arm: A robotic arm designed for diverse applications, including laboratory use, drones, and first-aid scenarios. It features Arduino integration and a long-distance remote-control system.

The framework is adaptable and flexible, allowing for expansion as new projects are developed. This educational project offers students opportunities to build communication, problem-solving, and technical skills valuable in both academia and industry. It can also be applied to areas like precision agriculture, freshwater studies, and atmospheric pollution research.

*Copyright is held by the author/owner.

Automated Threat Detection to Support the Process of Security Requirements Elicitation[*]

Work in Progress

Sugandha Malviya
Department of Computer Science
Ball State University, Muncie, IN 47396
sugandha.malviya@bsu.edu

Security requirements are essential for identifying and mitigating potential threats to the system being developed. However, eliciting these requirements can be particularly challenging as the threat landscape constantly changes, with new vulnerabilities and attack vectors emerging regularly. This dynamic nature of threats makes it difficult to define a set of comprehensive security requirements early in the system development lifecycle.

Unlike most existing threat detection tools that generally rely on predefined rules and patterns or use conventional machine learning algorithms, my proposed solution utilizes advanced Natural Language Processing (NLP) techniques. These techniques enable the processing of security documents in a deeply contextualized manner, thereby enhancing the threat detection process.

The complete process involves several different steps, including gathering a diverse set of security-related documents, such as incident reports, previous threat analyses, etc., for training and validation purposes, cleaning and preprocessing the dataset, and developing and testing the NLP model. However, a pivotal aspect of this process is the construction of a baseline model that uses a hybrid approach, integrating elements of both Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) to extract threats from textual security documents. The synergistic use of CNNs and RNNs allows the model to efficiently extract relevant features and to understand their contextual use. For instance, the model can discern the significance of a term like, "injection" within various contexts, distinguishing between its use in a medical document and identifying it as a security threat in the form of SQL injection

[*]Copyright is held by the author/owner.

in technical documents. This nuanced understanding is crucial for accurate threat detection and analysis in security documents.

Subsequently, the identified threat can be analyzed to generate misuse cases and security requirements and support the requirements elicitation process. This innovative approach aims to adapt dynamically to the evolving threat landscape and automatically generate relevant security threats.

A Malleable Framework for Efficient Learning[*] Work in Progress

Pradip Peter Dey, Mohammad Amin
Bhaskar Raj Sinha, Hassan Badkoobehi
School of Technology and Engineering
National University
9388 Lightwave Ave., San Diego, CA 92123
{pdey, mamin, bsinha, hbadkoob}@nu.edu

The pace of change is accelerating, and the ability to adapt and learn new things quickly is becoming essential. The proposed malleable learning framework is an approach designed to address the challenges of learning quickly. Artificial Intelligence, automation and so forth may eliminate many traditional jobs and create many new jobs that require new skills and knowledge. Employers are facing challenges in finding candidates with adequate skill sets. There is a gap between vacant positions requiring new skills and available candidates to fill them. However, it may be difficult for some learners to acquire these skills easily because traditional skill acquisition environments available in colleges and universities are not adequate for learners who want to rapidly acquire the skills without waiting for semesters, schedules, and registration processes. A wide variety of alternatives should be available to learners. Cost-effective learning strategies, approaches, environments, resources, and tools should be available to learners in cooperation with other stakeholders. The highly flexible framework proposed here combines all essential features in an innovative way to serve the learners for acquiring appropriate skills for the current and emerging job markets. Critical thinking, problem solving, and a mindset to continuously upskill and re-skill is essential. Using the multifaceted role of AI, we plan to emphasize AI enriched just-in-time learning in the malleable framework.

[*]Copyright is held by the authors/owners

The challenges of malleable framework design are related to the rapid development of educational technology, the diversification of student populations, the growing expectations related to the skills needed in current and future working life [1], and the controversy about ethical use of AI in education. With adequate research and preparation of resources and tools, learning environments may bring help and cooperation from stakeholders to serve the learners. The malleable leaning framework is based on the assertion that effective learning happens when it is adaptable and responsive to the needs of the learner combined with the demands of the industry. Unlike rigid academic program structures that follow fixed patterns and schedules, this framework allows for a more adaptable and tailored educational experience. This framework represents an innovative approach to education, emphasizing flexibility and personalization in the learning space. Additional learning strategies also need to be considered for workforce development in order to eliminate the gap between the requirements of vacant positions and available candidates. Although widespread implementation may still be challenging, the potential benefits of preparing learners for a rapidly changing future may make the effort worth considering. Including these ideas in academia, education may become a more effective instrument for lifelong success.

References

[1] Teemu Valtonen and Ulla Leppanen et al. “Learning environments preferred by university students: a shift toward informal and flexible learning environments”. In: *Learning Environments Research* 24 (2021), pp. 371–388.

Artificial Intelligence in the HigherEd Classroom – Boon or Bane?*

Work in Progress

Anuradha Rangarajan[1], Calvin Nobles[2]
Rangarajan Parthasarathy[3], Maurice Dawson[1]
[1]Illinois Institute of Technology, Chicago IL 60616
{arangarajan, mdawson2@iit.edu}
[2]University of Maryland Global Campus, Adelphi MD 20783
calvin.nobles@umgc.edu
[3]University of Wisconsin-Green Bay, WI 54311
parthas@uwgb.edu

The effective and efficient use of Artificial Intelligence (AI) technology is fast becoming a skill that makes students competitive in the job market and beyond. However, successfully incorporating AI in the classroom presents unique challenges that requires instructors and students to adopt, ethically use, and evaluate AI-infused pedagogy. Research literature points to the positive role of AI tools in improved appropriability and evocativeness, which lead to better learning outcomes through greater personalization of information gathering and better personal reflection processes. AI-driven chatbots could support learning styles of diverse students and provide an opportunity for slow learners to perform well, in contrast to a traditional classroom where all students may be required to keep pace with instructor's delivery. AI-powered Intelligent Tutoring System (ITS) can assist both students and instructors. Step-by-step feedback and suggestions for improvement could help the student's learning in an interactive manner. Instructors could also use AI supported ITS for simplifying and automating grading, thereby providing quicker and more meaningful feedback to students regardless of class size. Notwithstanding the benefits, it is noteworthy to highlight AI's downsides in the classroom. It is plausible that the availability of AI tools causes complacency in students who may skip in-person lectures and rely on remote learning using AI alone. AI based tools could be

*Copyright is held by the author/owner.

misused by students for completing assignments, leading to questions concerning academic ethics and fairness. Universities and instructors should reflect on how to modify assessments to objectively measure and quantify performance outcomes in an AI world.

HigherEd is navigating an unchartered territory which needs to be explored further, before a viewpoint can be formed on whether AI in the classroom is a boon or a bane or perhaps a little of both. This work-in-progress surveys broad use cases and challenges pertaining to the classroom adoption of AI.

The Changing Face of Computer Science Education – What Lies Behind and What Lies Ahead?*

Work in Progress

Anuradha Rangarajan[1], Calvin Nobles[2]
Rangarajan Parthasarathy[3], Maurice Dawson[1]
[1]Illinois Institute of Technology, Chicago IL 60616
{arangarajan, mdawson2@iit.edu}
[2]University of Maryland Global Campus, Adelphi MD 20783
calvin.nobles@umgc.edu
[3]University of Wisconsin-Green Bay, WI 54311
parthas@uwgb.edu

The domain of computer science (CS) education has seen many inclusions, exclusions, and transformations over the past decades. CS education has come a long way from its initial goal of offering training in core technology development including software programming in the commonly used software and hardware development to build better and faster computers. There was a point in time when CS education involved perspectives of computational learning, commensurate with the thinking that software and hardware could be used to solve problems in multiple knowledge domains (such as in library science). When faced with low enrollment and interest of women and minorities in CS education, there was an impetus on encouraging diversity in CS education, leading to diversity-inclusive CS instruction.

In more recent times, CS education has encompassed social media related computing, cloud computing, mobile computing, and innovative applications such as digital marketing. During this decade there has also been an increased focus on Information Technology (IT) education, a domain related to CS, whose scope includes software and hardware used for processing and securely transmitting information. The proliferation of computer technology has given birth

*Copyright is held by the author/owner.

to cybercrimes which, in-turn, has brought about the ever-increasing demand for Information Security and Cybersecurity education. A focus on datadriven decision making in industry has made imp between CS, business, and engineering departments in academia, to implement data science programs emphasizing business decision making skills and data analysis. Each of these milestones has presented both challenges and opportunities for CS education to develop into what it is today. The pace of technology change is accelerating the demand for a ubiquitous CS pedagogy, which calls for disruptive innovation made stronger by academia-industry collaboration. Our work-in-progress will explore the changing face of CS education, trace its evolution, and discuss what lies behind and what may lie ahead.

Course in a Box - AI[*]

Conference Tutorial

Ramachandra B. Abhyankar
Computer Science Department & Engineering
Indiana State University
Terre Haute, IN 47809
RB.Abhyankar@indstate.edu

1 Which AI ? AI Paradigms, Traditional, New, AML or Commonsense AI ?

A generalist being called upon to teach a course on "AI" is certainly possible – because of the interest among students in an "AI" class (with all the buzz surrounding "AI"). This can happen in situations involving retirement of experienced or specialist faculty, COVD-related disruptions at universities, etc. Often the starting point for a generalist faculty is the "catalog description" of a course that has been "on the books" in the university catalog. An example of such a course description is given below:

"CSxxx: Artificial Intelligence Concepts and applications, including artificial intelligence programming languages, history, present and future development and research, expert systems, natural language processing, intelligent machines/robots, and vision. Development of artificial intelligence course project."

The above describes a "traditional" AI course, that does not address topics in the "the New AI". "The New AI " topics, listed in [1] are: Neural Networks, Genetic Algorithms and Evolutionary Computing, Fuzzy Systems, Rough Sets, Chaos.

Books on AI emphasizing "Intelligent Systems" such as [2] , cover the following topics: Introduction to Knowledge-Based Intelligent Systems, Rule-Based Expert Systems, Uncertainty Management in Expert Systems, Fuzzy Expert

[*]Copyright is held by the author/owner.

Systems, Frame-Based Expert Systems, Artificial Neural Networks, Evolutionary Computation, Hybrid Intelligent Systems, Knowledge Engineering and Data Mining.

Another division of topics, made in [3] is between Adaptive Machine Learning (AML) and Good-Old-Fashioned-AI (GOFAI), which emphasizes Commonsense Reasoning. In recent times, AML has had much success in diverse areas: medicine, finance, defense, and the list goes on. In [3] Levesque cites examples where AML does not help, and Commonsense Reasoning is needed. Commonsense Reasoning [18] , unlike classical logic, is non-monotonic. Books on Knowledge Representation and Reasoning [19] , an important sub-area of Artificial Intelligence, discuss non-monotonic logics. The literature on non-monotonic logics is vast.

Then there is also the Logic-Based Approach to AI, that emphasizes applications of Logic in Artificial Intelligence as described in [4] and other books. Paradigms of Artificial Intelligence are discussed in [15].

It appears from the above that "AI" is a very wide field and no "AI class" will be able to cover all the topics. Consequently, an "AI class" will either have to be a "survey" of all topics, or will have to focus on a small subset of topics.

2 Student Background

The level of preparation of students in various topics will definitely affect the kind of course that would be right. An important issue in the design of an "AI class" is the programming background that students taking the class will have. If students have knowledge of one or more "AI languages" such as Prolog or Lisp, class time will not have to be spent on teaching these languages. In the same way, the background of the students in areas such as logic, mathematics and statistics, will affect what topics can be properly covered in the class; else time must be spent in class, covering these topics. .

3 Textbooks

Textbooks covering a fairly large number of topics are [5] and [7]. A "traditional AI " textbook is [13]. A newer significant textbook is [14]. AI programming techniques in Prolog and LISP are treated in [16] and [17].

4 Projects

Projects involving Constraint Problem Solving, Natural Language Understanding, Planning, Game Playing, and Deductive, Inductive and Abductive Reasoning can be assigned based on [6]. An Expert System project can be assigned based on material in [8], [9]. Projects using Theorem Proving Programs can be given based on [10],[11] ,[12] .

5 Syllabus

A syllabus can be prepared after the topics to be covered in the course have been determined.

References

[1] David L. Poole and Alan K. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2010.

[2] Larry Wos and Gail W. Pieper. *A Fascinating Country in the World of Computing*. World Scentific Publishing, 1999.

[3] Joseph C Giarratano and Gary D. Riley. *Expert Systems: Principles and Programming, Fourth Edition*. Course Technology, 2005.

[4] Ronald Brachman and Hector Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann, 2004.

[5] Ivan Bratko. *Prolog Programming for Artificial Intelligence (4th Edition)*. Pearson, 2011.

[6] Ernest Davis. *Representations of Commonsense Knowledge*. Morgan Kaufmann, 1990.

[7] Wolfgang Ertel. *Introduction to Artificial Intelligence, Second Edition*. Springer-Verlag, 2018.

[8] Adrian Groza. *Modelling Puzzles in First Order Logic*. Springer-Verlag, 2021.

[9] Achim G. Hoffmann. *Paradigms of Artificial Intelligence*. Springer-Verlag, 1998.

[10] John Arnold Kalman. *Automated Reasoning with OTTER*. Rinton Press, 2001.

[11] Hector J. Levesque. *Thinking as Computation: A First Course*. MIT Press, 2012.

[12] Hector J. Levesque. *Common Sense , The Turing Test , and the Quest for Real AI*. MIT Press, 2017.

[13] Dennis Merritt. *Building Expert systems in Prolog*. Springer-Verlag, 1989.

[14] Jack Minker. *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.

[15] Toshinori Munakata. *Fundamentals of the New Artificial Intelligence: Beyond Traditional Paradigms*. Springer-Verlag, 1998.

[16] Michael Negnevitsky. *Artificial Intelligence: A Guide to Intelligent Systems*. Addison Wesley, 2002.

[17] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Company, 1980.

[18] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis.* Morgan Kaufmann, 1998.

[19] Peter Norvig. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp.* Springer-Verlag, 1991.

Using CS2023 to Create Computer Science Courses and Curricula*

Conference Tutorial

Amruth N. Kumar[1], Cathy Bareiss[2]
[1]Computer Science Department
[1]Ramapo College of New Jersey
Mahwah, NJ 07403
amruth@ramapo.edu
[2]Bethel University
Mishawaka, IN 46614
cathy.bareiss@betheluniversity.edu

Abstract

In this special session, the knowledge model and competency framework of CS2023 will be presented. The steps for using them to design computer science courses and curricula will be discussed.

1 Overview

The Association for Computing Machinery (ACM) has been publishing curricular guidelines for computer science since 1968 [6, 5]. Since 1991, it has been collaborating with IEEE-Computer Society (IEEE-CS) to produce these decennial reports 1991 [2, 3, 4, 1]. For the latest curricular report on computer science (CS2023), the Association for the Advancement of Artificial Intelligence (AAAI) also joined the effort, given the increased role of artificial intelligence in undergraduate computer science. The three professional bodies together set up a task force of educators and practitioners from around the world that worked for three years to produce CS2023 (csed.acm.org). In this special session, the CS2023 report will be presented and discussed. The CS2023 task force undertook the following activities: • Revision of the knowledge model

*Copyright is held by the author/owner.

of the curriculum last published in 2013 [1]. The knowledge model consists of 17 knowledge areas. Each knowledge area consists of knowledge units that are groups of related topics and learning outcomes. • Design of a competency framework that can be used by adopters to build a competency model of the curriculum. This includes a framework for systematically identifying tasks, a format for competency specification, and an algorithm to use the competency framework to build a customized competency model. • Curricular practice articles written by experts on social issues, professional practices, pedagogical concerns, and programmatic considerations. The goal of the articles is to summarize the state of the art, inform educators, and attempt to advance computer science education practices.

2 Acknowledgements

Partial support for this work was provided by the National Science Foundation under Award DUE-2231333.

References

[1] ACM/IEEE. Acm/ieee-cs joint task force on computing curricula. 2013. *ACM Press and IEEE Computer Society Press. https://doi.org/10.1145/2534860*, 2017.

[2] Keith Barker Kim B. Bruce J. Thomas Cain Susan E. Conry Gerald L. Engel Richard G. Epstein Doris K. Lidtke Michael C. Mulder Jean B. Rogers Eugene H. Spafford A. Joe Turner Allen B. Tucker, Robert M. Aiken and Bruce H. Barnes. Computing curricula 1991: Report of the acm/ieee-cs joint curriculum task force. technical report. *ACM Press and IEEE Computer Society Press, New York, NY, USA*, 1991.

[3] J. Educ. The joint task force on computing curricula. 2001. computing curricula 2001. *Comput. 1, 3es*, 2001.

[4] Gordon Davies Mark Guzdial Renée McCauley Andrew McGettrick Bob Sloan Larry Snyder Paul Tymann Lillian Cassel, Alan Clements and Bruce W. Weide. Computer science curriculum 2008: An interim revision of cs 2001. *ACM Press, New York, NY, USA*, 2008.

[5] Della T. Bonnette Gerald L. Engel Richard H. Austing, Bruce H. Barnes and Gordon Stokes. Curriculum '78: Recommendations for the undergraduate program in computer science— a report of the acm curriculum committee on computer science. *ACM 22, 3 (March 1979), 147–166. https://doi.org/10.1145/359080.359083.*, 1979.

[6] John W. Hamblen Thomas E. Hull Thomas A. Keenan William B. Kehl Edward J. McCluskey Silvio O. Navarro Werner C. Rheinboldt Earl J. Schweppe William Viavant William F. Atchison, Samuel D. Conte and David M. Young.

Curriculum 68: Recommendations for academic programs in computer science: A report of the acm curriculum committee on computer science. *Commun. ACM 11, 3 (March 1968), 151–197.*, 1986.

Supercharging Python Scripting Education with ChatGPT! – How I use ChatGPT in my advanced python scripting class*

Conference Tutorial

Pak Kwan
School of Computing and Analytics
Northern Kentucky University
1 Nunn Drive, Highland Heights, KY 41099
kwanp1@nku.edu

Abstract

In recent years, the integration of artificial intelligence (AI) technologies into education has emerged as a promising approach to enhance student learning experiences and outcomes. This tutorial aims to explore how ChatGPT [1] can be effectively utilized to teach advanced Python scripting at the college level. Through interactive demonstrations, discussions, case study and hands-on activities, participants will gain insights into the potential applications of ChatGPT in the classroom and learn practical strategies for integrating this cutting-edge technology into their curriculum.

Description

This tutorial will provide college educators with practical guidance on incorporating ChatGPT into their advanced Python scripting courses. The tutorial will begin with an overview of ChatGPT and its capabilities, followed by a discussion of the challenges faced in teaching advanced Python scripting topics using traditional methods. Through live demonstrations and interactive exercises, participants will learn how ChatGPT can be used to generate text, assist with code completion, and provide problem-solving assistance in Python

*Copyright is held by the author/owner.

scripting. The tutorial will also address best practices for utilizing ChatGPT in teaching, including strategies for maximizing student engagement and learning outcomes. As a case study, the presenter will share his own experience teaching advanced Python scripting with ChatGPT, highlighting successes, challenges, and lessons learned. Participants will have the opportunity to engage in discussions, ask questions, and share their own experiences and insights with integrating AI technologies into the classroom.

Intended Audience

This tutorial is designed for college-level educators, particularly those teaching advanced Python scripting courses or interested in incorporating AI technologies into their curriculum. Participants should have a basic understanding of Python programming and an interest in exploring innovative teaching methodologies.

Tutorial Structure

- Introduction to ChatGPT.
- Challenges in Teaching Advanced Python Scripting.
- Integrating ChatGPT into Python Education.
- Case Study: Experience Teaching Advanced Python Scripting with ChatGPT
- Live Demonstration: Using ChatGPT in Python Scripting
- Best Practices for Utilizing ChatGPT in Teaching
- Conclusion

Biography

Pak Kwan currently is working as an Enterprise Architect at GE Aerospace and adjunct faculty at Northern Kentucky University, previously worked for Fifth Third Bank and IBM. Current research Interests: Big Data, Data Analytics, Parallel Database, Machine learning, Large Display and Data visualization, Learning theories.

References

[1] *ChatGPT: ChatGPT.* `https:://chat.openai.com`.

Grading According to Specifications: A Tutorial and Discussion[*]

Conference Tutorial

David L. Largent
Department of Computer Science
Ball State University
Muncie, IN 47306
dllargent@bsu.edu

Abstract

To provide more equity, and help students learn and take more ownership of their learning, we will explore how you might utilize specifications (specs) grading in your courses. We will explore the basic tenets of specs grading, changes I made in my courses, my experiences doing so, and explore how your courses might benefit. Most instructors can implement specs grading with relatively little effort.

My experience is that learners sometimes have an insincere interest in some of their courses, and do not put forth the effort they could and should. They also tend to beg for missed points and depend on doing well on one assignment to make up for another. Sometimes, they simply don't understand what you want them to do. There are also equity issues with points-based grading. Specifications grading provides an answer to these issues.

Nilson discussed fifteen criteria for evaluating grading systems, including upholding high academic standards, reflecting learning outcomes, motivating learners to learn and excel, discouraging cheating, and making learners feel responsible for their grades, to list a few [1]. She claims Specifications Grading better satisfies those criteria, as compared to point-based grading. Specs grading's basic tenets are to provide a very clear, detailed description of what is expected of the learner (specifications), and then evaluate their work as complete/incomplete against those specifications. Providing clear expectations can make a significant difference in a learner's ability to submit good work.

[*]Copyright is held by the author.

This complete/incomplete approach is more authentic to what learners will experience outside of academia. Employers expect a complete task, and do not think in terms of awarding (partial) points for work.

Instructional Context

I have applied specifications grading to many courses which included some or all of these characteristics in a variety of combinations:

- In-class discussions and pair/small group activities
- Group presentations
- Writing assignments
- Exams
- Final project
- First and advanced programming courses
- Programming projects competed individually, and in pairs/small groups
- Major and non-major courses
- First year to seniors

I started utilizing specs grading in 2017. Nilson claimed that it can restore rigor, motivate learners, and save faculty time [1]. I felt if even one of those claims were true, the approach was worth exploring further. I tried it in one course and now use it in all courses I teach.

Historically, I assigned points to everything learners did for a course, and then determined their course grade based on the percentage of points they earned. A learner could do poorly on some assignments, and great on other assignments, to offset the lack of points on the poor assignment. This generated an "average course grade," which did not necessarily reflect what the learner accomplished or at what level they learned course material. Learners asked for partial credit, and I gave it when evaluating their work.

To implement specs grading, I converted my assessments to complete/incomplete—they met the specifications, or they did not—and a few to tiered specifications translating to a letter grade. I provided limited "oops bits" to resubmit work. I determined the course grade based on how many of each assessment category a learner completed.

Tutorial learning outcomes

After participating in this tutorial, learners will be able to...

- Describe specifications grading's basic tenets
- Identify what others have tried that worked (or did not)
- Evaluate and incorporate specifications grading into their courses

References

[1] Linda Nilson. *Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time.* Stylus Publishing, LLC, Sterling, VA, 2015.

Analyzing the Contents of AI Benchmarks using Python*

Conference Tutorial

Bill Nicholson
Lindner College of Business Information Systems program
University of Cincinnati
Cincinnati, OH 45221
nicholdw@ucmail.uc.edu

Generative AI models are often evaluated through the use of benchmarks. These benchmarks consist of thousands of question/answer challenges across a wide variety of topics and sources. Some benchmarks are derived from Internet sources such as WikiHow.com. Some are composed by researchers and some are generated by AI. In this tutorial, we will make a deep dive into several popular benchmark datasets to learn more about contents and structure. We will create Python projects to analyze reading level, sentence structure, word frequency, and other metadata. The author's Materials can be found here: https://github.com/nicomp42/CCSCMW2024

Biography

Bill Nicholson is an Assistant Professor in the Information Systems program at the Lindner College of Business, University of Cincinnati. His research spans multiple areas, including databases, software engineering, programming education, and computational ontologies.

*Copyright is held by the author/owner.

Stopping To Teach the Halting Problem*

Nifty Assignment

Edward Aboufadel[1], William Dickinson[2]
[1]Department of Mathematics,
Grand Valley State University, Allendale, MI 49401
aboufade@gvsu.edu
[2]Department of Mathematics,
Grand Valley State University, Allendale, MI 49401
dickinsw@gvsu.edu

This nifty assignment is one that we have used for many years in our Theory of Computation course. This course covers deterministic and non-deterministic finite automata, push down automata, Turing machines, and explores the limits of computation including decidability. The key to proving the halting problem is to suppose the existence of a decider, H, that doesn't loop when given any pair (M, w) of a Turing machine as input and accepts if and only if M accepts w. H is then used as a subroutine in a clever Turing machine, D, and using an encoding of D as an input to D leads to a contradiction. Reasoning about these hypothetical Turing machines and understanding the resulting contradiction requires many levels of abstraction and is challenging for many of our students. To help our students, prior to getting to the proof of the halting problem, we use an assignment that has students code a finite automaton, F, that decides a particular language. We believe that this assignment makes the central idea of the proof of the halting problem more concrete and helps them understand the proof.

*Copyright is held by the author/owner.

Checksums Everywhere[*]

Nifty Assignment

James K. Huggins
Department of Computer Science
Kettering University
Flint, MI 48504
jhuggins@kettering.edu

Checksum validation is a real-world problem which can be adapted for use early in CS1. The structure of the assignment is relatively simple: given a number that contains a check digit, decompose the number into its individual digits, then perform a simple calculation on those digits to verify that the original number is valid.

Because a given checksum validation system is applied to a number with a fixed number of digits (*e.g.* the 16 digits of a credit card number), students in CS1 can write solutions that decompose the number into individually-stored digits (*e.g.* variables named "digit1", "digit2", *etc.*), then perform the required calculation on those individual digits.

For a numerical (integer) input, the only significant programming language features required are basic mathematical operations and conditional statements. If the input to the algorithm is encoded as a string, additional string operations would also be required to deconstruct the string into individual characters.

The wide variety of problem domains using checksums (e.g. credit card numbers, bank routing numbers, ISBN numbers) allows instructors to adapt the same basic assignment to different domains in successive terms, discouraging students from plagiarizing old submissions. Students can test their solutions not only on data supplied by the instructor, but on publicly-available data available to them in their own lives. External source materials are more likely to provide solutions using advanced language features (*e.g.* loops, arrays), providing defense against academic dishonesty.

[*]Copyright is held by the author/owner.

Unusual Standards*

Nifty Assignment

James K. Huggins
Department of Computer Science
Kettering University
Flint, MI 48504
jhuggins@kettering.edu

A classic example often used in early CS1 courses is temperature conversions between Fahrenheit and Celsius scales. The conversion requires a small calculation, and involves data values commonly used by US students. Of course, since this example is commonly used in instruction, it's unlikely to be usable as an assignment for students to independently complete.

Conversion between other common units of measure (e.g. the imperial and SI systems) is a natural extension of this exercise; of course, there are limited examples in common use. However, there are examples of "uncommon" units of measure that provide opportunities for simple exercises, often with a "nifty" fun component. We discuss two such "uncommon" units of measure: the "smoot" (a unit of length), and "furlongs per fortnight" (a unit of velocity).

A viable solution to these conversion problems requires relatively few features from modern programming languages: input, output, and mathematical operations. Depending on the environment, students may need help in implementing certain operations (e.g. "rounding up" instead of "rounding down" or "rounding to nearest").

There are a wide variety of "unusual" measurement systems available. This allows instructors to adapt the same basic assignment to different domains in successive terms, discouraging students from plagiarizing old submissions. Additionally, most of the "unusual" measurement systems have a humorous component to their definitions.

*Copyright is held by the author/owner.

AI as a Learning Tool for Introductory Programming[*]

Nift Assignment

James Roll
Department of Computer Science
University of Findlay
Findlay, OH 45840
rollj@findlay.edu

The goal of this assignment is to introduce introductory programming students to using generative AI tools like Claude and ChatGPT to help them in learning introductory programming. Students are shown how they can use AI tools to help explain basic programming concepts, decode cryptic error messages, explain why a program isn't working, and find syntax errors in and suggest fixes. Students are also encouraged to avoid using AI Tools to fully write programs at this point in their education, and introduced to the limitations generative AI tools for programming. This version of the assignment was written for an introductory Java programming course, but could easily be adapted to other programming languages.

[*]Copyright is held by the author/owner.

In Support of Equitable Grading Practices: A Tool to Auto Generate Multiple Versions of Paper Quizzes*

Nifty Assignment

Amy C. Larson
Department of Mathematics, Statistics, and Computer Science
Augsburg University, Minneapolis, MN 55454
`larsonam@augsburg.edu`

Weekly quizzes can provide both formative and summative assessment, and potentially increase student success. As summative, it is helpful if students are given multiple opportunities to pass, which is part of the philosophy of mastery learning and equitable grading practices. Students show mastery of a particular concept by answering all questions completely and correctly, and receive points only when they have shown mastery of a topic. Multiple attempts is great for students, but it can be challenging for the instructor. On one hand, this process is made easier because the grading is all or nothing (I put a star at the top or I circle what is incorrect and I do not deliberate over points). The gradebook states full points or "not yet" in the spirit of a growth mindset. On the other hand, it can be hard to write multiple versions of a quiz (on top of the multiple versions written to get around students sharing answers).

To mitigate the burden of writing multiple versions of a quiz, I have created a Latex framework to auto generate quiz versions with relative ease. As a user, you create a Latex file for each question. In that file, you define the preamble to the question, the postamble, and as many versions as needed for that particular question. Finally, you create a setup file that lists the question files and defines quiz versions based on the recombination of question versions. The code is heavily commented for transparency and for ease of modification.

In addition to the framework, I have posted several example mastery learning quizzes (with multiple versions) for an introductory programming course

*Copyright is held by the author/owner.

in Python. This Latex code and example mastery learning quizzes are freely available at https://github.com/AugsburgCS/quizwriter.

Building Strong Programming Skills with Weak Encryption

Nifty Assignment

Amy C. Larson
Department of Mathematics, Statistics, and Computer Science
Augsburg University, Minneapolis, MN 55454
larsonam@augsburg.edu

Password encryption is important for cybersecurity and it is relatable and relevant for students. That familiarity was leveraged to create two cybersecurity programming assignments – one in Python for *Introduction to Programming* and one in Java for *Data Structures*. As part of the assignment, students learn generally about encryption, securing passwords, and the vulnerability to hacking, which provides an opportunity to discuss personal, professional, and corporate responsibility with respect to information security. The assignment in both courses mimics a login process whereby a user enters their username and password, then those credentials are verified by hashing the password and comparing it to the associated stored (hashed) password. The "hashing" follows the spirit of encryption, but it is in no way usable for real password protection. However, the hashing process has educational value in the programming skills that students practice, in understanding number representation, and in gaining familiarity with the terms and process of password protection.

In the introductory course, students start the assignment by completing a worksheet to practice converting between decimal, binary, and hexadecimal, which they use as part of the hashing. In both courses, for the programming, they modify the user-entered password (a string) in a variety of ways as part of the hashing, gaining experience with string manipulation, ASCII, Unicode, bitstreams, conversions from binary to hex, and from hex to ASCII. User information is stored in a file, giving students practice with file I/O and data structures. Students store and retrieve data in both arrays and dictionaries, highlighting the difference between linear search and direct access. In the Java version for *Data Structures*, students use int[ ] arrays, as well as *ArrayList*, *LinkedList*, and *HashMap*, which are some of the data structures they implement as part of the course. Also, the login process is done through a pop-up window using Java graphics, therefore students learn a little about graphics and event-driven programming. Students are given a week to complete the programming assignment. In the Python course, it is assigned towards the end of the semester, and in the Java course, it is the first assignment that introduces them to the topic. All materials are available at https://github.com/AugsburgCS/cybersecurity.

A Comparative Analysis of Student Progress: Traditional Tools vs. Competency-Based Education in CS 1310 Foundations of Computer Science at Western Michigan University*

Wassnaa Al-Mawee
Computer Science Department
Western Michigan University
Kalamazoo, MI 49008-5466
wassnaa.al-mawee@wmich.edu

Abstract

This study investigates the efficacy of competency-based education (CBE) compared to traditional assessment methods in promoting student progress within the CS 1310 Foundations of Computer Science course, which focuses on discrete mathematics, at Western Michigan University. While traditional tools typically rely on standardized tests, grades, and fixed timelines, CBE prioritizes skill mastery and individualized learning pace. Through an analysis of student outcomes, this research demonstrates that CBE yields excellent advancement for students in this specific course. The findings illuminate the effectiveness of CBE in supporting deeper learning and skill acquisition within the context of computer science education at Western Michigan University.

1 Introduction

Education systems worldwide have long relied on traditional methods of assessment to measure student progress and success. However, as the landscape

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

of education evolves, there is a growing recognition of the limitations of traditional tools in effectively gauging students' true abilities and encouraging holistic learning. In response to these challenges, competency-based education has emerged as an innovative approach that prioritizes mastery of skills and competencies over seat time and grades. This paper aims to compare the efficacy of traditional assessment tools with competency-based education in evaluating student progress within the same class over two semesters.

The CS 1310 Foundations of Computer Science course at Western Michigan University provides an introduction to the theoretical underpinnings of computer science. Spanning over 4 credit hours, this course explores fundamental concepts such as finite automata, context-free grammars, sets, functions, relations, proof techniques, graphs and trees, sequences, counting, and probability. Through rigorous study and application, students gain a deep understanding of the theoretical foundations that underpin modern computing systems. The course serves as a gateway to more advanced topics in computer science and equips students with essential skills for success in the field.

However, a significant problem arises when using traditional assessment tools. While students may pass the course, they often do so without demonstrating mastery of specific foundational topics such as formal logic, proofs, mathematical induction, and counting. This lack of mastery can restrict students' ability to apply these critical concepts in practical settings and may ultimately limit their success in advanced coursework and professional endeavors. Thus, there is a pressing need to explore alternative assessment methods, such as competency-based education, to ensure that students not only pass the course but also acquire the necessary skills and competencies for future success in computer science.

In the following section, I will discuss relevant works on Competency-based Education (CBE) in Computer Science courses and research contributions aimed at addressing these concerns.

2 Literature Review

Research into educational methodologies has shown a growing interest in competency based education (CBE) and its potential to transform traditional learning environments. A key focus has been on the flexibility and personalization that CBE provides, allowing students to progress at their own pace by mastering specific skills and competencies [3]. This section discusses prior studies and their findings in the context of CBE and traditional assessment methods in the computing field. Several studies have explored the application of CBE in computer science education, examining its impact on student learning, engagement, and retention of knowledge. A study by Smith and Jones demonstrated that

CBE in introductory computer science courses led to higher student satisfaction and a deeper understanding of fundamental concepts compared to traditional assessment methods [7]. This was attributed to the flexible learning paths and emphasis on skill mastery inherent in CBE programs.

Similarly, Lee and Brown found that students in a CBE-based computer science program were more likely to retain knowledge and apply it in real-world scenarios [5]. The personalized approach in CBE, where students advance upon demonstrating competence, was credited with encouraging critical thinking and problem-solving skills. Additionally, a study by Davis and Nguyen noted that CBE encouraged self-directed learning and improved student motivation in advanced computer science courses, indicating that the approach's benefits extended beyond introductory levels [2].

A more recent study by Thomas and Mitchell examined the impact of CBE on collaborative learning in computer science [8]. The researchers found that CBE promoted teamwork and peer-to-peer learning, creating a more supportive and cooperative classroom environment. This collaborative aspect of CBE has been highlighted as a key advantage in modern education, especially in fields like computer science, where teamwork is crucial.

Traditional assessment methods, while still prevalent in many educational settings, have been criticized for their limitations. A study by Patel et al. pointed out that traditional methods, which rely heavily on standardized testing and fixed timelines, often fail to account for individual differences in learning styles and rates of progress [6]. This can lead to uneven grade distributions and a lack of engagement among students who do not excel under conventional assessment pressures.

Similarly, Johnson et al. found that traditional assessment in a computer science context could create undue stress and hinder creativity [4]. The rigid structure and emphasis on memorization were identified as potential drawbacks, suggesting that traditional methods might not be the most effective for promoting long-term retention and deeper learning.

Another study by Carter and Williams observed that traditional assessment methods in computer science often resulted in a high rate of attrition, as students who struggled with fixed timelines and standardized tests were more likely to drop out [1]. This raised concerns about the inclusivity and adaptability of traditional approaches in modern education.

The existing body of research highlights the advantages of CBE over traditional assessment methods in promoting student success and engagement. These studies provide a foundation for our comparative analysis, which focuses on the CS 1310 Foundations of Computer Science course at Western Michigan University. By examining both approaches in a single course over two semesters, this study adds to the growing evidence that CBE offers a more flexible and effective framework for student learning. Furthermore, this study contributes uniquely by evaluating CBE within WMU's specific educational context and curriculum structure, thereby assessing how CBE principles translate into practical benefits tailored to the institution's goals.

The remainder of the paper is organized as follows: Section Three describes the CS 1310 course. Section Four discusses the traditional tools used in this study. Following this, Section Five outlines the components and measurements of competency-based education. Section Six provides the comparative analysis, while Section Seven reveals the study's results. Finally, Section Eight concludes the paper and suggests directions for future work.

3 CS 1310 Foundations of Computer Science

CS 1310 Foundations of Computer Science serves as a cornerstone course within the curriculum at Western Michigan University, offering students an essential introduction to the theoretical underpinnings of computer science. This course spans four credit hours and covers a comprehensive array of fundamental concepts that are integral to understanding modern computing systems.

Throughout CS 1310, students engage with a diverse range of topics, including finite automata, context-free grammars, sets, functions, relations, proof techniques, graphs and trees, sequences, counting, and probability. By examining these theoretical foundations, students not only develop a deeper understanding of the fundamental principles of computer science but also acquire essential problem-solving and analytical skills that are invaluable in the field.

CS 1310 provides students with a solid theoretical framework upon which they can build their knowledge and expertise in subsequent courses. Moreover, the course lays the groundwork for advanced topics in computer science, paving the way for students to explore specialized areas of interest and pursue diverse career paths within the field.

In the context of this research, course delivery is facilitated through the utilization of eLearning platforms, with D2L being the developer of the Brightspace learning management system. Brightspace is a cloud-based software suite widely employed by educational institutions and businesses to facilitate online and blended classroom learning experiences. Competencies as learning outcomes can be implemented using Brightspace and linked to exams and quizzes questions. This section outlines the course design and structure employed for the selected two semesters, each consisting of 16 weeks, within the realm of CS 1310 Foundations of Computer Science.

3.1 Modular Course Structure

The course is structured into 16 weeks, with each week being designated as a module on the eLearning platform. This modular approach allows for organized content delivery and facilitates focused learning objectives for each instructional period.

3.2 Topics-Based Modules

Within the eLearning platform, each module is associated with a specific topic relevant to the CS 1310 curriculum. These topics align with the overarching learning objectives of the course and are carefully designed to provide students with a comprehensive understanding of foundational concepts in computer science.

3.3 Utilization of ZyBooks

ZyBooks serves as the primary educational resource for the course, offering interactive learning materials and engaging activities tailored to the CS 1310 curriculum. Through ZyBooks, students have access to dynamic content, including interactive exercises, animations, and self-assessment tools, enhancing their learning experience and promoting active engagement with course materials.

3.4 Blended Learning Approach

The course design incorporates elements of both synchronous and asynchronous learning, accommodating diverse learning preferences and schedules. While asynchronous components allow students to engage with course materials at their own pace, synchronous activities, such as live lectures or discussions, provide opportunities for real-time interaction and collaborative learning experiences.

3.5 Assessment and Feedback Mechanisms

Assessment strategies within the course are designed to evaluate students' comprehension and mastery of course content effectively. Formative assessments, such as quizzes and assignments, are integrated into each module to provide ongoing feedback and measure student progress. Additionally, summative assessments, including exams, assess students' overall understanding and proficiency in key concepts.

3.6 Course Outcomes Assessment

The creation of the EAMU vectors utilized for course assessment employs the following grading metric universally:

- Excellent corresponds to grades ranging from $(\infty$, 90%].
- Adequate corresponds to grades ranging from (90%, 75%]
- Minimal corresponds to grades ranging from (75%, 60%]
- Unsatisfactory corresponds to grades ranging from (60%, 0%]

4 Traditional Assessment Tools

Traditional assessment tools typically involve standardized tests, quizzes, homework assignments, and final exams to evaluate student learning. Grades are often assigned based on the percentage of correct answers or completion of tasks within a specified time frame. This approach assumes that all students learn at the same pace and achieve mastery of content within a predetermined timeframe. However, it fails to account for individual differences in learning styles, interests, and prior knowledge. Moreover, traditional assessments often prioritize memorization of facts over the development of critical thinking, problem-solving, and practical skills.

In the context of the CS 1310 Foundations of Computer Science course, these assessment components play a vital role in gauging students' understanding of key concepts and their ability to apply them effectively. Here's how each of these traditional assessment tools is utilized:

4.1 Assignments

Assignments are an essential component of the assessment process, providing students with opportunities to demonstrate their comprehension of course material and problem-solving skills. In CS 1310, students are typically assigned 11 assignments throughout the semester, each focusing on specific topics or tasks relevant to the course curriculum. These assignments may include problem sets, or written analyses, allowing students to apply theoretical concepts to real-world scenarios.

4.2 Participation Activities

Participation activities are designed to encourage active engagement and collaboration among students during class discussions or online forums. In CS 1310, students are expected to participate in 11 designated activities throughout the semester, contributing their insights, asking questions, and sharing their perspectives on course topics. Participation activities provide valuable opportunities for students to exchange ideas, clarify concepts, and deepen their understanding of course material.

4.3 Quizzes

Quizzes serve as formative assessments to assess students' ongoing progress and comprehension of course material. In CS 1310, students may be administered 11 quizzes throughout the semester, each covering specific topics or units of study. Quizzes may be conducted in-class and online via eLearning and typically consist of multiple-choice questions and short answer questions. Quizzes provide students with timely feedback on their learning progress and help instructors identify areas where additional instruction or support may be needed.

4.4 Exams

Exams are summative assessments designed to evaluate students' overall understanding and mastery of course material. In CS 1310, students are typically required to take three exams over the course of the semester, including two midterm exams and a final exam. These exams assess students' knowledge of key concepts, problem-solving abilities, and critical thinking skills. Exams may include a mix of multiple-choice questions, short answer questions, and longer-form problems, allowing students to demonstrate their proficiency across a range of topics covered in the course.

5 Competency-Based Education (CBE)

In contrast, competency-based education focuses on identifying specific skills and competencies that students need to master to succeed in their chosen field. Instead of progressing through a predefined curriculum at a set pace, students advance upon demonstrating mastery of each competency. This approach allows for personalized learning experiences tailored to individual student needs, strengths, and interests. CBE provides timely feedback to support continuous improvement. By focusing on mastery rather than grades, CBE promotes deeper learning and long-term retention of skills.

In a competency-based education framework, ensuring that students demonstrate a thorough understanding of each competency, referred to as a learning outcome (LO), is paramount. To achieve this, it is essential to set pass thresholds that reflect a minimum level of proficiency. In this section, I will outline the adjusted pass thresholds for each LO in the CS 1310 Foundations of Computer Science course, ensuring that students must attain at least 70% mastery to pass as shown in Table 1.

- **LO 1**: Proof Techniques
  - Demonstrate the ability to correctly identify the use of modus ponens and modus Tollens in given arguments.
  - Demonstrate the ability to correctly prove theorems using direct proof, indirect proof, proof by contrapositive, proof by contradiction, proof by cases, proofs of equivalence, existence proofs, uniqueness proofs, and proof by counterexample.
- **LO 2**: Mathematical Induction
  - Apply mathematical induction to prove simple mathematical statements and properties.
  - Formulate and execute complete proofs using mathematical induction for more complex mathematical problems.
  - Demonstrate the understanding of the base case, inductive hypothesis, and inductive step in mathematical induction proofs.
- **LO 3**: Combinatorics and Counting Techniques
  - Solve combinatorial problems involving permutations and combinations.
  - Calculate probabilities of various events using counting techniques.
- **LO 4**: Discrete Structures
  - Identify and differentiate between different types of discrete structures such as sets, trees, and graphs.
  - Apply discrete structures to model and solve real-world problems.
  - Construct and manipulate various types of trees and graphs to analyze data and relationships.
- **LO 5**: Mathematical Relations
  - Define and interpret different types of mathematical relations, including equivalence relations and partial orders.
  - Apply relational concepts to compare and analyze elements in various mathematical contexts.
  - Demonstrate the ability to determine the properties and characteristics of relations in given scenarios.

In the assessment of competencies within the CS 1310 Foundations of Computer Science course, a decaying average approach is employed to provide a detailed understanding of student proficiency over time. For example, students' performance on mathematical induction tasks is continuously assessed throughout the semester, with greater emphasis placed on recent assessments. By applying decaying weights to assessment scores, the measurement of mathematical induction competency becomes more sensitive to changes in student performance over time, providing a comprehensive view of proficiency levels.

Table 1: Adjusted Pass Thresholds for Competency-Based Assessment Components

Learning Outcome	Pass Threshold
LO 1	70%
LO 2	70%
LO 3	70%
LO 4	70%
LO 5	70%

6 Comparative Analysis

To compare the effectiveness of traditional assessment tools with competency-based education, I conducted a study over two different semesters: Fall 2022 and Spring 2024, with class sizes of 25 and 21 students, respectively. In Fall 2022, a traditional curriculum was used, emphasizing time-based assessments such as standardized tests, quizzes, and assignments. By contrast, in Spring 2024, the approach shifted to a competency-based education (CBE) approach, allowing students to progress at their own pace, contingent on demonstrating mastery of specified skills and competencies. To ensure a fair comparison, similar sets of questions were used in exams, quizzes, and assignments across both semesters.

Table 2 displays the EAMU vectors for each learning outcome in Fall 2022. The EAMU was calculated by averaging the scores from various quizzes and exam sections.

Table 2: EAMU Vectors for Learning Outcomes in Fall 2022

Learning Outcome	E	A	M	U
LO1	13	5	2	5
LO2	7	8	3	6
LO3	7	10	3	5
LO4-SETS	11	7	3	4
LO4-TREES	5	8	7	5
LO4-GRAPHS	14	5	3	3
LO5	14	5	3	3

In addition to the EAMU vectors, I also analyzed the grade distribution for students in Fall 2022 to better understand the spread of academic performance under traditional assessment methods. Figure 1 below shows the distribution of final grades for that semester, providing insight into the range of student outcomes. This grade distribution chart helps visualize how traditional assessments, such as standardized assignments, quizzes, and exams, influence the overall grading pattern.

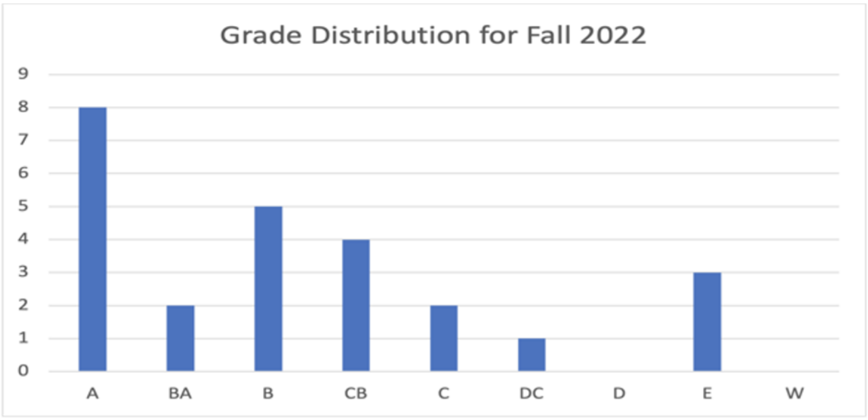


Figure 1: Grade Distribution for Fall 2022 (Traditional Assessment).

Similarly, I examined the EAMU vectors for each learning outcome during Spring 2024, focusing on a competency-based education (CBE) approach. This approach allowed students to progress at their own pace by demonstrating mastery of specific skills and competencies. Table 3 shows the EAMU vectors for each learning outcome in Spring 2024, calculated using a decaying average of scores from quizzes and exam questions.

Table 3: EAMU Vectors for Learning Outcomes in Spring 2024

Learning Outcome	E	A	M>70%	U
LO1	20	0	1	1
LO2	9	0	11	1
LO3	20	1	0	0
LO4-SETS	21	0	0	0
LO4-TREES	21	0	0	0
LO4-GRAPHS	21	0	0	0
LO5	21	0	0	0

For Spring 2024, which used a competency-based education (CBE) approach, I examined the grade distribution to understand how this method impacted student outcomes compared to traditional assessment. Figure 2 below shows the grade distribution for Spring 2024, reflecting the CBE approach that allows students to progress at their own pace by demonstrating mastery of specific skills and competencies.

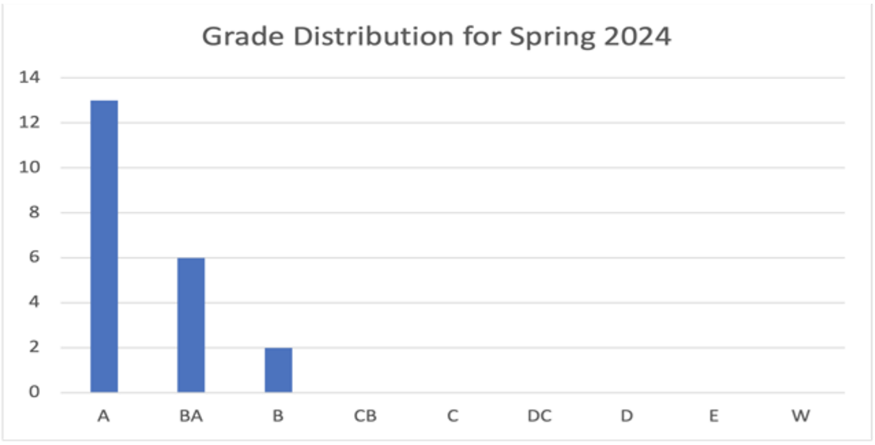


Figure 2: Grade Distribution for Spring 2024 (Competency-Based Education).

7 Results

The results of the study revealed significant differences in student mastery of learning outcomes and grade distributions between the traditional assessment group and the competency-based education (CBE) group. Students in the traditional assessment group showed a generally steady but uneven progress pattern across the Fall 2022 semester, with some demonstrating mastery in specific areas while others struggled to keep pace. The grade distribution chart for this group indicated a wide range of grades, suggesting that traditional methods might not consistently promote deep learning or accommodate diverse learning styles.

In contrast, the CBE group exhibited more consistent growth and a higher level of mastery across the learning outcomes in the Spring 2024 semester. Students in this group progressed at their own pace, achieving competency in a structured but flexible manner. The grade distribution chart for the CBE group showed a narrower range of grades, with a more balanced distribution, suggesting that the competency-based approach provided a more consistent and equitable learning experience.

8 Conclusion

This study highlights the benefits of competency-based education (CBE) compared to traditional assessment methods in promoting student success and progress. By emphasizing skill mastery and allowing for flexible, personalized learning, CBE creates a more inclusive and motivating environment, particularly in foundational computer science courses. The findings suggest that CBE not only encourages deeper learning but also leads to a more consistent and balanced grade distribution, which contributes to higher levels of student engagement and satisfaction. This research further demonstrates how applying CBE principles within WMU's specific educational context and curriculum structure results in practical benefits that align with the institution's goals.

While traditional assessment tools have their place in education, they often emphasize fixed timelines and standardized testing, potentially overlooking the need for personalized learning experiences. Integrating CBE principles can bridge this gap, providing students with a more adaptable and individualized approach to learning. This can better prepare students for success in an ever-changing world, where skills and competencies are crucial.

Future research should incorporate qualitative feedback from students to provide a comprehensive perspective on the effectiveness of both approaches. Additionally, further studies are needed to explore the long-term impact and scalability of CBE across various educational settings and disciplines. These insights will help to refine and optimize educational methods, contributing to a more effective and student-centered approach to learning.

References

[1] M. Carter and L. Williams. "Attrition in Traditional Computer Science Assessment". In: *IEEE Transactions on Education* 42.3 (2019), pp. 180–185.

[2] K. Davis and T. Nguyen. "Self-Directed Learning in Competency-Based Computer Science Programs". In: *Journal of Education in Computing and Technology* 45.3 (2021), pp. 200–210.

[3] J. Gervais. "The operational definition of competency-based education". In: *The Journal of Competency-Based Education* 1.2 (2016), pp. 98–106.

[4] R. Carter J. Johnson and W. Moore. "Stress and Creativity in Traditional Computer Science Assessment". In: *Journal of Computer Science and Education* 47.2 (2021), pp. 120–128.

[5] C. Lee and D. Browns. "Competency-Based Education in Computer Science: Fostering Critical Thinking and Problem-Solving Skills". In: *International Journal of Computer Science Education* 33.1 (2019), pp. 89–97.

[6] M. Gomez R. Patel and P. Singh. "Challenges of Traditional Assessment in Computer Science Education". In: *IEEE Transactions on Education* 40.4 (2018), pp. 230–235.

[7] A. Smith and B. Jones. "The Impact of Competency-Based Education on Student Satisfaction in Introductory Computer Science Courses". In: *The Journal of Computer Science Education* 34.2 (2020), pp. 100–110.

[8] L. Thomas and S. Mitchell. "Collaboration and Competency-Based Education in Computer Science". In: *Proceedings of the ACM Conference on Computing Education* 12 (2022), pp. 150–158.

Leveraging Visualization for Cryptography Education *

Imad Al Saeed
Computer Sciences Department
Saint Xavier University
Chicago, IL 60655
alsaeed@sxu.edu
773-298-3393

Abstract

Cryptography stands as the fundamental pillar ensuring secure communication and safeguarding data in the modern digital era. Nevertheless, comprehending the intricate concepts and algorithms of cryptography poses a significant challenge for computer science and computer information systems students due to their abstract and complex nature. Visualization methods offer a pathway to facilitate practical exploration and experimentation, enabling students to engage dynamically with cryptographic algorithms and protocols. Tools like Cryptool2 furnish students with a simulated laboratory environment, empowering them to explore cryptographic algorithms, simulate potential attacks, and scrutinize security weaknesses. This paper delves into the pivotal role of visualization techniques in the education of cryptography, emphasizing their capacity to augment educational outcomes and foster a profound grasp of cryptographic principles within the framework of a cybersecurity curriculum.

1 Introduction

Cryptography, the science of secure communication and data protection, plays a pivotal role in ensuring the confidentiality, integrity, and authenticity of in-

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

formation in the digital age [1]. Understanding cryptographic concepts and algorithms is essential for students pursuing careers in cybersecurity, information technology, and related fields. However, cryptography can be a complex and abstract subject, presenting challenges for learners in grasping its intricacies. Traditional methods of teaching cryptography often rely on theoretical explanations and mathematical formulations, which may not fully engage students or facilitate deep comprehension. In recent years, there has been a growing recognition of the potential of visualization techniques to enhance cryptography education by providing intuitive and interactive representations of cryptographic concepts and processes [6]. Visualization offers a powerful means to elucidate abstract concepts by representing them visually in a comprehensible manner, facilitating deeper understanding and engagement among learners. In the context of cryptography education, visualization tools and techniques can be employed to illustrate fundamental cryptographic operations, algorithms, protocols, and attacks. Visualizations can take various forms, including diagrams, animations, interactive simulations, and graphical representations of cryptographic algorithms and protocols. These visualizations enable students to observe the inner workings of cryptographic algorithms, visualize data transformations, and understand the flow of information during encryption and decryption processes. By providing visual representations of abstract cryptographic concepts, educators can make complex topics more accessible and engaging for students, leading to improved learning outcomes. The effectiveness of visualization in cryptography education has been demonstrated in numerous studies and research projects. For example, Kim and Choi (2019) developed an interactive learning tool for teaching public-key cryptography, which employed visualizations to illustrate key exchange protocols and digital signature schemes [4]. Similarly, Simoens, Cornelis, and Preneel (2006) explored visualization tools for teaching cryptographic protocols, demonstrating their utility in conveying complex concepts to students [6]. This paper explores the utilization of Cryptool2 as a pedagogical tool for teaching cryptography. Cryptool2's intuitive user interface provides a user-friendly environment for students to explore cryptographic concepts through hands-on experimentation. The software encompasses a diverse array of cryptographic algorithms, from classical ciphers to modern symmetric and asymmetric encryption schemes, allowing learners to gain practical insights into their operation and security properties.

2 Cryptool2 as a visualization tool

As the importance of cryptography continues to grow, so does the need for effective educational tools to impart its fundamental principles and techniques. Cryptool2 emerges as a versatile and comprehensive platform for cryptography education, offering a range of features tailored to both beginners and advanced learners.

Cryptool2 facilitates interactive experimentation with cryptographic protocols and attacks, enabling students to analyze real-world scenarios and understand the vulnerabilities inherent in various cryptographic systems [2]. Through the simulation of cryptographic attacks, learners develop a deeper understanding of cryptographic weaknesses and the importance of robust security measures.

In addition, Cryptool2 offers educational resources such as tutorials, demonstrations, and documentation, empowering educators to integrate cryptography seamlessly into their curricula [4]. The platform's extensibility allows for the integration of custom plugins and modules, enabling instructors to tailor the learning experience to meet specific educational objectives and learning outcomes.

3 Cybersecurity Curriculum at SXU

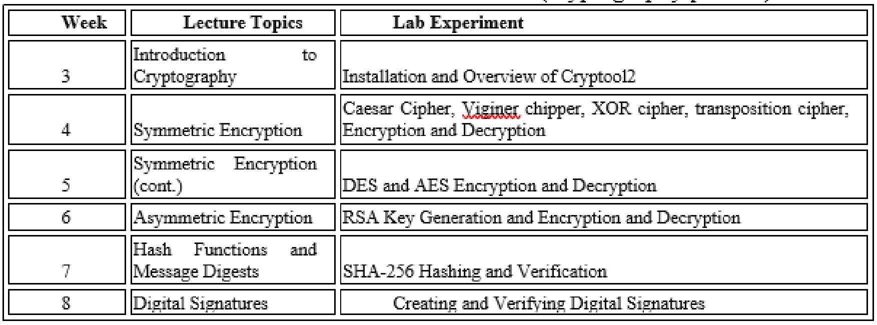
At Saint Xavier University, the Computer Science curriculum includes a pair of courses titled "Introduction to Cybersecurity I" and "Introduction to Cybersecurity II." These courses serve as comprehensive introductions to essential concepts, principles, and methodologies within cybersecurity. They provide students with a foundational understanding of key terminology, the evolving landscape of threats, and established cybersecurity frameworks. Additionally, these courses delve into cryptographic fundamentals, exploring principles, algorithms, and protocols crucial for safeguarding data and communications. Students engage with topics such as encryption methodologies, cryptographic algorithms, the intricacies of digital signatures, and the management of cryptographic keys. The Computer Science department further offers specialized courses in Network Security and Digital Forensics to supplement students' expertise in the field.

4 Methodology

Participants - In an Introduction to Cybersecurity I course, there are two cohorts of students: Group A (CT2) comprising 25 students enrolled in the Spring 2024 semester, and Group B (Traditional) comprising 25 students from the Fall 2023 semester. Both groups of students possess comparable academic backgrounds and possess equivalent prior familiarity with fundamental concepts in cryptography.

16-week cybersecurity course - Saint Xavier University's Computer Science program features a course titled Introduction to Cybersecurity I, integral to the Cybersecurity track within both the CS and CIS majors. Cryptography constitutes a significant segment of this course, spanning a six-week duration. A structured teaching approach is outlined in the table below to effectively impart knowledge in Cybersecurity:

Table1. six weeks of the course (cryptography portion)



Week	Lecture Topics	Lab Experiment
3	Introduction to Cryptography	Installation and Overview of Cryptool2
4	Symmetric Encryption	Caesar Cipher, Viginer chipper, XOR cipher, transposition cipher, Encryption and Decryption
5	Symmetric Encryption (cont.)	DES and AES Encryption and Decryption
6	Asymmetric Encryption	RSA Key Generation and Encryption and Decryption
7	Hash Functions and Message Digests	SHA-256 Hashing and Verification
8	Digital Signatures	Creating and Verifying Digital Signatures

Variables - The study examined several factors influencing learning outcomes, including students' majors within computer science and computer information systems subfields, the quantity of security courses completed, and their familiarity with open course virtual environments like Cryptool2. Additionally, the study considered the impact of time allocated to specific tasks as a potential influential variable in the learning process.

Hypothesis - the general hypotheses were:

H1: There is a hypothesis that suggests students instructed in cryptography using Cryptool2 attained superior average grades compared to peers taught via conventional approaches.

H2: There is an alternate hypothesis proposing that students educated in cryptography through traditional methods obtained inferior average grades relative to those instructed via conventional means.

5 The experiment

The objective of this study is to assess the efficacy of two instructional approaches—utilizing Cryptool2 (CT2) versus traditional methods—in teaching encryption and decryption within a classroom context. Cryptool2 offers an interactive platform enabling practical exploration of cryptographic concepts, whereas traditional methods generally rely on theoretical instruction and written assignments. The study aims to gauge students' understanding and retention of encryption and decryption principles under both methodologies. This investigation was conducted as outlined below:

5.1 Introduction to Cryptool2 (CT2) - Cryptool2 represents an open-source software platform crafted to streamline cryptographic experimentation and educational pursuits. It presents users with an intuitive interface alongside an extensive array of cryptographic algorithms, thereby empowering students to delve into encryption, decryption, key generation, and cryptographic protocols through practical engagement [5].

5.2 Setting Up Cryptool2 - The instructor provided step-by-step guidance to students for downloading and installing Cryptool2 onto their personal computers. The installation procedure is uncomplicated, ensuring accessibility for students possessing diverse levels of technical proficiency.

5.3 Exploring Cryptographic Algorithms - Within Cryptool2, the instructor showcased a range of cryptographic algorithms, including symmetric-key encryption (like AES, DES) and asymmetric-key encryption (like RSA, ECC). Through practical demonstrations using sample data, students were guided through the encryption and decryption procedures, offering insights into the functionality of these algorithms and their relative advantages and limitations [5].

5.4 Integration into Course Curriculum - Cryptool2 holds potential for inclusion within the curriculum of cybersecurity courses as a hands-on tool for imparting cryptographic concepts. The instructor seamlessly integrated Cryptool2 labs, exercises, and assignments into the coursework, providing students with opportunities to solidify theoretical understandings discussed in lectures and to implement their knowledge in real-world scenarios [2].

5.5 Assessment and Feedback - To gauge students' grasp of cryptography, the instructor employed Cryptool2-centric assignments and exercises. Feedback was offered on students' practical applications of cryptographic algorithms and protocols, aiding in skill enhancement and troubleshooting any challenges faced during their experimental endeavors.

6 Experimental Design

6.1 Group A (CT2): Students within this cohort are instructed in encryption and decryption techniques utilizing Cryptool2. They actively immerse themselves in practical exploration of cryptographic algorithms and protocols within the Cryptool2 interface. The instructor is readily available to offer guidance and support whenever required. See figure1 about the specific topic of your paper. Define any technical terms deemed to require clarification when they are introduced.

6.2 Group B (Traditional): Students within this group are educated in encryption and decryption via conventional means, encompassing lectures, reading materials, and written assignments. The instructor elucidated cryp-

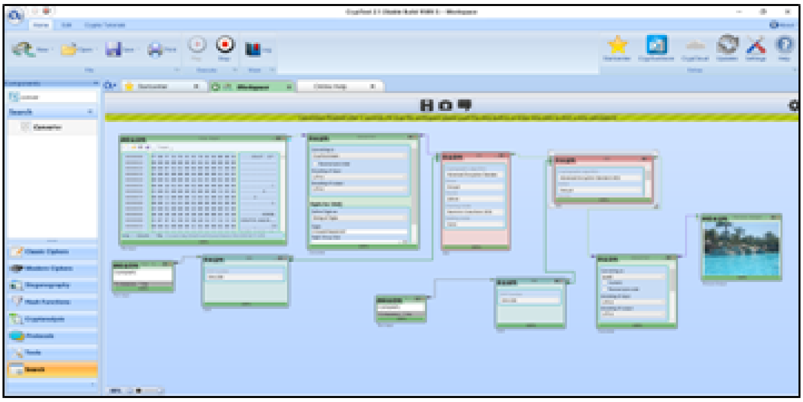
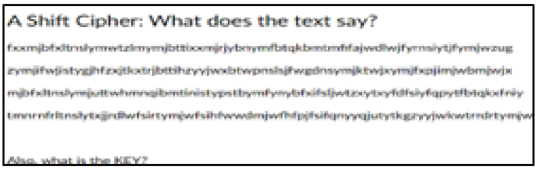


Figure1. Encrypt and decrypt an image using Cryptool2.

tographic concepts through theoretical frameworks and illustrative examples. See figure2 .


A Shift Cipher: What does the text say?

[illegible]

[illegible]

[illegible]

[illegible]

Also, what is the KEY?

Figure2. traditional way of Encryption and decryption.

6.3 Duration - The study extends over a period of six weeks, with each group receiving three hours of instruction per week. Weekly sessions focus on distinct cryptographic algorithms and methodologies, supplemented by practical exercises to reinforce learning.

6.4 Assessment - Following the six-week duration, students in each group undergo a comprehensive evaluation to gauge their comprehension of encryption and decryption fundamentals. This assessment encompasses theoretical inquiries, problem-solving challenges, and practical tasks concerning cryptographic algorithms and protocols.

6.5 Data Collection - Data was gathered from a variety of sources, including questionnaires, student feedback, the experimental study reflected by student grades and assessment performance, and post-surveys. Descriptive statistical analysis was conducted using SPSS software, whereby percentages and means were computed and incorporated into the analysis to enhance and elucidate quantitative findings. SPSS proves to be a robust tool due to its capacity to navigate intricate data relationships, rendering it a highly effective and efficient method for data analysis.

7 The result

To ensure impartiality, the lab assignments underwent assessment by two adjunct faculty members renowned for their extensive expertise in security matters. The findings from the experiment, contrasting the efficacy of learning encryption and decryption through Cryptool2 versus conventional methods, provide substantial insights into the comparative effectiveness of diverse instructional strategies within cryptography education. See the table below.

Table 1: Table2. Students' average grade for both groups.

Experiment Method	Average Grade (out of 100)
Cryptool2 (CT2)	85
Traditional	72

Firstly, the average grade of students in Group A (CT2), who learned encryption and decryption using Cryptool2, was substantially higher than that of students in Group B (Traditional), who were taught through traditional methods. The notable difference in average grades between the two groups suggests that the hands-on, interactive nature of Cryptool2 facilitated deeper comprehension and retention of cryptographic concepts among students. This finding aligns with existing literature, which highlights the benefits of experiential learning and active engagement in improving learning outcomes [3]. The hands-on experimentation with cryptographic algorithms and protocols within the Cryptool2 environment likely provided students in Group A with practical insights and real-time feedback, enhancing their understanding and mastery of encryption and decryption principles. By actively engaging with cryptographic concepts through experimentation, students were able to reinforce theoretical knowledge with practical experience, leading to higher levels of comprehension and performance.

In contrast, students in Group B, who learned through traditional methods such as lectures, readings, and written exercises, achieved lower average grades. This finding suggests potential limitations in the effectiveness of traditional instructional approaches in conveying complex cryptographic concepts. Theoretical explanations and written exercises may have been insufficient to fully engage students and facilitate deep comprehension of encryption and decryption principles, resulting in lower levels of retention and performance.

The disparity in average grades between the two groups underscores the importance of incorporating interactive, experiential learning tools like Cryptool2 in cybersecurity education. By providing students with hands-on opportunities to explore cryptographic concepts in a dynamic and interactive environment, Cryptool2 enables active learning and promotes deeper understanding, thereby enhancing student outcomes and proficiency in cryptographic skills.

Additional evidence was gathered to corroborate H1 and H2 through an online post-survey conducted via SurveyMonkey, coupled with class observations and presentations. The findings indicated that 90% of participants in Group A, as per the post-survey, affirmed that virtualization tools such as Cryptool2 significantly enhanced their learning capabilities, thereby validating H1. Moreover, 95% of participants in Group B expressed a strong interest, based on the online post-survey, in utilizing tools to overcome school security barriers and delve deeper into cryptography, thereby supporting H2. These results were further reinforced by 72% of students, as revealed in both post-surveys, advocating for additional coursework dedicated to cryptography, either as a core or elective course within the computer science department, bolstering the validity of the findings

8 Conclusion

The conclusion drawn from this experiment emphasizes the importance of incorporating interactive, experiential learning tools like Cryptool2 in cybersecurity education. By providing students with hands-on opportunities to explore cryptographic concepts in a dynamic and interactive environment, Cryptool2 enables active learning and promotes deeper understanding, thereby enhancing student outcomes and proficiency in cryptographic skills. Educators and institutions should consider integrating visualization and experiential learning techniques into cryptography education to enhance student engagement and comprehension. Further research is warranted to explore the long-term impact of interactive learning tools like Cryptool2 on student learning outcomes and to identify best practices for integrating such tools into cybersecurity curricula.

References

[1] Ross Anderson and Roger Needham. "Robustness principles for public key protocols". In: *Annual International Cryptology Conference*. Springer. 1995, pp. 236–247.

[2] J Fieglein and S Fischer-Hübner. "Towards a serious game for teaching symmetric cryptography". In: *European Conference on Games Based Learning*. Vol.2. 2018, pp. 194–202.

[3] H Janicke, H Reimer, and V Wulf. “A study on cryptographic knowledge of computer science students in German universities”. In: *International Conference on Cryptology and Network Security*. 2016, pp. 142–148.

[4] Y. Kim and J. Choi. “An interactive learning tool for teaching public-key cryptography.” In: *IEEE Access*. Vol.7. 2019, pp. 12243–12252.

[5] S. Schneider et al. “Cryptool2: Experiences, visions, and challenges”. In: *Proceedings of the 12th International Conference on Availability, Reliability and Security*. 2017, 77:1–77:6.

[6] P. Simoens, J. Cornelis, and B. Preneel. “Visualization tools for teaching cryptographic protocols.” In: *International Conference on Information Technology: Coding and Computing*. Vol.2. 2006, pp. 67–72.

Pedagogical Implications of Parser Combinators in Programming Languages Courses: A Comparative Study*

Abbas Attarwala[1], Pablo Raigoza[2]
[1]Computer Science Department
California State University
Chico, CA 95973
aattarwala@csuchico.edu
[2]Computer Science Department
Cornell University
Ithaca, NY 14850
pr428@cornell.edu

Abstract

This paper recounts the experience of teaching parser combinators in a programming language course using OCaml at both Boston University and California State University, Chico. The main focus is on how parser combinators are introduced when teaching parsing to students who are new to functional programming. Techniques such as boxes and color coding are employed to simplify the understanding of the concepts. Furthermore, teaching course evaluation data are presented to compare course outcomes, contrasting semesters when parser combinators were not used with those when they were incorporated into the teaching. Reflections and feedback from students provide insight into the effectiveness of these teaching methods. Additionally, a two-tailed Welch t-test is conducted on the teaching course evaluation data to assess the impact of using parser combinators.

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

This paper discusses the benefits of teaching parser combinators in a third-year programming language course. The box representations for the parser and the color coding for the parser combinators are introduced and explored, drawing on experiences from Boston University (BU) and California State University, Chico (CSU Chico). A common challenge that I[1] have observed is that students initially create ad hoc parsers based on the context-free grammars (CFG) provided for their projects. However, when project requirements evolve and a revised CFG is introduced in later parts of the project, these initial parsers do not scale well. As a result, students often face difficulties and must completely rewrite their parsing code to accommodate the new requirements. I propose that parser combinators could be a solution, potentially enabling students to develop parsers that are more adaptable and scalable. To evaluate this hypothesis, this paper compares and analyzes my teaching course evaluation data across various semesters, focusing on student performance and adaptability in courses taught with and without parser combinators.

In this paper, we define a parser as a function that takes a string as input and produces an output that is either a tuple consisting of the parsed value and the remaining unconsumed part of the string, or an indication that the parsing has failed. A parser combinator is a higher-order function that takes one or more functions or parsers as input and returns a new parser as output. It allows for the construction of complex parsers by combining simpler components in a modular and reusable way.

2 Literature Review

In our exploration of teaching methodologies for parser combinators, we have identified a notable gap in the existing literature. Although there is extensive documentation on the technical advantages and applications of parser combinators, their pedagogical aspects have been largely overlooked.

Parser combinators [10], offer functional programmers a clean and flexible method for constructing parsers. This flexibility is attributed to the abstraction they provide, distancing the programmer from complex parsing machinery. However, [10] also presents a trade-off: This abstraction comes with the cost of executing the combinators and the functions that build them, often necessitating repetitive execution. [6] mentions that parser combinators are used in parsing sequences generated by CFG, in specialized data formats like JSON and YAML, and markup languages such as XML and HTML. Their paper also

[1] First person in this paper refers to Abbas Attarwala

illustrates the use of parser combinators in programming language processing, specifically in identifying syntax errors.

Furthermore, [8, 9] emphasize the balance the parser combinators maintain between flexibility and abstraction. Parser combinators enable the creation of parsers in a style that remains close to the CFG. Highlighting the role of higher-order functions [7], along with [5, 10], points out the strengths in developing combinator libraries, particularly parser combinators. They underline the beauty of these abstractions in functional programming, but also note the scarcity of literature on their practical, maintainable, and scalable use.

Our research provides valuable insights into the pedagogical effectiveness of parser combinators. By conducting a comparative study of course evaluations from semesters with and without their use, we aim to demonstrate their impact on student learning. While the current literature thoroughly explores the technical strengths and applications of parser combinators, our research examines their pedagogical value, offering a different perspective in the realm of functional programming education.

3 Parser Combinators

In the Summer of 2020, while teaching CS 320 at BU my students built an interpreter for a stack-based programming language. They were provided with the initial CFG and a set of operational semantics. As the project progressed, new features were added, such as nested conditional statements. Initially, students created parsers using regular expressions or some complex parsing involving a stack, but these were ad hoc and struggled to adapt to the evolving grammar, leading to significant rewriting and frustration for both students and myself.

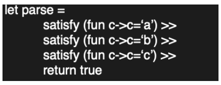
To address this, I integrated parser combinators into the curriculum, beginning at BU in the Fall of 2020 and continuing through Fall 2022 at CSU Chico. Before introducing this concept, I engaged students with a simple OCaml exercise involving string parsing to demonstrate the practical benefits of parser combinators. In this simple example, I ask my students to write an OCaml code to parse the first three characters of a string but only if it begins with 'a', followed by 'b' followed by 'c'. The code in OCaml is shown in Listing 1. I provide my students with `getFirstCharacter` function which accepts a `string` and returns back a `option` tuple of the extracted first character and the unconsumed string.

```
let parse s =
  match (getFirstCharacter s) with
  | None -> None
  | Some (firstC, rest) -> if firstC = 'a' then
      (match (getFirstCharacter rest) with
      | None -> None
      | Some (secondC, rest) -> if secondC = 'b' then
        (match (getFirstCharacter rest) with
        | None -> None
        | Some (thirdC, rest) -> if thirdC = 'c' then
          Some (true, rest)
          else None)
        else None
      else None
```

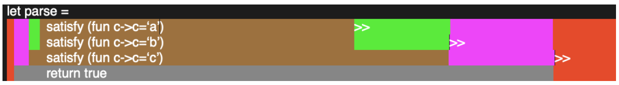
Listing 1: Parsing code without using parser combinators

Students quickly realize that while the initial code works, it's not scalable and becomes cluttered, especially with numerous error checks for parsing characters other than ‘a’, ‘b’, or ‘c’. To address these issues, I guide them through refactoring the code using parser combinators.

I define a parser as a function that accepts a `string` and returns an `(‘a, string) option` type in OCaml. The `option` type indicates that the function returns `None` if parsing fails, or `Some` if parsing is successful. In the case of success, the `Some` tuple contains two elements: the parsed value (represented by the type variable `‘a`) and the remaining unconsumed string. I represent parsers as boxes: the input is a `string` and the output is an `(‘a, string) option` as seen in Figure 2. Another way to think about parser combinators is like a glue that combines two parsers to create a new parser. The ≫ operator is a parser combinator, commonly referred to as the sequencing operator, which I implement during my lecture. It not only links two parsers—`p1` and `p2`—to form a new parser `p3` (`let p3 = p1 ≫ p2`), but also establishes a dependency where `p2` runs only if `p1` succeeds, passing the unconsumed string from `p1` to `p2`. The type of ≫ is defined as `’a parser ->’b parser-> ’b parser`. Following OCaml's operator naming conventions, the operator ≫ associates to the left. Figure 1 shows the refactored code that parses the first three characters of a string, specifically ‘a’, ‘b’, and ‘c’, using this method.



```
let parse =
    satisfy (fun c->c='a') >>
    satisfy (fun c->c='b') >>
    satisfy (fun c->c='c') >>
    return true
```



```
let parse =
    satisfy (fun c->c='a')        >>
    satisfy (fun c->c='b')            >>
    satisfy (fun c->c='c')                >>
    return true
```

Figure 1: Refactored OCaml code using parser combinator. The same code on the right is color coded to represent each parser. The red parser for instance is a sequence of the purple parser followed by the grey parser. satisfy, ≫, and return are parser combinators.

I demonstrate to my students that `satisfy` is a combinator that verifies if a string starts with a specific character. For example, `satisfy (fun c -> c = ‘a’)` returns a parser that checks whether the string begins with ‘a’. If this check passes, `satisfy (fun c -> c = ‘b’)` checks for ‘b’, followed by `satisfy (fun c -> c = ‘c’)` checking for ‘c’. This sequential checking, facilitated by the » operator, simplifies error handling. The » operator inherently handles errors, so if any parser in the sequence fails, the entire parsing process fails. Unlike the explicit error checks in Listing 1, parser combinators allow students to focus on parsing the required elements without worrying about extensive error handling for intermediate steps.

In Figure 1, I also use color coding for each parser, which students refer to in Figure 2 to visualize how the » operator unpacks the unconsumed string and feeds it to the next combinator in the chain. In the color-coded diagram, the first » in green sequences the two brown parsers to create the green parser. The second » in purple sequences the green parser with the next brown parser to create the purple parser. Finally, the third » in red sequences the purple parser with the grey parser to create the red parser. The red parser returns true if the string begins with ‘a’, followed by ‘b’, followed by ‘c’.

In Figure 2, I provide a color-coded visual representation of the parser sequence from Figure 1. The colors in the visualization match those used in the OCaml code in Figure 1. The red parser in Figure 2 is a sequence consisting of the purple parser followed by the grey parser. The purple parser itself is a sequence of the green parser followed by the rightmost brown parser. Finally, the green parser is a sequence of two brown parsers. The process begins with the red parser, which takes the input string “abcxyz”. This string is then passed through the purple parser to the green parser, and finally to the leftmost brown parser, represented by `satisfy (fun c -> c = ‘a’)`. The leftmost » that creates the green parser takes the unconsumed string “bcxyz” from the first brown parser and passes it to the second brown parser. The green parser’s output is the same as the second brown parser’s output. The next » that creates the purple parser extracts “cxyz” from the green parser and feeds it to the third brown parser. The purple parser’s output matches the third brown parser’s output. Finally, the third » that creates the red parser takes “xyz” from the purple parser and passes it to the grey parser. The `return true` parser adds `true` to the tuple and places the unconsumed string in the second position. The `true` indicates that the parsing has succeeded. Students are encouraged to consider what abstract syntax tree can be returned at this point instead of `true`. The grey parser’s output is also the red parser’s output. From the red parser’s perspective, it processes the input string “abcxyz” and returns the tuple `Some (true, ‘‘xyz’’)`.

To accommodate different parsing requirements, such as allowing zero or more spaces between characters, I provide an extensive library of parser combinators, including the `many0` combinator. This combinator takes a parser as input and runs it zero or more times, allowing it to parse sequences where a particular pattern may occur multiple times or not at all. This example effec-

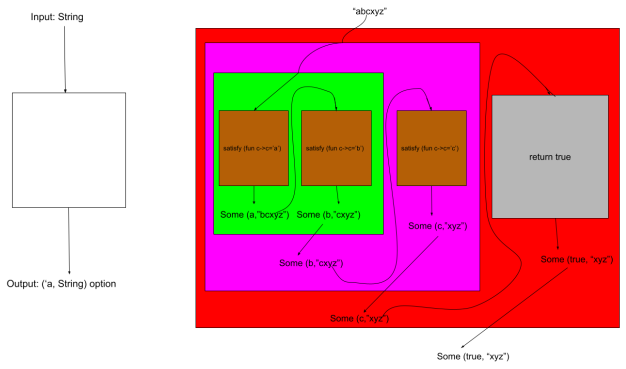


Figure 2: On the left is a box representation of a parser. On the right each color box represents a parser. For instance the purple parser is a sequence of the green parser followed by the right most brown parser.

tively demonstrates to students the ease of adapting parser combinators to new requirements. The new code that accommodates this is shown in Listing 2.

```
let space_parser =
satisfy (fun c -> c = ' '))

let parse_with_zero_or_more_spaces =
  satisfy (fun c -> c = 'a') >>
  many0 space_parser >>
  satisfy (fun c -> c = 'b') >>
  many0 space_parser >>
  satisfy (fun c -> c = 'c') >>
  many0 space_parser >>
  return true
```

Listing 2: OCaml code to parse strings with zero or more spaces between the letter 'a' and 'b'; between 'b' and 'c' and after 'c'.

In programming language courses, especially those focused on interpreter creation and extensive parsing, the inclusion of parser combinators is crucial. These combinators not only provide practical exposure to functional programming principles like higher-order functions and immutability but also enhance the readability and maintainability of code. This aligns well with the demands of modern, agile software development, offering flexibility and ease of use for rapid prototyping and adapting to evolving project requirements.

4 Teaching Course Evaluation Data

In Table 1, I present the evaluations for my Summer 2020 programming languages course at BU, which marked my first time teaching an OCaml course without parser combinators, over six weeks via Zoom. The following year, Summer 2021, I introduced parser combinators into the curriculum, dedicating 1.5 weeks to them and incorporating them into half of the assignments, as also detailed in Table 1. The latest evaluations from my Summer 2023 course at California State University, Chico, shown in Table 2, continue to reflect the use of parser combinators over the same six-week Zoom format, allowing direct comparison to the 2020 course without them. I have excluded evaluations from Fall and Spring semesters due to the different 16-week format and mixed in-person/online teaching during the pandemic.

Questions	Summer 2020			Summer 2021		
	N	SD	Mean	N	SD	Mean
The extent to which you found the class intellectually challenging:	16	.79	4.5	15	.96	4.13
The extent that assignments furthered your understanding of course content:	16	1.11	4.38	15	.5	4.47
The instructor's ability to present the material is:	16	.58	4.69	15	1.02	4.6
The instructor's overall rating is:	16	.77	4.69	15	.34	4.87

Table 1: Course Evaluation data from Summer 2020 and Summer 2021 teaching at BU on a scale of 1 (poor) to 5 (superior).

Some feedback from students in the Summer of 2020 at BU:

1. *I think we could have definitely had a little bit more time for the last few assignments as they are harder.*

2. *Problem sets were interesting and challenging.*

Some feedback from students in the Summer of 2021 at BU:

1. *Professor Attarwala was incredible at teaching this class! 320 with him was the most engaging remote class I've been in during the pandemic. His color coding, visualizations, and reinforcements really drilled in the material.*

2. *Professor has a great way of explaining concepts. His enthusiasm is definitely infectious and his use of visual aids especially the virtual blackboard with color-coded notation keeps me excited.*

Questions	**N**	**SD**	**Mean**
The course increased my knowledge of the subject matter:	20	.94	4.55
The assignments helped me understand the material:	20	.94	4.55
The instructor presented in an understandable manner:	20	.93	4.65
How do you rate the overall quality of teaching:	19	.54	4.79

Table 2: Course Evaluation data from Summer 2023 at CSU Chico on a scale of 1 (poor) to 5 (superior).

Some feedback from students in the Summer of 2023 at CSU Chico:

1. *His teaching style allows me to really understand concepts and I love how he visualizes concepts.*

2. *I very much enjoyed the prof. drawing on the board gave very good visuals pointers for the current material that was talked about.*

An analysis of the teaching evaluations in Table 1 and Table 2 reveals a subtle, but informative, trend. The introduction of parser combinators slightly affected the numerical ratings (especially of "The extent to which you found the class intellectually challanging" i.e., it decreased slightly at BU, however a comparable question at CSU Chico suggest that it increased again), but student testimonials emphasize the success of my visual and color-coded teaching methods in enhancing comprehension. When assessing the statement, "The instructor's ability to present the material is:", the numerical ratings decreased during the two semesters in which parser combinators were introduced. Conversely, for "The instructor's overall rating", the numerical ratings increased in the semesters that included teaching with parser combinators. This contrast between quantitative and qualitative feedback highlights the complexity of evaluating teaching effectiveness. While parser combinators increased course difficulty, effective teaching practices ensured positive learning experiences. Future investigations will employ my statistical frameworks [1, 2] to rigorously assess the benefits of parser combinators in programming education.

5 Impact of Parser Combinators on Teaching Effectiveness

In our examination of teaching outcomes, we anticipated that incorporating parser combinators—a notably complex topic—into the programming course curriculum would challenge students intellectually and improve their comprehension of the course material. In our research, the null hypothesis is stated as there is no difference in teaching effectiveness with the integration of parser combinators, and the alternative hypothesis, which anticipated a discernible impact, whether positive or negative. To investigate these hypotheses, we conducted a two-tailed Welch's t-test with an alpha level set at 5%, which indicates the threshold for rejecting the null hypothesis, across four dimensions of teaching effectiveness from the teaching course evaluations, i.e., (1) The extent to which you found the class intellectually challanging; (2) The extent that assignments furthered your understanding of course content; (3) The instructor's ability to present the material and (4) The instructor's overall rating.

In our analysis, we used Welch's two-tailed t-test and not the Student's t-test. The latter assumes homogeneity of variances across compared groups, the data presented in Section 4 suggests variability in this respect. Therefore, applying the Student's t-test might result in misleading outcomes. Welch's t-test is more appropriate for our data, as it does not require equal variances, as supported by the literature [3, 4]. It offers a more accurate analysis by adjusting degrees of freedom based on the sample sizes and variances of the groups compared. More formally here is how the *t*-statistic and the degree of freedom are calculated for the Welch's t-test:

$$t\_statistic = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \qquad Deg\ Freedom = \frac{\left(\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}\right)^2}{\frac{(\sigma_1^2/n_1)^2}{n_1-1} + \frac{(\sigma_2^2/n_2)^2}{n_2-1}}$$

μ_1, μ_2 are the means of the two groups, σ_1^2, σ_2^2 are their variances, and n_1, n_2 are the sample sizes. Table 3 displays the t-statistic, degrees of freedom, and p-value for each of the four evaluation questions. The comparison is between courses that included parser combinator instruction at BU in Summer 2021 and those that did not in Summer 2020. The table also presents t-statistics, degrees of freedom, and p-values for the same four questions, comparing the Summer 2023 parser combinator courses at CSU Chico with the non-parser combinator courses at BU in Summer 2020.

While statistical significance was not achieved in the results, it is remarkable that student evaluations consistently rated highly across all semesters, including those following the introduction of parser combinators. This also suggests a potential ceiling effect due to the high baseline of teaching performance (see Table 1 when the course was taught without using parser combinators at BU). Qualitatively, student feedback recognized and valued the increased depth and rigor that parser combinators brought to the course. This feedback aligns with my educational objectives of developing analytical skills and equipping students for the intricacies of real-world programming tasks.

The absence of statistically significant differences might be interpreted as an indicator of unchanged teaching effectiveness; however, it may also highlight the robustness of instructional quality in the face of introducing more complex subject matter. Future studies could explore different teaching methods or examine the long-term effects of incorporating advanced computational concepts into the curriculum. This research could provide valuable information on optimizing educational improvement strategies.

Evaluation Item	With Parser Combinators BU in Summer of 2021			With Parser Combinators CSU Chico in Summer of 2023		
	T-Stat	DF	P-Value	T-Stat	DF	P-Value
The extent to which you found the class intellectually challenging	1.167	27.19	0.2532	-0.173	33.89	0.8634
The assignments helped me understand the material	-0.294	21.13	0.7716	-0.488	29.49	0.6289
The instructor presented in an understandable manner	0.299	21.90	0.7675	0.158	32.30	0.8756
How do you rate the overall quality of teaching	-0.851	20.92	0.4045	-0.437	26.25	0.6658

Table 3: Results from Welch's test comparing teaching evaluations of BU Summer 2021 and CSU Chico Summer 2023 against BU Summer 2020.

6 Conclusion

In conclusion, the integration of parser combinators into the curriculum has been crucial in enhancing both code readability and adaptability, offering students a different application of functional programming principles in the context of parsing strings that is used very often in designing interpreter and compilers. The teaching method I used, which included color-coded diagrams, seems to have helped make parser combinators easier to understand. It is not clear if every student preferred this way, but overall, their feedback did not get worse, even with this challenging topic added to the course.

Although statistical tests did not produce significant results, this should not overshadow the pedagogical benefits observed. The consistency of high student evaluations, even with the incorporation of this advanced topic, suggests that the educational quality was maintained at its usual high standard.

In addition, students have reported that the challenge of engaging with such high-level material has been a rewarding experience. This qualitative feedback highlights the value of integrating such advanced topics into the curriculum, serving as a catalyst for developing critical thinking and problem solving skills.

Acknowledgement

We gratefully acknowledge the use of OpenAI's ChatGPT for proofreading, grammatical checks, and other text editing tasks.

References

[1] Abbas Attarwala. “Live coding in the classroom: Evaluating its impact on student performance through ANOVA and ANCOVA”. In: *2023 International Conference on Intelligent Education and Intelligent Research (IEIR)*. IEEE. 2023, pp. 1–6.

[2] Abbas Attarwala and Kun Tian. “A statistical framework for measuring the efficacy of peer review on students’ performance”. In: *2023 International Conference on Intelligent Education and Intelligent Research (IEIR)*. IEEE. 2023, pp. 1–7.

[3] Marie Delacre, Daniël Lakens, and Christophe Leys. “Why psychologists should by default use Welch’s t-test instead of Student’s t-test”. In: *International Review of Social Psychology* 30.1 (2017), pp. 92–101.

[4] Ben Derrick, Deirdre Toher, and Paul White. “Why Welch’s test is Type I error robust”. In: *The quantitative methods for Psychology* 12.1 (2016), pp. 30–38.

[5] Jeroen Fokker. “Functional parsers”. In: *Advanced Functional Programming: First International Spring School on Advanced Functional Programming Techniques Båstad, Sweden, May 24–30, 1995 Tutorial Text 1*. Springer. 1995, pp. 1–23.

[6] Mikhail Kuznetsov and Georgii Firsov. “Syntax Error Search Using Parser Combinators”. In: *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*. IEEE. 2021, pp. 490–493.

[7] Jamie Willis and Nicolas Wu. “Design patterns for parser combinators (functional pearl)”. In: *Proceedings of the 14th ACM SIGPLAN International Symposium on Haskell*. 2021, pp. 71–84.

[8] Jamie Willis and Nicolas Wu. “Design patterns for parser combinators in scala”. In: *Proceedings of the Scala Symposium*. 2022, pp. 9–21.

[9] Jamie Willis and Nicolas Wu. “Garnishing parsec with parsley”. In: *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala*. 2018, pp. 24–34.

[10] Jamie Willis, Nicolas Wu, and Matthew Pickering. “Staged selective parser combinators”. In: *Proceedings of the ACM on Programming Languages* 4.ICFP (2020), pp. 1–30.

Mind the Gap: Seven Years of Freshmen Retention Data and the Impact of COVID-19 on Undergraduate Computer Science Students[*]

Jennifer J. Coy[1], David L. Largent[1], and Sugandha Malviya[1]
[1]Department of Computer Science
Ball State University
Muncie, IN 47306
jennifer.coy@bsu.edu

Abstract

This paper presents an analysis of freshman retention data for Computer Science (CS) students over a period of seven years at a Midwestern medium-sized public university, with a particular focus on student demographics and the impact of the COVID-19 pandemic. The study examines demographic variables such as gender and ethnicity and compares data from pre-COVID and post-COVID times to explore changes in student retention and demographic representation.

The results show that the retention rates were significantly higher within the CS department compared to the university overall. Although it also revealed significant differences among different demographic groups, highlighting the challenges faced by women and minority students within the CS department. The onset of COVID further complicated these trends, disproportionately affecting these groups.

Through a comprehensive analysis of the data, the study aims to identify the key factors affecting retention within the CS field. In the future we plan to determine actionable steps to improve diversity and support strategies for underrepresented groups.

[*]Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

In the undergraduate computer science (CS) student population in the United States, a nationwide trend shows a lower representation of female and minority students than in the average university population [8]. At Ball State University (BSU), a Midwestern medium-sized public university, the CS student population follows the national trends but falls short in key demographic areas. Previous studies have found that women and underrepresented minorities are more likely to leave a computing major [3], [6]. Since one of the department's goals is to increase diversity participation in CS, it is important to clearly identify which populations could most benefit from additional support and opportunities.

Additionally, the impact of COVID on student success, while still being studied, seems to have had a disparate impact on women and minority students [4], [5], [7]. This study also examined trends in the data before and after the COVID pandemic and variations in retention and other key factors for different demographics and genders.

This analysis focuses on all first-time freshmen at Ball State University with a special emphasis on declared CS majors. The long-term goal is to identify actionable steps that can be taken to improve retention, especially for first-time students. This study's analysis allows the department to focus efforts on the areas which may yield the highest impact.

2 Data Set

The dataset for this study consisted of data for 21,624 first-time freshmen at Ball State University who first enrolled during the Fall semesters 2015 through 2021. Of these students, 8,236 had a declared major within the College of Sciences and Humanities, and 620 declared CS as their major. Each cohort (University, College, and Department) was compared based on gender and race/ethnicity demographics.

Data about students were extracted from university systems in the form of CSV files and de-identified before analysis. To accomplish de-identifications, data files including a student ID number, were imported into a database, where data about the same individual were joined into a single row of a table and the student ID number was removed, before being further analyzed using pivot tables.

2.1 Gender Distribution

As shown in Table 1 below, the University and College populations are composed primarily of female students, at approximately 61-62%, with 38-39%

male students.

Table 1: Gender distribution of cohorts at the University, College, and Department levels

Cohort	Male		Female		Other	
	Count	Percent	Count	Percent	Count	Percent
University	8,401	38.9%	13,215	61.1%	8	0.04%
College	2,723	38.3%	4,385	61.7%	2	0.03%
Department	530	85.5%	89	14.4%	1	0.16%

The CS department population was heavily skewed in the opposite direction, with nearly 86% male students, and about 14% female students. In comparison, the Computing Research Association 2020 Taulbee Survey [8] studied 151 US CS departments and reported that bachelors-level populations were composed of 79.1% male students, 20.9% female students, and 0.0% Nonbinary/Other students.

2.2 Race/Ethnicity Distribution

The distribution of students at the University, College, and Department level by race/ethnicity was categorized into four groups: White, Black/African American, Hispanic, and Other/Multiple. The University, College, and Department populations were composed primarily of students identifying as White, at approximately 75-77%, with 9-10% of the students identifying as Black/African American, about 6-7% of the students identifying as Hispanic, and 7-9% of the students with other or multiple races/ethnicities.

Table 2: Race/Ethnicity Distribution of cohorts at the University, College, and Department levels

Cohort	White		Black/AA		Hispanic		Other/Mult.	
	Count	Per.	Count	Per.	Count	Per.	Count	Per.
Univ.	16,728	77.4%	2,028	9.4%	1,346	6.2%	1,522	7.0%
College	5,323	74.9%	729	10.3%	501	7.0%	557	7.8%
Dept.	462	74.5%	59	9.5%	42	6.8%	57	9.2%

For comparison, the 2020 Taulbee Survey [8] reported that the enrollment in CS Bachelor's programs was distributed as 33.7% White, 4.3% Black or African American, 10.4% Hispanic, 22.7% Asian, and 28.9% Other. At Ball State University (Table 2), there is a higher Black/African American proportion and a lower percentage of Hispanic students, than the Taulbee survey. The Asian population was also lower and is included in the Other category in this analysis.

2.3 Retention Status

The analysis focused on incoming freshmen retention and was limited to examining undergraduate students who were first-time freshmen one fall and continued as students into the following fall. The number of students entering in the spring and summer semesters was found to be much smaller and so were excluded from the analysis. A student's retention status was categorized into one of three values:

- YES: they were retained in the major from their first fall to second fall semesters.
- NO: they remained at the university during this time period but switched majors.
- GONE: they left the university between their first fall and second fall semesters.

3 Results: Seven Years of Freshmen Retention Data

Table 3 shows that the total retention of students at the university (regardless of whether a student changed their major) was remarkably consistent, with approximately 27% of students leaving the university after their first year. The CS department had a higher YES retention rate than either the university or the college, so that fewer students changed their majors when starting in the department, as compared to students in other majors.

Table 3: Freshmen Retention for the University, College, and Department

Cohort	YES Retention		NO Retention		GONE Retention	
	Count	Percent	Count	Percent	Count	Percent
University	10,738	49.7%	4,963	23.0%	5,923	27.4%
College	3,665	51.6%	1,488	20.9%	1,957	27.5%
Department	346	55.8%	101	16.3%	173	27.9%

3.1 Retention Grouped by Gender

When the retention status is analyzed by gender, some additional differences emerge, as shown in Table 4. In this data, it is evident that both male and female students are retained in the CS major (the YES category) at a higher rate than at the college or university level. Within the CS major, male students had a higher rate of YES retention status than females.

Table 4: Freshmen Retention by Gender

Cohort	Gender	YES Retention		NO Retention		GONE Retention	
		Count	Per.	Count	Per.	Count	Per.
Univ.	Male	4,006	47.7%	1,950	23.2%	2,445	29.1%
	Female	6,729	50.9%	3,011	22.8%	3,475	26.3%
College	Male	1,432	52.6%	535	19.7%	756	27.8%
	Female	2,232	50.9%	953	21.7%	1,200	27.4%
Dept.	Male	298	56.2%	80	15.1%	152	28.7%
	Female	48	53.9%	21	23.6%	20	22.5%

3.2 Retention Grouped by Race/Ethnicity

When the retention status is analyzed by race/ethnicity, the YES retention rate for CS students was higher than the university rate for all races/ethnicities except for Black and African American students, as shown in Table 5. Within the department, this retention rate (39%) was substantially lower than the other ethnicities in the department, as well as the college (46%) and university retention rates (46%) for the Black/African American population. In contrast, the department's retention rate for Hispanic students (52%) was markedly higher than the college (49%) and university (44%) rates.

Table 5: Freshmen Retention by Race/Ethnicity

Cohort	Race/Eth.	YES Retention		NO Retention		GONE Retention	
		Count	Per.	Count	Per.	Count	Per.
Univ.	White	8,540	51.1%	3,911	23.4%	4,277	25.6%
	Black/AA	929	45.8%	407	20.1%	692	34.1%
	Hispanic	593	44.1%	294	21.8%	459	34.1%
	Other	676	44.4%	351	23.1%	495	32.5%
College	White	2,810	52.8%	1,138	21.4%	1,375	25.8%
	Black/AA	337	46.2%	139	19.1%	253	34.7%
	Hispanic	244	48.7%	103	20.6%	154	30.7%
	Other	274	49.2%	108	19.4%	175	31.4%
Dept.	White	269	58.2%	73	15.8%	120	26.0%
	Black/AA	23	39.0%	12	20.3%	24	40.7%
	Hispanic	22	52.4%	9	21.4%	11	26.2%
	Other	32	56.1%	7	12.3%	18	31.6%

3.3 Retention Grouped by a Student's Incoming High School GPA

Several other factors were analyzed in this study and are reported elsewhere [1], [2]. Of particular interest here is the High School Grade Point Average (HSGPA) and its relation to the student's retention status within the CS major.

For this population, the average HSGPA for female CS majors (3.6) was higher than for male CS majors (3.4). The average HSGPA also varied by ethnicity, with White and Hispanic students having the highest incoming HSGPA for the department (3.45 and 3.63, respectively), and the average HSGPA for the Black/African American and Other populations averaged 3.28 and 3.35, respectively.

When a student's incoming HSGPA is compared to their fall-to-fall retention status, the average HSGPA for students who are retained in the major (YES) was 3.54 which is higher than the average of those students in the NO (3.43) and GONE (3.24) retention categories.

4 Results: Pre-COVID and Post-COVID Analyses

These seven years of data were split into pre-COVID and post-COVID categories, to look for signs of the impact of the pandemic, where the pre-COVID range included the falls of 2015-2018 (4 years of data), and the post-COVID range included the falls of 2019-2021 (3 years of data). At the time of the analysis, our data set was complete only through fall 2021.

4.1 Pandemic Impact on Gender Demographics

When the gender demographic data is examined through pre-COVID and post-COVID categories, we see that the percentage of female students in the CS department increased from a pre-COVID value of 13% to 16% post-COVID. Similarly, the percentage of female students in the college increased from 62% to 65% post-COVID.

4.2 Pandemic Impact on Race/Ethnicity Demographics

For students identifying as White, the percentage for the department and the college pre-COVID were both around 77%, while post-COVID both percentages dropped to roughly 70%. For Hispanic students, the CS numbers increased from 6% to 8%, similar to the college's increase from 6% to 9% post-COVID. Black/African American students, who showed an increased representation in the CS department from 8% to 12% post-COVID, was a larger increase than was seen in the college (10% to 12% post-COVID).

4.3 Pandemic Impact on a Student's Incoming High School GPA

The HSGPA of males and females showed little change when comparing pre-COVID and post-COVID categories. However, when the data are examined by race/ethnicity, the average HSGPA differed pre-COVID and post-COVID for the CS department and college populations. As shown below in Figure 1, the HSGPA for Black/African American students decreased post-COVID at both the department and college level. The White population HSGPA, however, increased post-COVID. Students identifying as Hispanic had mixed results, with HSGPA decreasing post-COVID for students who declared CS as their major, while an increase was observed at the college level.

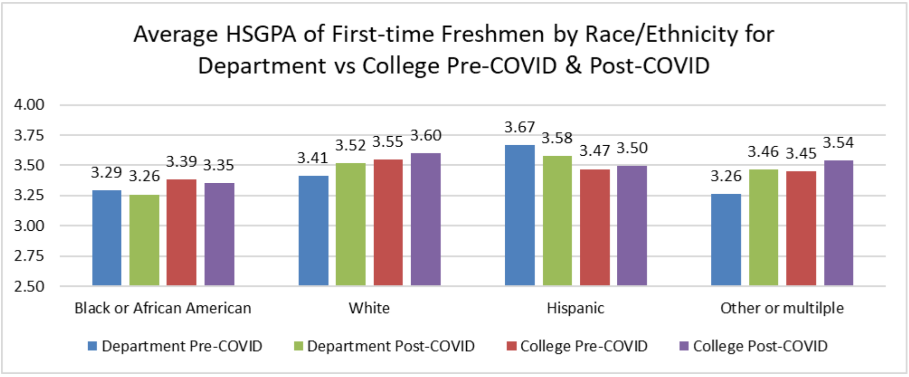


Figure 1: Pandemic Impact on a Student's Incoming High School GPA

4.4 Pandemic Impact on Retention Status by Gender

The data in Table 6 show that, before the pandemic, students who first declared as CS majors were more likely to stay in the YES retention category than the college cohort, regardless of gender. The largest difference is seen in female CS students, who were retained in the major at a rate of 59% for CS students, compared to 50% of students in the college. Additionally, pre-COVID, female students who first declared as CS students were also more likely to remain at the university (YES+NO categories) one year later (88%) versus female students (75%) in general within the college.

However, post-COVID, these trends changed. While male CS students still showed a higher YES retention rate than male students in the college (59% for CS vs 55% for the college), female students showed a lower YES retention rate (45% for CS vs 51% college). Female CS students choose to change their major (NO category) at double the rate of male CS students. Additionally, at

Table 6: Pandemic Impact on Retention by Gender

Cohort	Gender	YES Retention		NO Retention		GONE Retention	
		Pre	Post	Pre	Post	Pre	Post
College	Male	52%	55%	22%	17%	26%	29%
	Female	50%	51%	25%	19%	25%	30%
Dept.	Male	55%	59%	18%	10%	27%	31%
	Female	59%	45%	29%	20%	12%	35%

the college level, the percentage of students leaving the university increased by 3% for male students and 5% for female students.

The trend on a year-by-year basis is also intriguing, when the number of females/males in the GONE category are analyzed as a percent of the total number of female/male freshmen for that year. These trends are shown in Figure 2. It should be noted, that post-COVID, the rate at which students left the college (GONE) increased, though the cause of this relationship cannot be determined by this study.

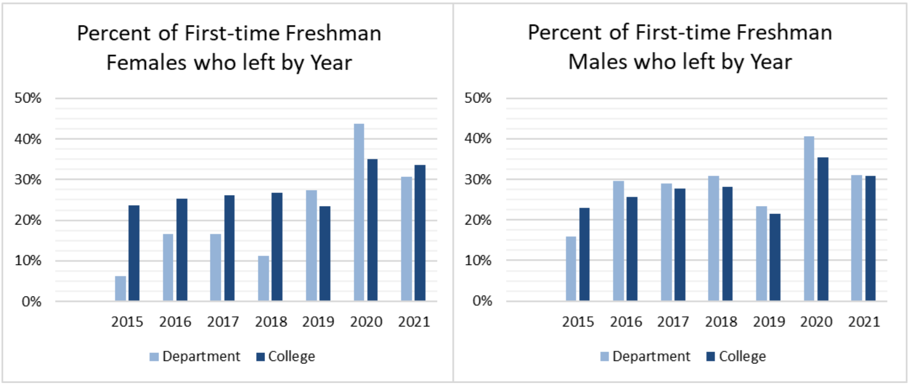


Figure 2: Pandemic Impact on Retention (GONE Category) by Gender, year-by-year

4.5 Pandemic Impact on Retention Status by Race/Ethnicity

Examining the data by race/ethnicity categories (Table 7) shows that CS majors are more likely to stay in their first major (the YES category), compared to the college, except for Black or African American students. For these students, the pre-COVID comparisons are 31% of CS students in the YES category vs 45% for the college. Post-COVID is similar, with 47% percent for CS and 50%

for the college. This is a positive trend, however, where the rate for the Black or African American students in the Yes category is approaching the college value. Interestingly, pre-COVID students switched their major more often, with the NO retention category having values around 20%. Post-COVID, this rate dropped into the teens for both CS and the college.

Table 7: Pandemic Impact on Retention by Race/Ethnicity

Cohort	Race/Eth.	YES Retention		NO Retention		GONE Retention	
		Pre	Post	Pre	Post	Pre	Post
College	White	52%	53%	24%	18%	24%	29%
	Black/AA	45%	50%	21%	16%	33%	34%
	Hispanic	48%	48%	21%	22%	31%	30%
	Other	44%	54%	22%	16%	34%	30%
Dept.	White	59%	56%	19%	11%	22%	33%
	Black/AA	31%	47%	24%	17%	45%	37%
	Hispanic	45%	65%	27%	10%	27%	25%
	Other	52%	63%	12%	13%	36%	25%

Examining the overall rate for those students retained at the university (YES + NO categories) shows that, pre-COVID, the CS retention rates were comparable to the rates for the college for most race/ethnicities, except for Black and African Americans, where the CS retention rate lags the college rate 55% to 67%. Post-COVID, the CS retention rate for Black and African American students moved closer to the college rate (63% vs 66%). Some larger differences were seen post-COVID in white students who remained at the university (67% CS vs 71% college) and both Hispanic and Other showed increased retention of CS over college (75% vs 70%).

5 Conclusion

This paper presents an analysis of freshman retention data for Computer Science (CS) students over a period of seven years at a Midwestern medium-sized public university, with a particular focus on student demographics and the impact of the COVID-19 pandemic.

In terms of gender, the university population is composed of 61% female students, while the CS department has a much lower female representation at 14%. When comparing the post-pandemic data to pre-pandemic data, we see an increase of 3% in the percentage of female students within the department. These numbers, while trending upward, are still below the national average of 21% of female CS majors [8]. It is desirable to increase female representation in computing, and so the department will examine student recruitment practices to identify methods to potentially influence this gender disparity as well as explore methods of retaining females on parity with males.

The race/ethnicity demographic distribution of the CS department is similar to the university's distribution. When the data were examined for the effects of the pandemic, Hispanic and Black/African American representation increased slightly within the department, improving diversity, though further improvement is desirable.

When considering the fall-to-fall retention status, students in the CS department were more likely to be retained in their major than the general university cohort. This trend holds for both male and female students, but substantial differences were seen for some race/ethnicity populations, where students identifying as Hispanic were retained in the major at a higher rate, and Black/African American students were retained at a concerningly lower rate, comparatively. The higher retention of students within the department overall is a positive factor, overall, but some data about the race/ethnicity retention are concerning.

Additionally, the impact of the pandemic on these demographics and incoming HSGPA was examined. In particular, an increase in the representation of women post-COVID was observed. Changes in the race/ethnicity demographics were also observed post-COVID, with the representation of White students decreasing, and minority representation increasing. On average, little change in the HSGPA by gender was observed; however, after the pandemic, the incoming HSGPA of Black/African American decreased while the HSGPA of White students increased.

Retention numbers showed a noticeable decline post-COVID, with an increased percentage of students leaving the university, and a larger impact on female students than male students. Notably, when the data are examined by race/ethnicity, overall retention for White students decreased post-COVID, but retention for minorities increased at both the departmental and college levels, so that a larger percentage of minority students remained at the university in the post-COVID data set.

This study aims to identify the key factors affecting retention rate and facilitate ways to improve diversity and support strategies for underrepresented groups within the CS field. In the future, additional exploration of factors which may exhibit a causal relationship will be undertaken.

Acknowledgement

The authors thank Dr. Paul Buis and Dr. Rebecca Pierce for their valuable discussion and analysis. This work was supported by funding from the Ball State University Office of the Vice Provost for Academic Affairs.

References

[1] Jennifer J. Coy and David L. Largent. “Mind the gap! Searching the data for retention challenges”. Conference presentation. Original Lilly Conference on College Teaching, Oxford, OH, United States. Nov. 2022.

[2] Jennifer J. Coy, David L. Largent, and Rebecca Pierce. “Mind the Gap II: Searching Data for Retention Trends and Challenges”. Conference presentation. Original Lilly Conference on College Teaching, Oxford, OH, United States. Nov. 2023.

[3] S. Katz et al. “Gender and race in predicting achievement in computer science”. In: *IEEE Technology and Society Magazine* 22.3 (2003), pp. 20–27. DOI: `10.1109/MTAS.2003.1237468`.

[4] Catherine Mooney and Brett A. Becker. “Investigating the Impact of the COVID-19 Pandemic on Computing Students’ Sense of Belonging”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 612–618. DOI: `10.1145/3408877.3432407`.

[5] Shamima Nasrin Runa, Brett A. Becker, and Catherine Mooney. “Variations in Sense of Belonging in Undergraduate Computing Students Through the COVID-19 Pandemic”. In: *Proceedings of the 2022 Conference on United Kingdom & Ireland Computing Education Research*. UKICER ’22. Dublin, Ireland: Association for Computing Machinery, 2022. DOI: `10.1145/3555009.3555029`.

[6] N.B. Tamer and J.G. Stout. *Recruitment and retention of undergraduate students in computing: Patterns by gender and race/ethnicity (2016)*. `https://cra.org/cerp/wp-content/uploads/sites/4/2017/05/CS_RecruitmentRetention.pdf`. (accessed: 07/01/2024).

[7] Mark Zarb et al. “Becoming a CS1 Student in the Time of COVID”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’21. Virtual Event, USA: Association for Computing Machinery, 2021, p. 1362. DOI: `10.1145/3408877.3439518`.

[8] S. Zweben and B. Bizot. *2020 Taulbee survey bachelor’s and doctoral degree production growth continues but new student enrollment shows declines*. `https://cra.org/wp-content/uploads/2021/05/2020-CRA-Taulbee-Survey.pdf`. (accessed: 07/01/2024).

A Learn-By-Doing Software Security Course[*]

Charles Moreno, Carson Peterson, BJ Klingenberg, and Bruce DeBruhl
Computer Science and Software Engineering
California Polytechnic State University
San Luis Obispo, CA, USA
bdebruhl@calpoly.edu

Abstract

Over the previous 20 years the need for software engineering and computer security education in undergraduate computing curriculum has become apparent. At Cal Poly, we have adopted stand-alone courses in both of these domains but have identified the intersection of software engineering and cybersecurity as a domain with curricular opportunity. In this paper, we outline a secure software engineering course and share our experience with running this course. Our course is focused on practical hands-on education with three large projects covering producing a secure software product, threat modelling, and malware design. Lastly, we cover the ethical considerations of this course and potential pitfalls of similar secure software engineering courses.

1 Introduction

Over the last 20 years, the need to introduce undergraduate computer science students to software engineering processes and principles has been recognized as well as the need for software engineering to exist as an independent academic discipline from computer science. Simultaneously, over the last decade the need for computer security as an essential component of undergraduate computer

[*]Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

science educations has been recognized and largely adopted. However, the intersection of these two topics presents some unique and exciting opportunities for novel educational offerings. In this report, we introduce a secure software engineering course that bridges cybersecurity and software engineering to create a rewarding opportunity for undergraduate students.

In our course, we use hands-on education to cover secure software engineering, secure software development, threat modelling, and malicious software. This course is complimentary to our existing cybersecurity and software engineering curriculum allowing a deep exploration of the topics. Throughout this project we emphasize hands-on curriculum, practical and applicable materials, and concepts over individual tools. We start the course by introducing two seminal secure software engineering frameworks: the Microsoft Security Development Lifecycle [3] and Secure-by-Design. These frameworks are applied by students in a multi-week development project building on their previous software engineering experience. As the application is being developed students are introduced to various detailed topics around secure software project management, development, deployment, documentation, and testing and apply these concepts to their ongoing project. Once a project is developed students are taught methods for threat modelling and apply it to a student's project.

To round out the course, we cover modern malware design and detection techniques. For a hands-on and practical experience to malware education students setup a virtualized environment for Windows and Kali Linux, a cybersecurity focused Linux distribution. The students then develop a Trojan [5], a malicious program that a user is tricked into running. Students apply common malware analysis and detection techniques and how they are used in malware research. Students then apply stealth techniques to understand how modern malware authors avoid antivirus software.

As with any computer course, and specifically cybersecurity course, ethical considerations must be presented to the students. At the beginning of the course, including the syllabus, we introduce students to the need to think carefully about any actions they take. In particular, we tell students to only carry out penetration tests or malware analysis on systems they own and have permission to test. This is reiterated throughout the course as concepts are introduced with continued emphasis on legal and ethical frameworks.

In the remainder of this report we discuss the course's curricular context, the course design, the details of the projects, and lessons learned.

2 Background and Related Courses

Cal Poly has a strong history of undergraduate education in computer science and software engineering with focus areas in *Software Engineering* and *Com-*

puter Security. In both of these focus areas we have a set of exciting curricular options for students to learn and practice their craft. The current course report offers curriculum that will compliment this existing course work.

While each of the existing software engineering and security courses provide students with in-depth knowledge there is a clear curricular gap. In particular, students currently do not have a course to develop their skills in the secure software engineering space which we address here.

2.1 Software Engineering Curriculum

Cal Poly created a Software Engineering major in 2003. The curriculum covers topics ranging from Software Requirements and Design through Software Construction and Deployment. All computer science students are required to take an introductory course in software engineering that covers requirements elicitation, UI/UX design, development, testing, and DevOps. Students create a software web application using Agile methods in teams. As an elective, students may choose to take additional courses including individual software design and development or user-centered interface design and development. In the individual software design course students learn about software design, construction, and design patterns. In the UI course students are introduced to the importance of user-centered principles in the design of good interfaces and effective human-computer interaction.

Software engineering students must take a multi-quarter introduction to software engineering course that provides deeper coverage of requirements design, UI/UX design, development, testing, and DevOps. Additionally SE students are required to take the individual software design and UI course. Lastly, SE students take a year long capstone sequence where they work with a client to deliver a software product. This includes learning about product vision, user stories, UI/UX mock ups, design patterns, development, code quality, testing, product versioning, Continuous Integration, Continuous Deployment, and product delivery.

2.2 Computer Security Curriculum

Cal Poly has developed a concentration in privacy and security for both computer science and computer engineering students with a goal of allowing students to choose a set of courses that meet their particular cybersecurity interests. All students in the computer science (CSC), computer engineering (CPE), and in the future software engineering (SE) programs must take an introduction to computer security course. This course is designed as a survey of computer security covering security fundamentals, symmetric cryptography, public-key cryptography, hash algorithms, authentication, access control, software security, social engineering, network security , web security, and cybersecurity ethics. Because of its survey nature, each topic only is allotted one week of coverage and one assignment. For software security students are introduced to buffer overflows, integer overflows, types of malware, and security by design. For security fundamentals students are introduced to Saltzer and Shroeder's principles [12] and the CIA triad [13].

Students in all computing programs are also able to take multiple technical electives in the area of computer security. This includes standalone courses in cryptography engineering, privacy engineering, binary exploitation, network security, wireless security, hardware security, and a future course in secure machine learning. Most relevant to this course are the cryptography engineering course, privacy engineering and binary exploitation courses.

In the cryptography engineering course students are taught about a broader set of cryptographic algorithms as well as how to appropriately use libraries to incorporate these algorithms into their projects. In the privacy engineering course, students are taught about the technology, ethics, and policies that govern users data and privacy. Students learn about how to design a system to empower users to make decisions about their data and the students delve deeply into the ethical considerations around privacy for users. Lastly, the course on binary engineering introduces students to reverse engineering and exploit development against compiled software. This course focuses on low level code analysis and memory corruption. These three courses, combined with the secure software engineering course would provide a student with a very solid background on implementing a software project with appropriate privacy, security, and cryptographic underpinnings.

3 Software Security: A Learn-by-Doing Approach

In order to fill the curricular gap of secure software engineering identified, we provide a hands-on, or learn-by-doing, course where students can learn the theory of secure software engineering and get practical experience. We outline our course in Table 1. We start with the Software Development Lifecycle (SDLC) and introduce topics of securing the developer's environment and the need and definition of the Software Bill of Materials (SBOM). Then we move on to how to validate the security of a software solution including static and dynamic testing. As many projects are employing a Software as a Service (SaaS) strategy and utilizing cloud services, we cover securing both the local and cloud run-time environments since any security exposures in these areas can lead to significant product and business impacts. Then we cover strategies for designing for recovery and testing and we have students review and test each other's projects to highlight blind spots and vulnerabilities that many

Week	Lecture Topics	Lab Topics
1	Microsoft SDLC, Git security, SBOM, and supply chain security	Full stack project initialization
2	Static software pen test, dynamic software pen test, crypto pen test	Adding an SBOM and Crypto
3	Secure environment management (passwords and keys), secure distribution and signing, web vulnerability scanners	Software pen testing
4	Securing cloud environments	Pen testing continued
5	Design for recovery, verification, and testing	Application hardening
6	Threat modelling	Threat modelling an application
7	Threat modelling, CVSS, CVEs, and ATT&CK	Threat modelling continued
8	Malware introduction	Implementing a Trojan
9	Malware static analysis and evasion	Analyzing a Trojan
10	Malware dynamic analysis and evasion	Evading Anti-viruses

Table 1: In this table, we outline a secure software engineering course that teaches students to design and implement secure software.

developers don't recognize during the development process. Finally, we turn our focus to learning how the creators of malware think, and explore how they would create and deploy malware. This helps the students think through closing other vulnerabilities that malware creators exploit. We anchor these topics in three large student projects that we discuss in detail below. We discuss the secure implementation project in Section 4, the threat modelling project in Section 5, and the malware project in Section 6.

We have four guiding principles for curricular design. First, we introduce ethical considerations for each new concept or tool so that students consider the potential use and misuse of them. Second, we design our course so students could apply their knowledge in hands-on labs. It has been shown that students applying their knowledge to a hands-on problem set can help improve their learning [19]. Furthermore, we design our course using a modern tech stacks. Anecdotally, we find students are more engaged in technology that they perceive to be relevant to their future roles. Lastly, we design the course to introduce general concepts and then allow students to choose from a number of competing tools. This creates students that are flexible to different future tech stacks and tools they may encounter in different roles.

4 Learn-By-Doing Secure Application Design

In our first project, we introduce students to secure software development by guiding them in implementing and testing a web application. In general, we introduce students to concepts in lecture with a real-world focus on the importance of the topic as well as different tools and implementation options. We then have students implement the concept into their project with flexibility for their implementation choice. Project-based learning encourages student engagement because they boost student interest and motivation through direct interaction with projects, knowledge retention from practical experience, and diversification because they are adaptable to different cybersecurity topics and effective for teaching technical aspects [8]. To encourage continuous learning and self-efficacy there were also tasks that were solely the student's responsibility in the project. For example, students had to setup JSON Web Tokens and configuring https by using standard documentation. In our curriculum, this course took the theoretical concepts taught in the Intro to Computer Security course and combined them with the practical skills learned in the Intro to Software Engineering course. A core objective of this project was to get students to think like a security engineer. Students learned that developers have to balance security and usability, and that security should be in the back of a developer's mind throughout the whole development and production process in cycles instead of at one or multiple set points.

In our intro to software engineering courses students work to implement a full-stack web application including backend storage. The software engineering project focuses on developing a frontend and backend web application and deploying it in a team. The goal was to learn and apply general software engineering best practices using industry standard tools like GitHub, JavaScript, React, Express and MongoDB while also experiencing the challenges of a team environment. For our first project in the secure software engineering course, students apply cybersecurity best practices to the full-stack development process they previously learned. Implementation topics include authentication, access control, and secure coding practices. Analysis topics include security focused documentation and testing including developing a software bill of materials and applying vulnerability scanning tools.

4.1 Exemplar Project

Students start the project similar to a similar project from the Intro to Software Engineering Course using the Create React App Template [2]. Their initial app was just supposed to have a Login page and a protected Landing page that could only be accessed when logged in, embodying important access control concept. Students then added the cloud database using MongoDB Atlas, implemented JSON Web Tokens in the backend to be used as cookies in the frontend, configured encrypted traffic with https and used a software named CylconeDx to create a software bills of materials. These exercises followed relevant lectures. For example, in lecture we discussed details about software bills of materials including what they were, why they were important and tools to make them.

Students enjoyed these tasks since they implemented web components they interact with on a daily basis. Students then partnered up and applied vulnerability scanning tools to look for insecurities in each other's projects, returning a report to the other person about their project. Students then had to fix at least five of the reported vulnerabilities. For example, one student achieved this by adding input sanitization, proper error handling and type checking. Lastly, students implemented OAuth through Google in their application.

4.2 Ethical considerations

The secure web application project was a way for students to learn about software security however there are some important ethical considerations. One dilemma is that the class teaches students how to use vulnerability scanning tools. This may be used for malicious purposes if one wanted to scan a project for vulnerabilities for the purpose of exploiting them. When introducing vulnerability scanning we make sure to warn the students about only using these tools when they have written permission to pen test an application.

In addition, with the timeframe for the course and project there are some limitations to what can be implemented. For example, students weren't obligated to encrypt data stored in the database. This would be problematic if the class was just as is, but usually students learn that data stored in databases should be encrypted in other classes. If they don't have this knowledge, though, the student would be missing a critical security practice. Insecure coding practices could lead to dire consequences like a data leak or privacy violation. It is important to help students scope the limitations of their project.

5 Threat Modelling an Application

Students were introduced to threat modeling through lecture on the subject including what to consider and what questions to ask when threat modeling. Lecture covered common techniques including STRIDE [1] and PASTA [18], developing attack trees [15], persona non grata (making personas of potential threat actors) [7], an introduction to hybrid threat modeling and quantitative threat modeling [7], and security cards [17].

The goal of the threat modelling project was to take what the students learned in lecture and apply it to their project, fostering active participation and hands-on learning experiences. The timing of the project followed the completion of the secure web application project providing a familiar target to practice threat modeling techniques with. This project supported a core objective of getting students to think like a security engineer. Research done with cybersecurity professionals found that they describe "the security mindset in terms of three interlocking habitual mental processes: unconscious monitoring for anomalies and potential threats, deliberate investigating of systems to identify security flaws, and evaluating the relative risks of those flaws once discovered“ [16]. Students were taught to mimic such habits through the implementation of developing and threat modeling their application. First, students practiced monitoring for anomalies by performing explicit analyses on the project. They were also taught to investigate systems for vulnerabilities through the use of vulnerability scanning tools including Nessus, Snyk and GitHub Dependabot in addition to modeling diagrams of the secure web application they created and analyzing these diagrams for potential security flaws. Lastly, students were taught to evaluate the degree of risk caused by these flaws and vulnerabilities through learning about bug bars in lecture in addition to implementing fixes to found vulnerabilities (i.e. they had to balance danger of bugs with ease of fixing). In the end, this project served as a great way to morph the students' mindsets to think like a cybersecurity professional.

Students threat modeled their projects to emulate a real-world exercise that often starts with a fully functional product. This means that the system's requirements and specifications for functionality were defined before considering security concerns. A research paper about the secure software development life cycle found that "by integrating security practices from the initial design phase, employing secure coding practices, conducting regular security testing, and following secure configuration and patching processes, organizations can significantly mitigate the risk of security vulnerabilities“ [9].

We ideally aimed to shadow the secure software development cycle throughout our project, leaving students with practical experience that will hold in industry. While it is important to think of security from day one throughout the implementation of an application, it is also important to be able to connect the classroom experience to the real-world. Performing threat modeling techniques on the application they themselves developed allowed students to understand the related concepts in a practical and immersive manner that encourages retention.

5.1 Exemplar Project

Students were tasked with keeping meticulous documentation and analyzing a partner student's project thoroughly using different thread modeling techniques. First students were asked to conduct an initial interview with the partner student in order to get some background on the application the student made. Then they performed a manual analysis, systematically analyzing the architecture of the application including all existing files, the technology stack used, analysis tools that are being run, documentation available, database implementation and the pages available in the frontend and backend. This was mainly an eyeball test, explicitly scrutinizing the code for any threat factors and getting as clear a picture as possible for the application architecture. Students then performed a dynamic analysis on the secure web application. For this they determined which routes a non-authenticated user had access to, consider where a user could inject data, how stored data is shown to users, how the database is secured, etc. In addition, the thinking changed in that students had to switch from looking so much for insecure coding practices to thinking about if there were threat factors as the application ran.

After the initial analyses, students were tasked with creating three different diagrams based on the application they were threat modeling. The first was a diagram of the overall architecture of the application. It was supposed to show how users interact with the application, how the backend and frontend connects, how any external applications connect, and how any data stores connect. The second was a dependency diagram where students could just use the dependencies listed in the package.json file. It was important to note the dependencies and their current revision levels because a security engineer wants the whole software pipeline to be secure, including making sure that dependencies are up to date with vulnerability patches. The last was a data flow diagram that denoted how the data was stored in addition to how it passed through the application. A student reflected that this one took the longest but it was the most thorough and probably most important because the vulnerabilities the student ended up fixing were caused by improper data flow (i.e. no input sanitization and improper error handling) [14]. Learning how to make such diagrams gave students practical experience that will improve the future quality of their security analysis tasks. After finishing these diagrams, students applied each of the STRIDE frameworks to the diagrams multiple times.

After that all that was left was to create threat actor personas and write an executive summary. Students started by identifying direct and indirect stakeholders in the system and then identifying ways that the system could be used or abused to negatively impact stakeholders. Students came up with different threat actor scenarios that entailed what resources the threat actor would have, what method they would execute when attacking, any possible motivations and the human assets like private data and societal wellbeing that are impacted by the attacker. Students concluded the project with the mindset that they are security consultants for a company: writing an executive summary to pitch why a hypothetical company should trust their application's security with us based on how we secured this application.

5.2 Ethical considerations

Ethical considerations are paramount in cybersecurity education, especially within the realm of threat modeling and vulnerability analysis. One significant concern arises from the potential misuse of newfound knowledge, as students may be tempted to exploit vulnerabilities they uncover during their analyses. Furthermore, there's the ethical dilemma of creating threat models for applications against the owner's wishes, which can arise when owners are reluctant to fix existing security flaws due to costs or other motivations. Another ethical consideration is the possibility of individuals conducting threat modeling services privately noting vulnerabilities for future exploitation, raising concerns about privacy and trust. This highlights that trust between a cyber security practitioner and those who they are protecting is an important factor to take care of [6]. Lastly, it is vital to make sure that all stakeholders are correctly identified as to respect the interests of everyone involved in an application.

6 Learn-By-Doing Malicious Software

For the last project, students implemented and analyzed malware. In lecture, we discussed different types of malware and how different attacks work, for example, a command-and-control bot where a server controls a different system via some sort of implanted software. We then instructed students to set up a safe environment to practice the concepts they learned including a target Windows 11 virtual machine and a Kali Linux virtual machine to give them access to real-world tools a hacker might use.

From here the hands-on approach really takes hold, now that students understood the concepts of different attacks we instructed them on a progression of malware development. They work on expanding their malware slowly increasing its trustworthiness in the host Windows 11 computer by rewriting their C code as C#. Then they hide the malicious code with encryption and encoding. After that, they used process injection to hijack an existing trusted process. Next, students work on Virtual machine bypasses by emulating human behavior to prevent effective dynamic analysis. Students are introduced to the use of multiple static and dynamic analysis tools that are common in industry. Finally, student gets free reign to determine an upgrade to their malware and implement it. This is all in an attempt to show how malware authors think and implement safeguards to keep their malware from getting discovered and allowing students to think how a hacker thinks. Along with that, giving them an opportunity to create their own malware gets them to take ownership of the project while doing research in different avenues of software security in an attempt to find out how they want their malware to function and ultimately they will demonstrate one of the avenues they researched.

6.1 Exemplar Project

After the completion of the initial projects, one student decided to increase the malware's stealthiness. The student evolved their project to increase its stealthiness by using image steganography, where you hide malicious data in a benign medium. The student hid their code in four different ways; The first of the four ways was implemented by hiding shell code at the bottom of a deceivingly innocent dog photo and keeping that dog file in a folder that had a script that would read the bottom of the photo and allocate the shell code inside memory. This method was the least effective at hiding the malware. Next the student tried a similar where the executable file with all the code hidden within it was at the bottom of the picture. Then a simple script would open the picture, recompile the executable, and run it. This was more effective at hiding the malware but was still detected by some antiviruses.

The next two attempts to maximize stealth utilizing the same idea of hiding the shell code and executable file. However, the attempts were more stealthy by hiding both the shell code and executable file in their own respective dog photos inside the images' pixels. The student did this by employing a technique called least significant bit steganography where the least significant bit of each RGB color is swapped to the bit it needs to be for the hidden code. The downside to this approach is now the data you want to hide is limited by the size of the image as you can only hide 3 bits per pixel. For example, a standard HD photo is 1920 by 1080 pixels which means you have 2,073,600 pixels and you can store 3 bits per pixel leaving you at 6,220,800 bits or 0.741577 megabytes of hidden data. This is reflective of real-world challenges in malware design where code size must frequently be minimized. The student ran both of these versions through various anti-viruses and found none of them were able to detect it. As demonstrated by the example student's project, this project allowed for hands-on learning and an opportunity to be creative in their design.

6.2 Ethical considerations

Although teaching students how to create malicious software can be a great resource for them to think like a hacker, it does come with inherent risks. The most obvious risk is "that teaching offensive hacking skills increases the risk to society by drawing students toward criminal acts" [11]. After learning a student might take their newfound abilities and try them in the real world on computers they don't have permission to attack. These attacks might be for learning, entertainment, or economic gain, and this is a serious problem leading to possible criminal proceedings and or leaked data that could be taken advantage of. Teaching students to hack “so that they may use these skills in their future profession is in teaching them the ethical and legal implications of their skill, and the ramifications of misusing their skill classroom” [10] is essential to prevent students missteps towards illegal and harmful hacking. We therefore frequently emphasize the appropriate scope to use these technologies and the consequences of misusing them.

7 Conclusions and Lessons Learned

We continue to refine this course based on feedback and are working to improve student experience. In general, the course is well received by students and many are very excited about the practical hands-on experience gained. To improve the course in the future, we intend to make the following changes.

- *Rescoping the secure design project* - In order to spend more time focusing on the threats and hardening of applications, it may be advantageous for the students to bring the project they completed in their introduction to Software Engineering courses and use this as a basis for the secure application project. This will allow for a more robust application baseline for more interesting data flows and assessments. Alternatively, we could supply students with a full-stack application and have them add the login, security, and perform vulnerability and hardening activities on that sample application.
- *Develop or threat model first?* In two offerings of this course we have swapped whether we focus on threat modelling or secure development first. These two tasks are intimately inter-related and we believe that interleaving them are likely to offer the best educational experience. The exact interleaving is to be determined.
- *Splitting the course* With the student interest and depth of material in this area, we plan to split this course into two standalone courses in the future. The first will focus on the secure software engineering, secure development, cloud security, and threat modelling. The second will be a deeper dive into malware design and analysis in diverse environments.
- *Handling multi-architecture environments* One unique challenge to the malware design project was the use of shellcode payloads. Most students will have limited experience with x64 or ARM-v8 native assembly development. Because of this, we depend on existing shellcode from tools like Metasploit [4]. This required us to develop and support labs for both the Windows X86/AMD64 architecture and the Mac aarch64. Emulation of x64 on Mac is limited and presented some challenges.

References

[1] Steven F Burns. “Threat modeling: A process to ensure application security”. In: *GIAC Security Essentials Certification Practical Assignment. Version* 1 (2005).

[2] *Facebook Create React App Github Repository.* `https://github.com/facebook/create-react-app`.

[3] Michael Howard and Steve Lipner. *The security development lifecycle.* Vol. 8. Microsoft Press Redmond, 2006.

[4] David Kennedy et al. *Metasploit: the penetration tester’s guide.* No Starch Press, 2011.

[5] Jyoti Landage and MP Wankhade. “Malware and malware detection techniques: A survey”. In: *International Journal of Engineering Research* 2.12 (2013), pp. 61–68.

[6] Kevin Macnish and Jeroen Van der Ham. “Ethics in cybersecurity research and practice”. In: *Technology in society* 63 (2020), p. 101382.

[7] Nancy R Mead et al. “A hybrid threat modeling method”. In: *Carnegie MellonUniversity-Software Engineering Institute-Technical Report-CMU/SEI-2018-TN-002* (2018).

[8] Madhav Mukherjee et al. “Strategic Approaches to Cybersecurity Learning: A Study of Educational Models and Outcomes”. In: *Information* 15 (Feb. 2024), p. 117. DOI: `10.3390/info15020117`.

[9] Martin Otieno, David Odera, and Jairus Ekume Ounza. “Theory and practice in secure software development lifecycle: A comprehensive survey”. In: (2023).

[10] Brian A Pashel. “Teaching students to hack: Ethical implications in teaching students to hack at the university level”. In: *Proceedings of the 3rd annual conference on Information security curriculum development.* 2006, pp. 197–200.

[11] Ronald E Pike. “The “ethics” of teaching ethical hacking”. In: *Journal of International Technology and Information Management* 22.4 (2013), p. 4.

[12] Jerome H Saltzer and Michael D Schroeder. “The protection of information in computer systems”. In: *Proceedings of the IEEE* 63.9 (1975), pp. 1278–1308.

[13] Spyridon Samonas and David Coss. “The CIA strikes back: Redefining confidentiality, integrity and availability in security.” In: *Journal of Information System Security* 10.3 (2014).

[14] Simon Schneider et al. "How Dataflow Diagrams Impact Software Security Analysis: an Empirical Experiment". In: *arXiv preprint arXiv:2401.04446* (2024).

[15] Bruce Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[16] Koen Schoenmakers et al. "The security mindset: characteristics, development, and consequences". In: *Journal of Cybersecurity* 9.1 (2023), tyad010.

[17] Nataliya Shevchenko et al. *Threat modeling: a summary of available methods*. Tech. rep. Carnegie Mellon University Software Engineering Institute Pittsburgh United . . ., 2018.

[18] Tony UcedaVelez and Marco M Morana. *Risk Centric Threat Modeling: process for attack simulation and threat analysis*. John Wiley & Sons, 2015.

[19] Hsien-Tsai Wu et al. "The impact of supplementary hands-on practice on learning in introductory computer science course for freshmen". In: *Computers & Education* 70 (2014), pp. 1–8.

UNDERSTANDING THE EFFECTS OF ADDING A CS0 COURSE TO A COMPUTER SCIENCE CURRICULUM: A LONG-TERM STUDY*

David R. Surma[1] and Zoe E. Surma[2]
[1]Department of Computer and Information Sciences
Indiana University South Bend
South Bend, IN 46634
dsurma@iu.edu
[2]Software Engineer
sports-reference.com
Philadelphia, PA 19119
zs@sports-reference.com

Abstract

Retention in computer science programs has been, and continues to be, a critical issue for most higher education institutions. With the coming of the so-called *demographic cliff* in the next few years, retention improvement is seen as vital to the long-term health of the institutions and their programs. Most computer science departments have been working on retention issues for many years and have had mixed results. At our university, a mid-sized regional university, historically the loss rate of students after taking the first programming course (typically called CS1) has ranged from a third to nearly 50%. Initiatives to improve student persistence have been many including changes in content delivery, having more widely available and free tutoring, instituting peer mentoring programs and embedding student supplemental instructors into the course sections. At times the retention rate has improved but reversion to the

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

mean soon followed. Because of this, the computer science department created a new course, CS0, over a decade ago with the goal being to help prepare students to be successful in CS1. The course would provide a gentle introduction to basic programming concepts using a less complex programming language than what is used in CS1. The course would focus on problem solving and algorithm creation that would use real-world examples in hopes of sparking student interest and increasing their engagement. Sufficient time has now passed to allow for collection of years of data that can be used to make judgments on the success and shortcomings of this initiative. Fourteen years of results are analyzed to help determine the effectiveness of adding this new course in improving performance in CS1 and in increasing retention. Additionally, the allocation of departmental resources is discussed and used to consider the overall effectiveness of having added this course to the curriculum.

1 INTRODUCTION

Improving retention of computer science students in higher education has been of great importance to departments for many years. Historically, on a national basis with every incoming cohort, roughly between a third and a half of students fail to successfully complete the first programming course [11, 5, 6, 3]. This course, typically called CS1, has not been designed to be a weed-out course but, in effect, it has acted in this manner. There are many initiatives chronicled in computer science pedagogy literature that have tried to reduce this drop-out rate [7, 1, 10, 2]. Frequent ideas employed call for curriculum and programming language changes, teaching approach changes, increased tutoring, and using student mentors and supplemental instructors [4, 13, 12]. The thought in these studies is that while there are some students who are not suited for the challenging pursuit of a computer science degree, there is a significant number of students who could be successful if new or different teaching techniques and course strategies were employed. At our university, faculty have been focused on retention issues for at least the past 15 years. In 2011, several computer science faculty members published a paper that describes the department's work in creating a new course, dubbed CS0, designed to provide a gentle introduction to computing concepts [8]. The course presents real-world problems hoping to capture student interest that focuses on problem analysis and algorithm development during the first half of the course. In the second half, solutions would be developed and coded using a simple programming language [8]. The goal was to not overwhelm students with learning programming language syntax but to engage students by designing and writing algorithms and then using the computer to solve the problems. In meeting this goal, the hope was that students would have success not only in this course, but that they would be better prepared to succeed in the following CS1 course.

Now that over a decade has passed since the CS0 course has been developed and made part of the computer science curriculum, this research presents an analysis of the effectiveness of this course in meeting its founding objectives. The course designers' goal was to create a course that would help increase the students pass rate for the CS1 course. Therefore, of central importance is to look at the pass rate for students in CS1 who had taken the new CS0 course and to compare it to the rate for students who did not take this course. Then, any potential improvement would need to be weighed against the cost of implementing the new course. It is also important to examine the new course itself and to consider student performance in it. One question to explore is to see if the traditional drop-out rate of CS1 now applies to this CS0 course.

The organization of this paper is as follows: Section 2 describes the data collection techniques and the constraints on the data and in gathering it. Section 3 presents the data collected along with key observations. After the data has been presented, in Section 4, an analysis of the departmental resources required to implement the CS0 course is discussed. In today's reality of declining enrollment, available faculty resources have also declined. Any benefit of allocating increasingly scarce faculty resources must be weighed against the cost and the ability to sustain the effort. Finally, conclusions including ongoing and future research efforts are presented in Section 5.

2 DATA CONSTRAINTS

Data was collected beginning with the Fall 2009 semester and continued through the Spring 2023 term. The data collected for this study primarily involved information gleaned from the final grade rosters for the CS1 and CS0 courses. At this institution, there is both a Computer Science and an Informatics major. Both programs require students to successfully complete CS1. For this study, only students who took the CS numbered courses are considered. While it is true that some Informatics students switch to Computer Science, and vice-versa, the number is quite small and has virtually no impact on the overall data collected. Additionally, some science majors take CS1 as either a requirement (Physics) or as an elective (Math). Because the grade rosters do not explicitly denote this distinction, the collected data will include these other science students. Since there are typically no more than 2-3 of these students per term, including them has only a minimal effect. Other data required is whether students in CS1 have taken the CS0 course. This information was collected from looking at class rosters for these courses.

There are some issues that need to be considered when analyzing the data. Up until the Fall 2020 semester, incoming students to the computer science

Table 1: Pass rate for CS0

Year	Total Students	Number Passing	Number Not Passing	% Passing	% Not Passing
F2009-Sp2012	246	170	76	69.1%	30.9%
F2012-F2019	402	265	137	65.9%	34.1%
Sp2020-Sp2021	73	41	32	56.2%	43.8%
F2021-Sp2023	147	107	40	72.8%	27.2%
TOTAL	**868**	**583**	**285**	**67.2%**	**32.8%**

program, when meeting with their advisors for the first time, were asked if they had any programming background. If they said that they had, they were advised to start with CS1. Otherwise, they were advised to take CS0. Proof was not requested, and the level of understanding was not explored to any detail. Because the students advised to start with CS1 had claimed programming in their background, it would seem that they would have a better chance of success in the course. What is interesting to see is how these students performed compared with second-semester computer science students who had taken CS0 as their starting point. Another issue that impacts the data is that, beginning with the Fall 2021 semester, the department made CS0 required for all students unless they performed high enough on an optional test-out/placement exam. Hence, from this point on there would be a fairly large group of students with at least some programming experience taking CS0, and that makes it natural to assume that performance in CS0 would improve.

Another issue that has a small impact on the collected data is that it is not uncommon for students to retake CS1 due to not meeting the department's minimum grade requirement. The collected data looks at section-wide results and does not catch this case. However, when looking at twenty-eight semesters of data, this amount is negligible.

3 COURSE PASS RATES FOR CS0 AND CS1

The course pass rates for CS0 are given in Table 1 with the data presented in four parts. (For this study, *pass rate* is defined as a grade of C- or better as this is the lowest grade that the computer science program accepts.) The first row shows data for the first three years that the course was taught. During this time, it was taught exclusively by two of the course designers. These faculty members had a strong interest in the course and their level of enthusiasm in teaching it was very high. They devoted much time to developing and refining the course projects as well as improving content with each offering. The data shows that the pass rate for this period was 69.1%. The department's historical pass rate for the CS1 course is less than 60% so this pass rate (for CS0) is nearly 10% higher. Although the CS0 course is admittedly less rigorous than CS1 [8], it was encouraging to see this amount of improvement.

The second row presents results for the next seven and a half years. In this time period, additional department faculty members taught the course. Most used the resources developed by the course designers while only modestly incorporating their own materials. The pass rate for this time period dropped to below 66%, which was understandable as the instructors were learning to teach this course using the active classroom approach of the designers [8]. However, the pass rate was still an improvement over the department's historic CS1 pass rate.

The third row in the table shows results for the three semesters that were greatly impacted by the COVID pandemic. Restrictions were imposed on in-class teaching and the use of the weekly in-person lab was discontinued. In keeping with what happened in CS programs all over the country, student performance declined significantly. Although many efforts were employed to help students with on-line learning, the pass rate dropped to just over 56%. The last row in the table shows data for when the COVID restrictions were lifted and in-person teaching resumed. Additionally, starting with the Fall 2021 semester, the CS0 course was required for all new CS students (with the exception being those who passed an optional placement/test-out exam). The data shows that the pass rate rose dramatically to over 72%. Because many students who had prior programming experience had not been taking CS0 prior to this time now started taking the course, a good deal of improvement was expected and did happen.

Next, considering the data for all years the pass rate was 67.2%. As stated, the traditional pass rate for a first university programming course falls between 50% and 66.7% which puts our results on the high end of this range. When looking at just our university, the average pass rate has gone up from our historic 60% mark to this 67.2% rate. It is too early to get a full picture of how the change to require CS0 for all new students impacts this pass rate, but based on the last row in Table 1, early indications are promising that it will have a large positive impact.

To summarize the data in Table 1, students were performing better with their first university computer science course, CS0, than what they did when CS1 was their starting point. The loss rate was still high but improvement did occur. Next, it is important to keep in mind that the primary goal of creating the CS0 course was to make it so that more students would be able to *pass the CS1 course*. Therefore, Table 2 is presented that shows the pass rates for the CS1 course separated by students who had taken CS0 and those who had not. The data is divided similarly to what was done in Table 1. Namely, the

Table 2: Pass rate for CS1 for students starting with and without CS0

Year	Total Stds	# Stds w/o CS0	Num Passed	% Passed w/o CS0	# Stds w/ CS0	Num Passed	% Passed w/ CS0
F2009-F2019	1082	825	543	65.8%	257	172	66.9%
Sp2020-Sp2021	80	28	18	64.3%	52	39	75.0%
F2021-Sp2023	91	14	10	71.4%	77	60	77.9%
TOTAL	**1253**	**867**	**571**	**65.9%**	**386**	**271**	**70.2%**

first row presents data from the inception of CS0 until the semester before the COVID outbreak. The second row has the data for the three semesters of COVID and the third row presents data after CS0 was made a required course in the computer science curriculum. The first row data in Table 2 shows that for students taking CS0 as a starting point the pass rate for CS1 increased by only 1.1% compared to students who did not take CS0. During the three semesters of COVID restrictions there was a significant change. For this time period, students who had taken CS0 had over a 10% better success rate than those without it. It is difficult to ascertain a reason for this, but it could have something to do with the fact that students who had taken CS0 were in at least their second semester of college and might have been better able to handle the COVID changes than first semester students. Additional study is necessary to find other possible reasons for this uptick.

The last row has the results after CS0 was made a required course. The pass rate of the students taking CS0 rose to nearly 78%. The gap between students who took CS0 and who did not was 6.5%. Analyzing this amount is difficult because the students who did not take CS0 are students who either placed high on the placement test or whose program (e.g. Physics, Mathematics) would not count CS0 and therefore needed to begin with CS1. Still, 6.5% improvement for the CS0 students is a significant amount.

Considering the overall values in the table, the percentage improvement was 4.3%. While such an improvement is a positive step for the department, it must be noted that this value is aided by the 6.5% gap from when CS0 was made a required class. More time is needed to see how this 6.5% amount will vary. The lingering question is whether an approximate 5% pass rate improvement justifies the department resources dedicated to having this additional course. That question, along with a detailed analysis of the data presented in these two tables, is addressed in the next section.

4 ANALYSIS AND EFFECTS ON DEPARTMENTAL RESOURCES

Looking again at Table 1 and focusing on the pre-COVID (pre-Spring 2020) data, the pass rate for the CS0 course was approximately 67%. As stated, this was an improvement over our university's historical beginning computer science pass rate. However, it must be remembered that CS0 was designed to be a significantly gentler course than the traditional CS1 course. The pace of the course was far slower, and the amount of programming constructs covered was less. While this pass rate was better than the pass rate of CS1, a question is whether the department is simply moving the drop-out course from CS1 to this new CS0. Our university is a state university and as such, has admissions requirements that allows some students to enter the computer science program with relatively weak backgrounds. Therefore, there will typically be a group of students who quickly learn that computer science is not for them and will leave the program after an intro course no matter the level (CS0 or CS1). Still, the departmental belief is that the retention increase is significant enough to continue offering the course. As for scheduling and offering the course, in order to implement CS0 the department needed to offer, on average, two sections of it every semester. Being a 4-credit course, the 16 credits per year amounts to close to a full-time faculty position. This is a significant teaching investment, and so it is important to see if this is paying dividends. Also, there is an opportunity cost as this faculty position could be devoted to other endeavors. In order to get a more complete picture of whether this investment is paying off, the full computer science program for students must be considered. The computer science major requires approximately 55 additional credits of formal coursework after CS1 is completed. Using this number of credits and multiplying by the university's approximate tuition/per-credit charge yields the revenue generated per student for those completing the program. Based on an average faculty salary, roughly five additional students would need to be retained in order to break even. The data shows that for an incoming cohort of 80 students, there would be these 5 additional students retained if they would take CS0. (Note: Historically, we frequently have had incoming cohorts that have exceeded 80 students, but this amount is a safe prediction for the coming years.) Of course, this analysis assumes that all students who move on to CS2 complete the major. This is clearly not the case as students leave the major for a myriad of reasons at various times. However, it does significantly increase the odds of students continuing in the major and as such, makes it a worthwhile investment of faculty resources.

The department decided that the CS0 course was serving its purpose and beginning in Fall 2022, this course became required. Students passing a placement exam would allow CS0 to be bypassed making CS1 the starting point. By requiring CS0, it ensured that 2 or 3 sections of it would have to be offered each semester. If this would further increase retention into CS1, there is a possibility that an additional section of CS1 would also be needed. With budget restrictions for the foreseeable future, the possibility *of adding* any additional faculty, including adjuncts, is small. Therefore, the department may need to do some shuffling of upper division electives to accommodate these additional lower-level sections. This presents a challenge, but it is a good one for the health of the department.

5 CONCLUSIONS AND FUTURE RESEARCH

Looking at the data it is evident that students who take the CS0 course pass the CS1 course at a markedly better rate than students who started with CS1. The reasons for this improvement are no doubt many and varied. First, the CS0 course introduces students to basic programming constructs such as conditionals, loops, and functions. Then, when they see these again in CS1 they are better prepared to understand and use them. Next, they have been through the processes of high-level problem analysis, understanding problem input-outputs, algorithm development and then turning the algorithms into actual code. They would also be familiar with program testing in addition to properly documenting and formatting their code. Students in CS1, while they may have had some prior programming background, simply did not have the experience of CS0 and may be seeing many of the concepts presented for the first time.

Another factor to consider is the level of student maturity. Since students who have taken CS0 are, in general, second semester students, they would be slightly more advanced. Considering retention into the next course, CS2, students who have passed two programming courses are more confident that they would be able to successfully complete it and that encourages them to carry on with the computer science program and attempt the course. Additionally, a concept under much consideration at higher education institutions is students' *"sense of belonging"* [9]. Being in a second course, many of the students who took CS0 would have met other students and developed relationships for studying and working together on projects and assignments. This could definitely lead to better performance in CS1 and a greater penchant to persist to CS2.

Considering the drop-out rate for the CS0 course, typical trends are evident. There is a pretty high drop-out rate that mirrors national averages. Hence, the idea that the drop-out rate has simply shifted from CS0 to CS1 has some merit. However, the drop rate from the CS major is not as high for the students taking CS0 and the ultimate good effects make taking CS0 worth

Table 3: Preliminary Data - Persistence to CS2 for students starting with and without CS0

Year	Total Stds	# Stds w/o CS0	# Taking CS2	% Persist	# Stds w/ CS0	# Taking CS2	% Persist
2009-2012	234	196	75	38.3%	38	25	65.8%
2013-2017	292	228	69	30.3%	64	44	68.9%
TOTAL	**526**	**424**	**144**	**34.0%**	**102**	**69**	**67.7%**

it for students. Therefore, the data appears to make a clear case that having the CS0 course leads to students doing better in CS1 and moving on to CS2. The question centering on departmental resources is still an important, and open, one. In order to make a definitive conclusion about whether the expense and commitment to having this new course is worth it, additional research and data are needed.

The research being done now is studying the data for retention into CS2. Table 3 presents preliminary data for the persistence, or retention, of students who started with CS1 and CS0 and who enrolled in CS2. (The data is preliminary since it is incomplete showing only 9 years. Full data will be gathered and analyzed soon.) Looking at this data, students who started in CS1 without CS0 persisted to take CS2 at just a rate of 34%. The students who did have CS0 persisted at nearly 68%, or about double. Again, considering historical averages, there is typically a significant loss of students after taking a CS1 course. The reasons for this include students doing poorly in CS1 and learning that the field is not for them. However, with the introduction of a CS0 course, students will now have had 2 semesters of programming before they would take the CS2 course. Could it be that there will not be as large of a loss after CS1 since the students are, in effect, more *invested* in their program? If the retention rate into CS2 and beyond is improved enough, it would certainly justify the faculty resource commitment for the CS0 course. Further, it could lead to larger enrollments for upper-division courses. Hence, obtaining complete data and analyzing it would appear to be a valuable endeavor. When done, the department will have gained a more complete answer as to whether the investment of having the CS0 course in the curriculum has been worthwhile.

References

[1] A. Alvarez and M. Larranaga. “Experiences Incorporating Lego Mindstorms Robots in the Basic Programming Syllabus: Lessons Learned”. In: *Journal of Intelligent & Robotic Systems* 81 (2016), pp. 117–129.

[2] D. Baldwin. “Discovery Learning in Computer Science”. In: *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*. ACM, Feb. 1996, pp. 222–226.

[3] L. Barker, C. L. Hovey, and L. D. Thompson. “Results of a Large-Scale, Multi-Institutional Study of Undergraduate Retention in Computing”. In: *Proceedings of the 44th Annual Frontiers in Education Conference*. IEEE, 2014, pp. 1693–1700.

[4] T. Barnes, Heather Richter, and et. al. “Game2Learn: Building CS1 Learning Games for Retention”. In: *Proceedings of the 12th annual SIGCSE conference on Innovation and Technology in Computer Science Education*. June 2007, pp. 121–125. DOI: `10.1145/1269900.1268821`.

[5] J. Bennedsen and M. E. Caspersen. “Failure Rates in Introductory Programming”. In: *ACM SIGCSE Bulletin* 39.2 (2007), pp. 32–36.

[6] J. Bennedsen and M. E. Caspersen. “Failure Rates in Introductory Programming: 12 Years Later”. In: *ACM Inroads* 10.2 (2019), pp. 30–36.

[7] M. Corney, D. Teague, and R. N. Thomas. “Engaging Students in Programming”. In: *Twelfth Australasian Conference on Computing Education*. 103. 2010, pp. 63–72.

[8] H. Hakimzadeh, R. Adaikkalavan, and J. Wolfer. “CS0: A Project-Based, Active Learning Course”. In: *Proceedings of the IAJC-ASEE International Joint Conference on Engineering and Related Technologies*. University of Hartford, Apr. 2011.

[9] Kelly-Allen et al. “Belonging: A Review of Conceptual Issues, an Integrative Framework, and Directions for Future Research”. In: *Australian Journal of Psychology* 73 (2021), pp. 87–102. DOI: `10.1080/00049530.2021.1883409`.

[10] E. Lindoo. “Back to the Basics In an Effort to Improve Student Retention in Intro to Programming Classes”. In: *The Journal of Computing Sciences in Colleges* 34.2 (2018), pp. 72–78.

[11] L. E. Margulieux, B. B. Morrison, and A. Decker. “Reducing Withdrawal and Failure Rates in Introductory Programming with Subgoal Labeled Worked Examples”. In: *International Journal of STEM Education* 7.19 (2020). DOI: `https://doi.org/10.1186/s40594-020-00222-7`.

[12] R. Takas, T. Karasz, and et. al. “Successful Steps in Higher Education to Stop Computer Science Students from Attrition”. In: *Interchange* 53 (2022), pp. 637–652.

[13] H. T. Wu, P.C. Hsu, and et. al. “The Impact of Supplementary Hands-on Practice on Learning in Introductory Computer Science Course for Freshman”. In: *Computers & Education* 70 (2014), pp. 1–8.

A Pitfall of SQL Aggregate Functions*

Jamil Saquer
Computer Science Department
Missouri State University
Springfield, MO 65897
jamilsaquer@missouristate.edu

Abstract

Aggregate functions are essential SQL constructs. The usage of aggregate functions is limited to certain SQL clauses such as select and having. They are allowed in the where clause under certain conditions. Many students are not aware of this. This can lead to writing SQL queries that look correct but they are not. This paper aims to clarify this and to show cases where it is allowed to use aggregate functions in the where clause. It also gives recommendation to educators to help them clarify this issue to their students.

1 Motivation

In an undergraduate databases exam, the author of this paper noticed that several students wrote an SQL query that uses an aggregate function in what seemed to be a logical way. However, when trying the query in a database management system (DBMS) such as PostgreSQL or Oracle, the query did not work. Based on the fact that this query was given by several students and the instructor was initially confused by the mistake, this clearly indicates that such usage of aggregate functions must be confusing to students. The author investigated the reason behind the error in the query to understand the mistake and to help students avoid such pitfall in the future.

*Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

2 Introduction

Databases is an essential course in the computer science undergraduate curriculum [1]. One of the fundamental subjects that students learn in this course is Structured Query Language (SQL). The basic structure of an SQL query is given in Figure 1, where A_i represents an attribute, r_i represents a relation, and P is a predicate [5]. An SQL query can contain subqueries and other constructs such as exists, set operations, and forming groups.

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

Figure 1: Basic structure of an SQL query.

In an exam, students were asked to answer the following question:
Given the following database, where primary key attributes are underlined:
supplier(SID, name, city)
product(productID, description, color)
catalog(SID, productID, price)
Use the set membership operator to find names of suppliers that supply less than 10 products. Notice that the purpose of the table catalog is to show products supplied by different suppliers.
A solution that was provided by several students is as follows:

```
select name
from supplier
where SID in (select SID
              from catalog
              where catalog.SID = supplier.SID
              and count(productID) < 10);
```

Figure 2: SQL solution given by several students as an answer for an exam question.

This solution is logical and looks to be correct. It uses the set membership operator, in. Since the inner query is a correlated subquery that has the condition `catalog.SID = supplier.SID`, then the inner query will be evaluated once for every tuple from the outer query. To explain this, let us consider the instances of the tables supplier and catalog shown in Table 1 and Table 2, respectively.

Table 1: Instance of the supplier table.

SID	name	city
100	Mark	Omaha
200	Mary	Springfield
500	George	Dallas
150	Russ	Dallas

Table 2: Instance of the catalog table.

SID	productID	price
100	P100	10.45
100	P200	15.79
100	P300	14.50
200	P400	14.75
200	P200	14.95
⋮	⋮	⋮
200	P213	29.95
⋮	⋮	⋮

The DBMS will consider the first tuple from the table supplier, and will execute the inner query for that. I.e., the query will be translated to the query shown in Figure 3.

```
select name
from supplier
where SID in (select SID
              from catalog
              where catalog.SID = 100
              and count(productID) < 10);
```

Figure 3: The query in Figure 2 after substituting supplier.SID by the SID of the first tuple in the table supplier.

The first three tuples in the catalog table satisfy the inner query, and the count is less than 10. So, the system would give the name Mark as part of the answer. On the other hand, when considering the second tuple in supplier, and assuming that Mary supplies more than 10 products, when the inner query is evaluated for Mary who supplies more than 10 products, the name Mary would not be produced as part of the output. So, logically the above query makes sense.

Nonetheless, when trying the query in Figure 2 in PostgreSQL or Oracle, it does not work. For example, PostgreSQL gives the error message shown in Figure 4.

```
ERROR:  aggregate functions are not allowed in WHERE
LINE 114:                    and count(productID) < 10);
```

Figure 4: PostgreSQL error message produced by the query given by students.

3 Pitfall of Aggregate Functions

The error message in Figure 4 indicates that the usage of the aggregate function – count() in this case – is not allowed in the where clause. However, this is not always true. For example, an aggregate function is allowed in a subquery in the where clause as shown in the query in Figure 5, which uses the max() aggregate function to find the productID of the most expensive product.

```
    select productID
    from catalog
    where price = (select max(price) from catalog);
```

Figure 5: Valid usage of an aggregate function in the where clause.

The queries in Figure 2 and Figure 5 show that the usage of aggregate functions can be confusing to some students.

4 Correct Solution

To understand why the answer provided in Figure 2 is not correct, let us look at a correct solution to the exam question, which is shown in Figure 6.

The inner query in this solution forms groups based on SID. It then uses the having clause to keep groups that have less than 10 tuples. I.e., it keeps the tuples for suppliers that supply less than 10 products. After that, the inner query gets the SIDs for these suppliers. Then, the outer query uses the set membership operator, in, to get the names of these suppliers.

```
select name
from supplier
where SID in (select SID
              from catalog
              group by SID
              having count(*) < 10);
```

Figure 6: Correct query that uses the set membership operator to find names of suppliers that supply less than 10 products.

5 Discussion

A general SQL query can use groups, where, having, subqueries, as well as other constructs. A more general structure of an SQL query than what is given in Figure 1 is shown in Figure 7, where A_i and B_j represent attributes, r_i represents a relation, and P and Q are predicates [5].

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P
group by $B_1, B_2, ..., B_l$
having Q

Figure 7: General structure of an SQL query.

Predicates in the where clause are applied before the formation of groups, while predicates in the having clause are applied after the formation of groups. The purpose of the having clause is to impose conditions on groups where groups that do not satisfy the having predicate are filtered out.

Aggregate functions are used to aggregate information on groups. Examples of aggregate information include finding the sum, average, and count for a group of numbers. For example, to find the number of products supplied by each supplier, one needs to form groups over SID in the relation catalog, as shown in the inner query in Figure 6. Then, the having clause is used to keep the groups that have less than 10 tuples, i.e., the count is less than 10.

If an aggregate function is used in a query that does not include a group by construct, then the system assumes that there is only one group consisting of the tuples that satisfy its where clause [6]. An example of this is shown in Figure 8.

In the absence of a where clause, the group consists of all the tuples. For example, the query in Figure 9 works fine because the whole relation is consid-

```
select count(*)
from supplier
where city = 'Dallas'
having count(*) > 2;
```

Figure 8: Valid SQL query that includes where and having clauses but not a group by clause. In this case, the input to the aggregate function is one group consisting of the subset of the tuples that satisfy the where clause.

ered as one group, and the system uses the having clause to impose a condition on that group.

```
select count(*)
from supplier
having count(*) > 2;
```

Figure 9: A query that uses having clause with an aggregate function but without group by clause.

If keyword having is replaced by keyword where in the query in Figure 9, which would give the query in Figure 10, the system would give an error message similar to what is shown in Figure 4. Although simpler, this query has the same mistake as the query in Figure 2, which was provided by students in the exam.

```
select count(*)
from supplier
where count(*) > 2;
```

Figure 10: Replacing having by where in the previous query.

If keyword having is replaced by keyword where and a subquery is used to retrieve the result of the aggregate function as given in Figure 11, the query works fine.

The main mistake that students committed in the solution in Figure 2 is writing where count(productID) < 10. To avoid this mistake, students need to know the SQL constructs where aggregate functions can be used. They also need to understand when it is possible to use aggregate functions in the where clause and the correct syntax to do that.

select count(*)
from supplier
where (select count(*) from supplier) > 2;

Figure 11: Using a subquery in where clause to get result of an aggregate function.

6 Where Aggregate Functions can be Used

SQL aggregate functions can be used in the select clause and in the having clause [2, 3, 4, 5, 6]. Examples of this are shown in Figures 5, 6, and 8. An aggregate function can be used in the where clause if the aggregate function is used in a subquery and if the result of the subquery is a single value that can be computed before the outer query executes [3]. An example of this is shown in Figure 5. In that example, the inner subquery executes first and finds the maximum price, which is stored in memory. Then, the outer query executes and compares the price for each tuple with that stored value. This case can be generalized to explain the situation in Figure 6, where the result of the inner subquery is a set of values that can be computed and stored in memory before the outer query executes. Notice that, the query in Figure 6 uses the aggregate function count in a subquery in the where clause of the outer query.

Another case [4] states that an aggregate function can be used in the where clause if that is part of a correlated subquery. An example of this is given in Figure 12. Here the inner subquery will be computed once for every tuple from the outer query, as was explained in Section 2. The query in Figure 12 finds names of suppliers that supply less than 10 products. Nonetheless, it does not use the set membership operator.

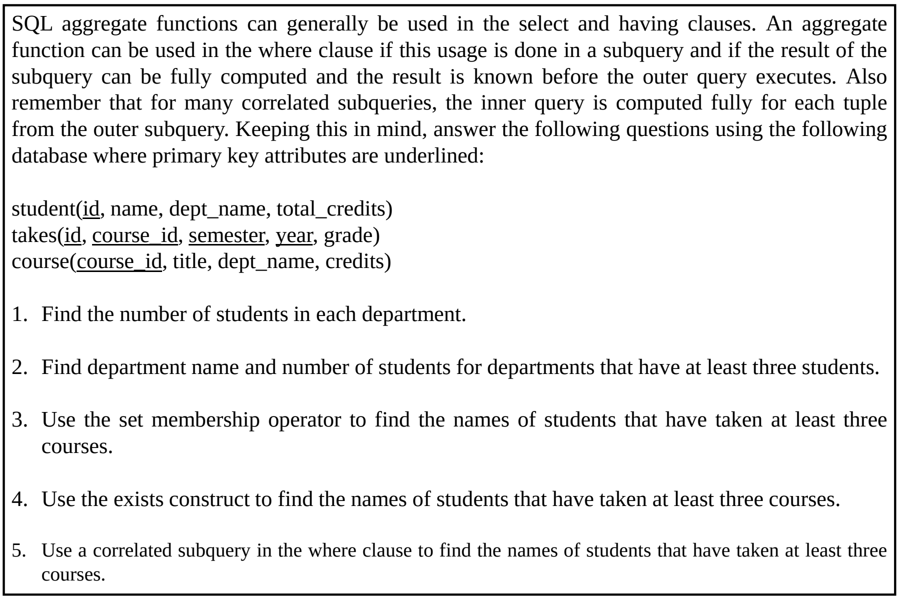
```
select name
from supplier
where (select count(*)
       from catalog
       where catalog.SID = supplier.SID) < 10;
```

Figure 12: Example of using an aggregate function in a correlated subquery in the where clause.

Notice that, the inner query in Figure 12 uses the aggregate function count in the select clause. Therefore, the aggregate function is used properly in this inner query. On the other hand, the inner query in Figure 2 uses the aggregate function count directly in its where clause. Therefore, it violates the rules discussed above (i.e., the rules mentioned in [3]) about using aggregate functions.

7 Helping Students Avoid the Pitfall and Recommendations to Educators

To help students understand the correct usage of aggregate functions and avoid the pitfall mentioned in this paper, the author developed a homework assignment that he used the following semester that he taught the databases course. Part of the assignment is given in Figure 13. The assignment starts by reminding the students about the constructs where aggregate functions can be used. It also reminds students by the rules about when aggregate functions can be used in the where clause. The assignment includes several questions that required students to use aggregate functions in the select, having, and where constructs. It also required them to use the set membership operator and correlated subquires.



SQL aggregate functions can generally be used in the select and having clauses. An aggregate function can be used in the where clause if this usage is done in a subquery and if the result of the subquery can be fully computed and the result is known before the outer query executes. Also remember that for many correlated subqueries, the inner query is computed fully for each tuple from the outer subquery. Keeping this in mind, answer the following questions using the following database where primary key attributes are underlined:

student(id, name, dept_name, total_credits)
takes(id, course_id, semester, year, grade)
course(course_id, title, dept_name, credits)

1. Find the number of students in each department.
2. Find department name and number of students for departments that have at least three students.
3. Use the set membership operator to find the names of students that have taken at least three courses.
4. Use the exists construct to find the names of students that have taken at least three courses.
5. Use a correlated subquery in the where clause to find the names of students that have taken at least three courses.

Figure 13: Part of a homework assignment designed to help students understand the correct usage of aggregate functions.

When the assignment was graded, the instructor made sure that students received feedback explaining the mistakes they made. Then, in the exam, the instructor gave a question very similar to the question in Section 2. None of the students committed the same mistake that was committed by the students in the previous semester. Based on this, the author of this paper recommends that when teaching aggregate functions, educators need to make sure that:

- Their students know the SQL constructs where aggregate functions can be used.
- They also need to explain to the students the rules regarding how aggregate functions can be used in the where clause. This is important because many database textbooks do not mention these rules.
- Provide students with an activity such as a homework assignment or a lab emphasising these rules, to help them practice using aggregate functions in different constructs. In particular, practice using aggregate functions in the where clause is important in helping the students avoid the pitfall mentioned in this paper.

8 Conclusion

Although the logic of an SQL statement may seem correct, this does not necessarily mean that it is correct. There are rules that an SQL query needs to adhere to. One such rule is regarding using aggregate functions. Students need to know these rules and to adhere to them. Aggregate functions can be used with or without a group by clause. In the absence of a group by clause, aggregate functions are applied to either all or a subset of the tuples in an input relation. Usually a group by clause is used to apply aggregate functions to groups of the tuples. Conditions imposed on groups are done using the having clause. Aggregate functions are usually used in the select and having clauses. When an aggregate function is used in the where clause, that must be done in a subquery. Moreover, the DBMS must be able to fully execute the inner subquery before it executes the outer query. It is important for students to understand these rules so that they avoid the pitfall of using aggregate functions inappropriately in the where clause.

References

[1] ACM. *Computer Science Curricula 2013.* `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`.

[2] Thomas Connolly and Carolyn Begg. *Database Systems: A Practical Approach to Design, Implementation, and Management.* Harlow, England: Pearson Education Limited, 2005.

[3] Carlos Coronel and Steven Morris. *Database Systems: Design, Implementation, & Management*. Boston, MA: Cengage Learning, 2016.

[4] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Boston, MA: Pearson Higher Education, 2016.

[5] Abraham Silberschatz, Henry F Korth, and Shashank Sudarshan. *Database system concepts*. New York, NY: McGraw-Hill Education, 2022.

[6] SAP Software Solutions. *Where You Can Use Aggregate Functions*. `https://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.sqlanywhere.12.0.1/dbusage/where-you-use-agg-fns-sorting.html`.

Developing an OpenCV-based App for Segmenting Cells in Tomato Fruit Tissue[*]

Jorge Daniel Quinteros[1], Lei Zhang[2],
Esther van der Knaap[2], Sofia Visa[1]
[1]Mathematical and Computational Sciences Department
College of Wooster
Wooster, OH 44691
{jquinteros25,svisa}@wooster.edu
[2]Department of Horticulture
University of Georgia
Athens, GA 30602, USA
{Lei.Zhang,esthervanderknaap}@uga.edu

Abstract

We present an OpenCV-based app for segmenting and measuring cells in tissue images, and we illustrate its application for generating hypotheses in genetics research. The app is used to process thousands of tissue images to identify genotypes with large cell size. These selected genotypes will be further studied to identify genes responsible for cell size.

1 Introduction

Discovering what genes regulate size and shape in tomato fruit is important because it helps breeders to create new plant crosses for various consumer needs. For instance, small tomato fruit varieties like the cherry tomatoes are appreciated for salads and snacking, medium and meatier sized ones like the

[*]Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Roma tomatoes are used for caning and sauces, while very large beefsteak tomatoes are used for sandwiches.

Recent advancements [4], [1], [8], [6] have identified several size and shape genes, however the developmental stage at which these genes are expressed is still unknown. Ongoing research [3] addresses this by generating microscopic images of tissue in the tomato fruit ovary at various developmental time points and by evaluating the cell size to identify how early these genes are expressed. Using OpenCV [2] and Python scripting, we have developed an app for automating this laborious cell tissue evaluation. Precisely, our tool detects cells, labels them, and computes their corresponding areas. While this tool is specifically developed for ovary and pericarp tissue in tomato fruit, it can be applied to other tissue types and other plants.

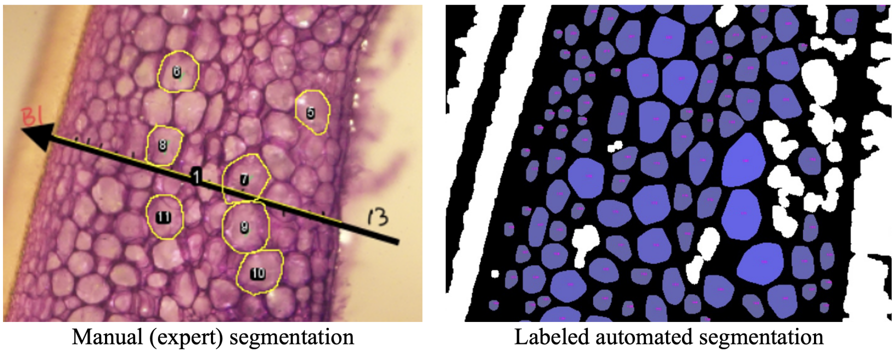
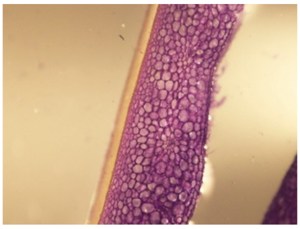


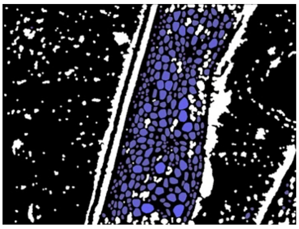
Figure 1: Manual (expert) segmentation of top seven largest pericarp cells (left) and automated segmentation of pericarp cells (right).

Currently, our geneticist collaborators hand-circle cells with the help of a general-purpose image analysis software like ImageJ [5]. Figure 1 illustrates seven cells manually annotated by a human expert. This is tedious, time consuming, and error-prone work as many times the human expert has a difficult time deciding the next largest cell among many cells of similar size. In this case, the expert simply chooses one, and moves along with this tedious work. Our tool, exemplified in Figures 1 and 2, speeds up this stage of researching genes regulating size and shape, by allowing to rapidly compare hundreds or thousands of tissues to identify the most likely time-point at which these size and shape genes are expressed.

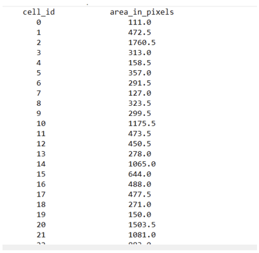
There are several image processing tools available like the macro in ImageJ, for instance, but this macro requires high contrast images and therefore does not work well for our images which are a bit on the faded spectrum. Since working on this project, another similar software was published, namely Cell-



Input: Original pericarp image



Output: cells detected



```
cell_id          area_in_pixels
   0                 111.0
   1                 472.5
   2                 1760.5
   3                 313.0
   4                 158.5
   5                 357.0
   6                 291.5
   7                 127.0
   8                 323.5
   9                 299.5
   10                1175.5
   11                473.5
   12                450.5
   13                278.0
   14                1065.0
   15                644.0
   16                488.0
   17                477.5
   18                271.0
   19                150.0
   20                1503.5
   21                1081.0
```

Output: text file with cell areas

Figure 2: The app outputs an image with labeled cells (middle picture) and a text file listing the cell areas (right picture).

pose [7]. This tool uses deep learning, and it is trained on 540 images. At a first glimpse, Cellpose works well on a variety of tissues including ours, but our software processes images faster than Cellpose.

2 Development of the OpenCV app for cell segmentation

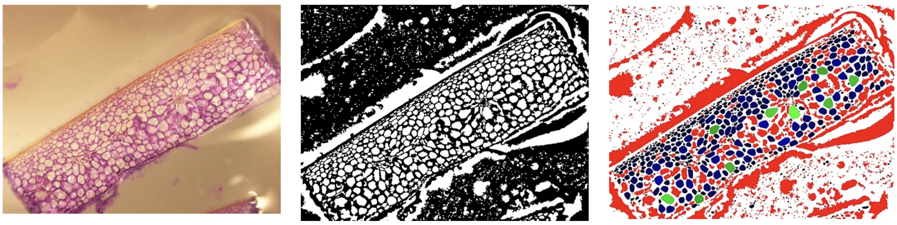
As illustrated in Figure 2, the app reads a pericarp tissue image of a mature green fruit and generates two outputs, namely an image with labeled segmented cells and a text file listing the cells and their corresponding areas. If the researcher is interested in processing only the top 10% largest cells, they will sort the cells in decreasing order of their size and select only the top 10% part of the text file. Alternatively, one can run the app requesting the segmentation of a specific number of largest cells. For instance, Figure 3 shows an output of 10 largest cells in a pericarp tissue.

Our cell detection and measurement app is written in Python and uses extensively OpenCV functionality. The app accepts .tif images but also works with other image types recognized by OpenCV, a free and open-source library of functions for real-time computer vision [2]. The main five steps of the app are included below and illustrated in Figure 3.

1. **Convert input image to RGB, and extract grayscale Color Channel.**
2. **Extract local maxima to capture cells using a Gaussian Threshold Noise-resistant Peak.** Here we apply cv.GaussianBlur(), a Gaussian filter to blur out sharp edges. Then we select those pixels from the original grayscale image with a value lower than the blurred image. In other words, we select many more core white pixels situated in the mid-

dle of the cell and fewer pixels situated at the edge of the cell. This step produces an OpenCV mask of the image.

3. **Fill holes in Cell Mask with morphological operations.** Here we use cv.floodFill(), a floodfill technique that removes holes inside of a cell. In brief, we start with one black background pixel and set to white all the reachable background pixels. In this new mask, the only remaining black pixels are the holes in the cells. Next, we do an OR operation between the mask from step 2 and the negative of the floodfilled mask to obtain a new mask that includes all the cells without any hole.
4. **Extract individual cells using the OpenCV cv.findContours() routine.**
5. **Filter cells with OpenCV contour properties** – in this step we remove cells that are intersecting the margin of the image, and we use the solidity measure from OpenCV to remove concave objects and long line objects, as they usually are margins of the tissue.
6. **Output the annotated image and write into a text file the cell area.**



Original image

The mask after filling holes in the cell (mask after step 3)

The red cells are filtered out; the green cells are the top 10 largest cells (mask after step 5)

Figure 3: Various steps in our OpenCV app for detecting tissue cells and computing their areas.

The OpenCV app described above is run in command line and accepts three parameters:

1. The folder with all the images to be segmented.
2. A scale if using one; this is given as a number and the program uses it defined as the number of pixels per unit.
3. The number of largest cells to highlight/detect in the image; if this parameter is missing, the app detects all cells.

Currently, we are working on creating a user-friendly tkinter graphical user interface for the app.

3 Comparing segmentations obtained with the OpenCV app and the human expert

With an automation tool in place, next, we focus on evaluating if its output is comparable to the human expert one. For this, we compare the areas of the eight largest cells computed via the two methods. As shown in Figures 4, 5, 6, we use three different sections from the same fruit, namely images 4360, 4361, and 4363. In these figures, for each of the eight cells shown on the x-axis the three bars correspond to areas in square millimeters computed (1) by the human expert; (2) by the app; and (3) by the app corrected by a 1.7 multiplicative factor, respectively.

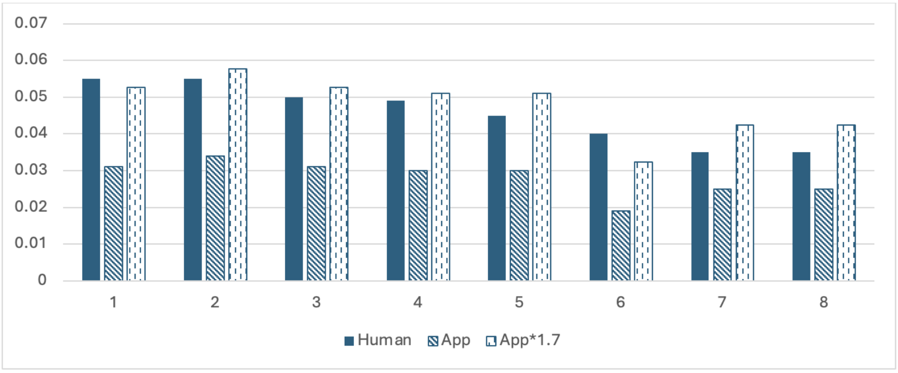


Figure 4: Cell area comparison for the eight largest cells in the first section of a single fruit (image 4360). For each cell, the three bars correspond to area computed by the human expert, by the app, and by the app*1.7, respectively.

Our app consistently underestimates the cell areas because it creates thicker intercellular walls visible in output images such as the ones shown in Figures 2 and 3. However, the areas computed by the app follow the same trend as the ones computed by the human expert. Further, by multiplying the app calculations with a corrective factor of 1.7, we observe that these new estimates approach the human expert evaluations – see Figures 4, 5, 6. From a genetics expert point of view, we have two remarks about this result:

1. **The human experts are not an absolute truth**, they too have biases in selecting by eye the largest cells, and at times, could not decide the largest of multiple cells of seemingly similar size. In addition, there are various biases between two or more human experts in segmenting these tissues, consistency of evaluation being one of them.

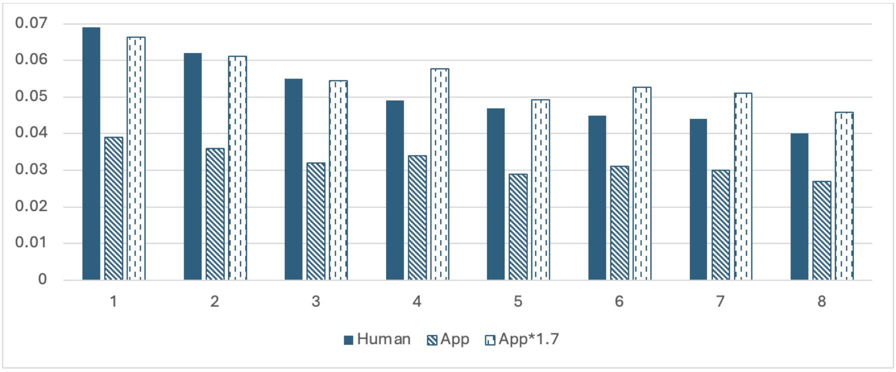


Figure 5: Cell area comparison for the eight largest cells in the second section of a single fruit (image 4361). For each cell, the three bars correspond to area computed by the human expert, by the app, and by the app*1.7, respectively.

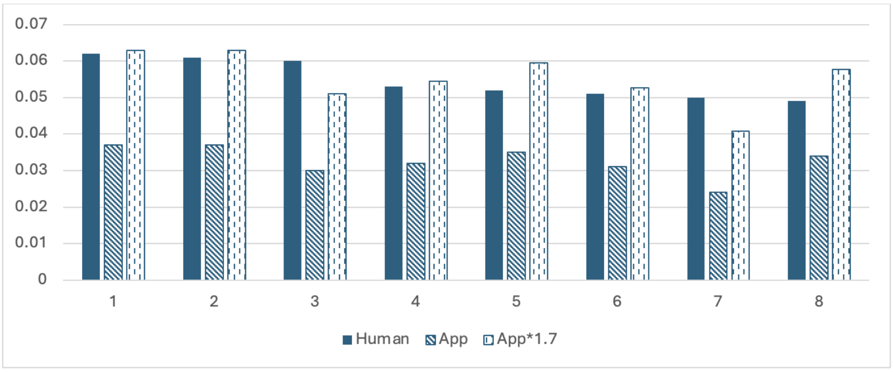


Figure 6: Cell area comparison for the eight largest cells in the third section of a single fruit (image 4363). For each cell, the three bars correspond to area computed by the human expert, by the app, and by the app*1.7, respectively.

2. **Both, the app and the human expert generate same hypothesis.** This automated model is used to generate hypotheses that can be further researched, so its main purpose is to analyze efficiently hundreds or thousands of tissues to select candidates for additional genetic studies. For instance, let's assume that the task is to compare tissues from varieties A and B to decide which one develops larger cells. If the human expert evaluates the average cell area of the top ten largest cells to 0.04 and 0.09, respectively, this will direct the researcher to study the variety B for identifying genes that regulate cell size. However, the automated model arrives to the same hypothesis as both average cell areas are smaller by a factor of 1.7, i.e. 0.024 for variety A and 0.053 for variety B.

4 Generating hypotheses with the OpenCV app

The next example illustrates one way in which our collaborators from Univ. of Georgia use this app to identify worthy hypotheses that are further investigated via more time-consuming lab experimentations. In this example, our plant geneticist collaborators utilized a tomato population 20S87 with plants mostly identical in their genome. Only a small region containing a few genes are different between those noted as genotypes A, B, and H. The genotype A plants carry larger fruits than genotypes B and H plants. The hypothesis is that the larger fruits from genotype A is caused by larger cell size. If such hypothesis is proven, we may further deduce that one of the genes in the differed genomic region regulate cell size.

Table 1: The 135-image data for population 20S87 are obtained from 3 genotypes x 5 plants x 3 fruit x 3 sections

Tissue name	Plant number "pl"	Fruit number "fruit"	Section number "sect"
A-20S87-pl-fruit-sect	17, 23, 29, 31, 33	1, 2, 3	1, 2, 3
B-20S87-pl-fruit-sect	20, 26, 27, 30, 66	1, 2, 3	1, 2, 3
H-20S87-pl-fruit-sect	07, 15, 16, 19, 35	1, 2, 3	1, 2, 3

To compare the cell size in tomato fruits, pericarp tissue (flesh tissue) was sectioned from each fruit and imaged under microscope. The geneticists provided us with 135 images (3 genotypes * 5 plants * 3 fruit * 3 sections). For each genotype, we collect fruits from five different plants denoted by "pl" in Table 1. From each plant we collect three different fruits, labeled as "fruit" in the table, and from each fruit we dissect three pericarp sections to view under microscope.

The research question we are interested in answering is: *which (if any) of these three genotypes carry a gene allele that produces large cells?* To answer this, we use the OpenCV app for identifying the 10 largest cells in every image and then compute their areas. Next, for each genotype, we compute with a Python script the per-plant minimum, average, and maximum areas (in pixels), as shown in Figure 7. On this figure we observe that genotype A-20S87 has the largest cells while genotype H-20S87 has the smallest cells; however, a statistical test will tell how relevant these differences are.

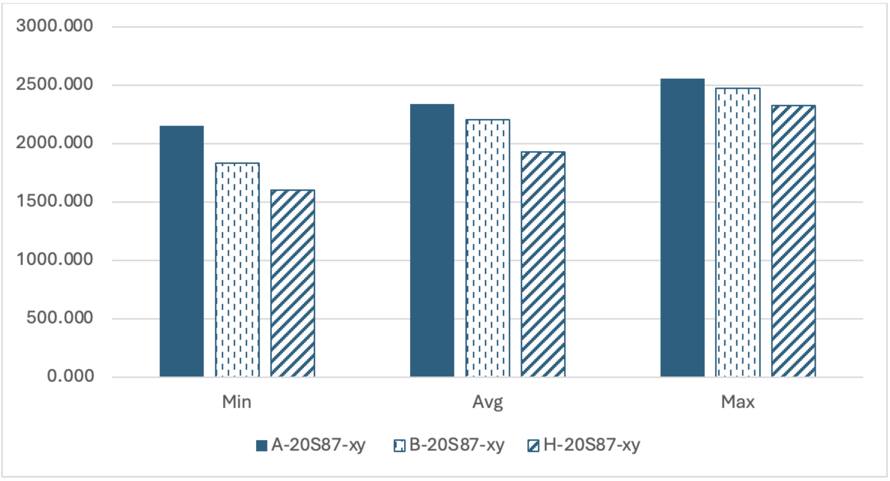


Figure 7: Per-genotype minimum, average, and maximum area; genotype A-20S87 has the largest top 10 cells.

Table 2: Pairwise ANOVA between A, B and H genotypes, analysis of the 10 largest cells; The A vs H experiment rejects the null hypothesis meaning there is a significant difference between the cell areas in these two genotypes, namely there are larger cells in genotype A than in H.

ANOVA statistics for the OpenCV app	A and B	**A and H**	B and H
F	0.993	**8.881**	2.670
p	0.348	**0.017**	0.140

Table 2 shows the pairwise results of an ANOVA analysis for areas of these three genotypes. Based on F and p values, the two genotypes to be further explored are A and H. Namely, ANOVA for these two entities rejects the null hypothesis, meaning there is a significant difference between the cell areas in genotypes A and H. This further implies that the dominant allele in A creates significantly larger cells than the allele in H, thus answering the question posed above. Therefore, the gene allele or mutation that will be researched further is the one in genotype A.

Table 3: Pairwise ANOVA between A, B and H genotypes, analysis of the 100 largest cells; Only the A vs H experiment rejects null hypothesis meaning there is a significant difference between the cell areas in these two genotypes, namely there are larger cells in genotype A than in H.

ANOVA statistics for the OpenCV app	A and B	**A and H**	B and H
F	4.753	**10.831**	2.493
p	0.060	**0.011**	0.152

Table 4: Pairwise ANOVA between A, B and H genotypes, analysis of all cells; none of the three comparisons reject the null hypothesis.

ANOVA statistics for the OpenCV app	A and B	A and H	B and H
F	4.452	4.001	2.915
p	0.067	0.080	0.092

Table 3 summarizes the ANOVA analysis when looking at the largest 100 cells. In this experiment, again, we observe that the largest cells are in genotype A and the smallest ones in H. In fact, the same hypothesis is generated as in the 10 largest cells experiment, but with even better statistics: a larger F-value and a lower p-value. However, when repeating the experiment using all cells (see Table 4), this hypothesis is no longer evident. Namely, while the F-value of 4.001 might suggest some differences between the cell sizes from the two genotypes, the p-value of 0.080 is larger than the confidence threshold of 0.05 to be able to reject the null hypothesis. At a second look, this result is not surprising if considering the average per-plant cell count plotted in Figure 8. On average, genotype A has the larger number of cells (an average of 582 per plant), and genotype H has the fewest number of cells (463 cells per plant).

This suggests the following two hypotheses which are evident in the tissue samples from Figure 9:

1. Genotype A consists of many large cells and of many small cells ready to grow.
2. Genotype H has fewer cells in general, and these cells are of average size.

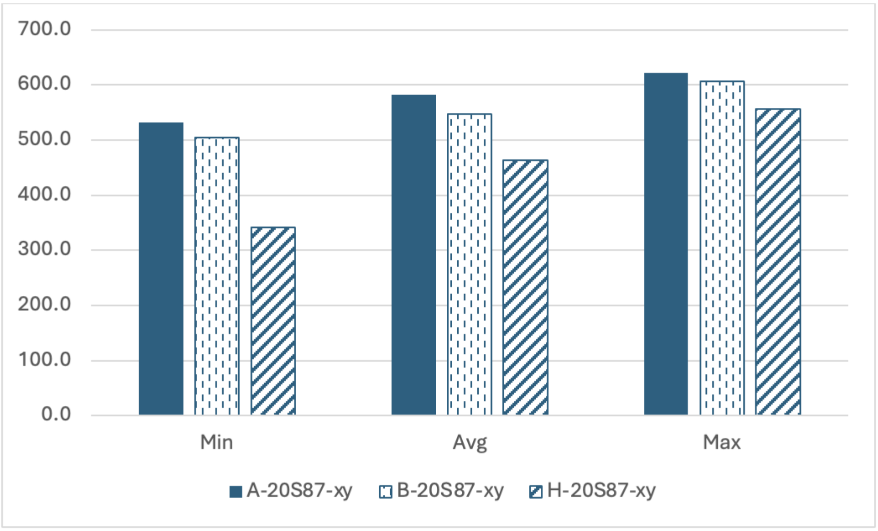


Figure 8: Per-plant (x-axis) minimum, average, and maximum number (y-axis) of cells; on average, genotype A-20S87 has the largest number of cells (582 per plant) and genotype H-20S87 has the least number of cells (463 cells per plant).

When we incorporate all cells in the ANOVA analysis, we work with overall similar total tissue areas in both genotypes A and H. Further, under the two hypotheses described above, the cell area average in the entire genotype A tissue can be similar to the one in genotype H. For instance, an average of 4 can be generated by cell areas of 7, 7, 2, 2, 2, an A-like genotype area distribution with large variance of cell sizes, some cells being large and others being small. However, an average of 4 can also be generated by cell areas of 4, 4, 4, 4, an H-like genotype area distribution consisting of less cells of medium size. Figure 9 shows actual tissue examples of the two genotypes. These samples are consistent with the above analysis and hypotheses. Also, as the cells get smaller, they are harder to segment from the background in the image. Therefore, including more of the cells past a certain point would decrease the effectiveness of the experiment – this is another deterrent for using all cells in the tissue for this type of tissue comparison.

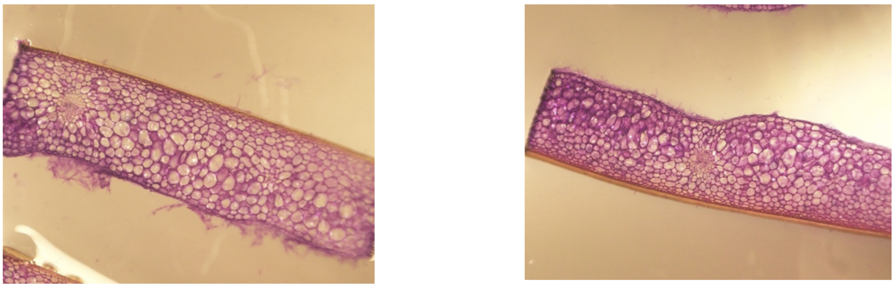


Figure 9: Tissue samples from genotypes A-20S87 (left) and H-20S87 (right).

The above analysis leads to an important observation. Namely, **to generate hypotheses about tissue cell-size, it is best to compare the top 10% or 20% largest cells in the tissues;** it is *not* recommended to compare all cells in the tissues.

Another important note is that, when this type of tissue comparison is done by the human researcher, the top 10 largest cells experiment is what the human would normally do, because segmenting the 100 largest cells would be too tedious. But with this app, the largest 100 cells analysis is completed within a few minutes, demonstrating the usefulness of this app.

5 Conclusions and future work

Here we present an efficient OpenCV app written in Python which is useful in segmenting cells in fruit tissues and in computing their area. In addition, we illustrate on a real example how this app generates hypotheses to be further researched by genetics experts. With recent advancement in deep learning for image analysis, similar tools were recently developed using this powerful technology. One such tool is Cellpose developed in 2021. Our next step is to compare our tool with Cellpose, in terms of recognition power and speed of processing. We will also look at how these tools fare against the human expert in many tissue images. An initial comparison reveals that Cellpose is more accurate in evaluating cell areas, but it is much slower than our OpenCV app.

Acknowledgement: This work is funded from the research grant NSF Award 2048425.

References

[1] J. Clevenger et al. "Network analyses reveal shifts in transcript profiles and metabolites that accompany the expression of SUN and an elongated tomato fruit." In: *Plant Physiology* 168 (2015), pp. 1164–1178. DOI: http://dx.doi.org/10.1104/pp.15.00379.

[2] Itseez. *Open Source Computer Vision Library*. https://github.com/itseez/opencv. 2015.

[3] E. van der Knaap, I. Goldman, and S. Visa. *NSF Grant, Robust organ patterning by OFPs, TRMs and SUNs in plant morphogenesis*. 2020.

[4] G.R. Rodriguez et al. "Distribution of SUN, OVATE, LC, and FAS in the Tomato Germplasm and the Relationship to Fruit Shape Diversity." In: *Plant Physiology* 156 (2011), pp. 275–285.

[5] C. A. Schneider, W. S. Rasband, and K. W. Eliceiri. "NIH Image to ImageJ: 25 years of image analysis." In: *Nature Methods* 9.7 (2012), pp. 671–675. DOI: http://dx.doi.org/10.1038/nmeth.2089.

[6] E. Sierra-Orozco et al. "Identification and characterization of GLOBE, a major gene controlling fruit shape and impacting fruit size and marketability in tomato." In: *Hort. Res.* 138 (2021).

[7] C. Stringer et al. "Cellpose: a generalist algorithm for cellular segmentation." In: *Nature methods* 18.1 (2021), pp. 100–106.

[8] S. Wu et al. "The control of tomato fruit elongation orchestrated by sun, ovate and fs8.1 in a wild relative of tomato." In: *Plant Science* (2015), pp. 99–104.

Using Problem-Driven Learning to Teach Computational Thinking in CS1[*]

Rocky K. C. Chang

Department of Computer Science
Calvin University
Grand Rapids, MI 49546
rocky.chang@calvin.edu

Abstract

This experience report describes the author's use of the classic man-cabbage-goat-wolf (MCGW) riddle to teach the main elements of computational thinking in CS1. Instead of teaching each computational thinking element individually, in the process of solving the MCGW problem the students are able to understand the process of abstracting a problem to a representation that can be understood by computer, finding a suitable algorithm to solve the abstracted problem, and decomposing the problem into different logical and more manageable parts. The survey responses from two recent classes clearly validate the effectiveness of this problem-driven approach even for a large class of very diverse backgrounds.

1 Introduction

Computational thinking (CT) has long been recognized to be a necessary skill in this information era [9]. There have been many initiatives in making CT as part of the K-12 education. For example, Switzerland recently introduced a national curriculum, called Lehrplan 21, mandating Computer Science Education [4]. In the higher education, on the other hand, a lot of effort has been

[*]Copyright ©2024 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

spent on teaching CT to first-year, non-CS majors [6], such as Science majors [3]. This paper concerns teaching CT to a CS1 class which consists of 80% CS majors and 20% non-CS majors. Moreover, a majority of the CS majors did not have any prior exposure to CT and computer programming.

Recently, there have been reflections on how to deliver CT more effectively mostly to the K-12 education. Denning points out that the current definition of CT is still vague and confusing [2]. This causes the teachers in the field to wrestle with what to deliver and how to assess CT. A more fundamental challenge is to substantiate the benefit of CT to everyone. Yaşar suggests that the problem lies with "linking CT to electronic computing devices and equating it with thinking by computer scientists" [10]. In this paper, by CT, the author means data abstraction, algorithm design, and problem decomposition—the three major tools for solving problems using a computational approach.

In the rest of this paper, the author will first describe the overall approach to teaching CS1 on CT. After that, a problem-driven learning approach using the classic man-cabbage-goat-wolf to teach the entire problem-solving process is introduced. Finally, survey results are presented to evaluate the effectiveness of this problem-driven approach.

2 A Problem-Driven Approach

The course to be presented in this paper is a CS1 course. Every CS Freshman is required to take this course. The class size ranges from 150 to 170 taught in two sessions. The main goal of this course is to equip them with CT and practical skills to solve problems computationally. This course is expected to lay a good foundation for them to study other CS courses. Although some students already have experience on computer programming or other related knowledge before enrolling into this course, most of them do not. Moreover, around 20% of them major in the business side of Financial Technology and Accountancy. Learning CS subjects requires them to change their thinking process and learning approach.

Owing to the students' very diverse backgrounds, the major challenge is to find an effective approach for all students to acquire CT and practical skills. The author has been experimenting with several approaches, ranging from a programming-oriented approach to an algorithm-oriented approach. Starting from several years ago the author devised a *problem-driven approach* which by far is the most effective one.[1] In a nutshell, this approach demonstrates the major elements in CT through the process of solving problems. These problems must be big enough to apply most, if not all, of the major components.

[1]This approach is more commonly known as problem-based learning [7]. Problem driven approach is used in this paper to emphasize that the *entire* course is based on this approach.

However, the problem cannot be too difficult for the students. The author has used the man-cabbage-goat-wolf (MCGW) problem [1] and the stock span problem [8] which meet the aforementioned requirements. Due to the page limitation, this paper reports the experience only for the MCGW problem.

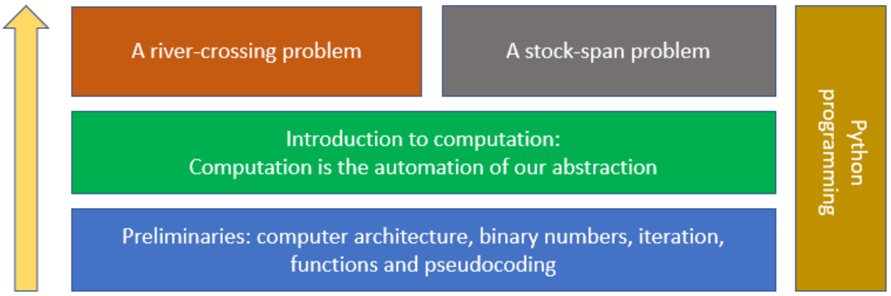

A river-crossing problem
A stock-span problem
Introduction to computation:
Computation is the automation of our abstraction
Preliminaries: computer architecture, binary numbers, iteration,
functions and pseudocoding
Python programming

Figure 1: The course structure of CS1. The vertical arrow indicates that the horizontal subject blocks are taught bottom up. The vertical block of Python programming is taught throughout the course.

Figure 1 shows the structure of CS1. There are four major blocks to cover: preliminaries, the meaning of computation, and the two problems (MCGW and stock span). In addition to that, the author also teaches them the basics of Python programming, so that they could code the solutions to the two problems. Except for the first two weeks, the author covers the four blocks and Python programming in parallel. The author covers the four blocks using a bottom-up approach. First of all, the students are taught computer architecture, number representations in computers, iterations, and functions. Students often have problems with using for loops and functions. An early exposure of these two important constructs will help them master them.

Teaching CT starts from the block on Introduction to Computation. The key question to answer in this block is "What is computation?" The answer to that is "**Computation is the automation of our abstractions**" [9]. The author in [5] uses a simple and yet revealing example to illustrate the meaning of computation. The example is to develop a function to add a number of multidigit numbers. The students are shown that the starting point to solve the problem is a given addition table for two single digits. The example thus introduces how a complex problem can be solved by building functions based on a simple two-digit addition table. Moreover, the "addition" could be replaced by any operation. Therefore, the abstraction (i.e., understanding what the table represents) is performed by us. On the other side, the machine, which is oblivious to the meaning of the data, carries out the mechanical calculation to produce the expected results, thus the automation part.

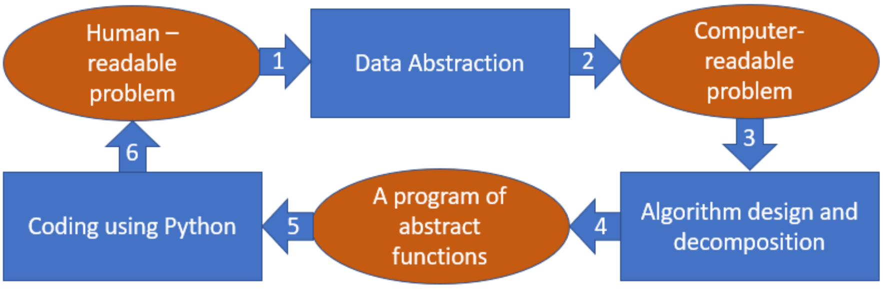

Human – readable problem
1
Data Abstraction
2
Computer-readable problem
3
Algorithm design and decomposition
4
A program of abstract functions
5
Coding using Python
6

Figure 2: The problem-solving cycle. The cycle starts with the human-readable problem and ends with a Python program that solves the human-readable problem.

Figure 2 shows the entire problem-solving cycle that covers the three major elements in CT: data abstraction, algorithm design and problem decomposition.

1. The goal of data abstraction is to transform a human-readable problem (input (1)) to a computer-readable problem (output (2)). This step is crucial to solving many problems, including the MCGW problem. However, this step could be trivial for other problems, such as Tower of Hanoi.

2. The goal of the algorithm design is to design an efficient algorithm to solve the computer-readable problem obtained from the data abstraction. The algorithm required for solving the MCGW problem is a shortest-path algorithm. However, this algorithm is too difficult for first-year students to understand the importance of time complexity. The author therefore chooses the stock span problem for the algorithm design.

3. The problem decomposition concerns how to decompose the problem into a set of "smaller" problems, each of which could be tackled independently from each other. Many of these smaller problems come from the data abstraction process, while others from the algorithm design. The MCGW problem is a good example to illustrate decomposition. We put the algorithm design and decomposition in the same block because they can be carried out at the same time. The output of this block is a pseudocode program consisting of abstract functions.

Finally, the abstract functions are coded to solve the human-readable problem at the end of this problem-solving cycle.

3 The MCGW problem

The MCGW problem is a classic riddle that has been told in different ways. It is simple to understand and at the same time is sufficient for demonstrating the entire problem-solving process. The classic problem calls for a man to bring a cabbage, a goat and a wolf from the east side of a river to the west side in a smallest number of trips. Several constraints make this problem harder to solve: (1) Besides the man, the boat can accommodate at most one more party. (2) The goat and the wolf cannot be left alone without the man. (3) The cabbage and the goat cannot be left alone without the man.

This paper reports the author's two recent years of experience. The author covered the data abstraction and decomposition process in the class and assigned the coding task to the students as a class project in the first year. Besides the code, the students were also required to hand in a two-page report to document the entire problem-solving process. In the second year, the author covered it in the class and also made the code available. The author assigned a three-couple river crossing problem, which is an extension to the MCGW problem, as a class project.

3.1 Data abstraction

The data abstraction phase is a crucial step in solving this problem. In going through the entire problem-solving process, the students have learned a number of useful abstraction techniques. First of all, the students learn how to abstract away nonessential details about the problem and retain those that are essential to solving the problem. Second, the students pick up the idea of modeling the problem using a 4-tuple (state): (E/W E/W E/W E/W) which indicates the respective locations (E: east; W: west) of the man, cabbage, goat and wolf. Third, out of the 16 distinct states, they apply the problem constraints to classify the states into a set of legal states and a set of nonlegal states. Finally, they connect those legal states together to form a ten-node bidirectional graph as depicted in Figure 3. At the end of this phase, the problem is abstracted to finding a shortest-path on the graph with the starting state ($EEEE$) and the ending state ($WWWW$).

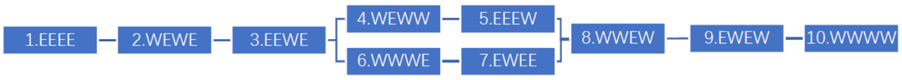


Figure 3: A graph representation of the data in the MCGW problem. The graph consists of 10 nodes and 10 bi-directional links. Each node is a state of the problem. A link connecting two nodes if they can transition to each other.

Throughout this process, the students are also exposed to a number of abstract data types: *boolean* for the locations, *set* for the set of legal/illegal states, *sequence* for the state and the shortest path, and *graph* for the interconnection of the legal states. Those students who have prior experience on computer programming are often confused about the relationship of the abstract data types and the data types supported by a programming language.

3.2 Algorithm design and decomposition

To solve the shortest-path problem, most of the students use the code provided to them, because devising such algorithm is out of the scope of this introductory course. More "advanced" students apply the breadth search algorithm to solve the problem. The author also does not require the student to evaluate the time complexity of the provided algorithm which will be taught in the stock span problem.

After abstracting the MCGW problem to a shortest-path problem, the students are ready to develop a solution by decomposing it into a set of subproblems. It is not hard for them to recognize the need to form the graph, to find the shortest path, and to print out the solution in a human readable format. Moreover, when solving the graph-forming problem, they need to pick up two more important skills. First, they need to decide the input for solving this problem which is either the set of all possible states or the set of legal states. Second, they learn how to further decompose the graph-forming problem. At this point, they are taught two ways of creating a graph: adjacency list and adjacency matrix. Since the graph is sparse, they use the adjacency-list method, and they learn to identify a subproblem of finding the neighboring states for each legal state. The solution to each subproblem is expressed as a function. Figure 4 shows the abstract functions to solve the problem.

3.3 Coding with Python

As mentioned before, Python programming is taught in parallel to the CT elements. By the time the students are required to code the solutions to the MCGW problem, they have already learned various Python built-in data types, such as list, tuple, set, and dictionary. At this stage, the students learn the best way to implement the abstract data types identified in the data abstraction phase. It presents less problem for them to implement boolean, sequence, and set using the corresponding Python data types with the same names. To implement the graph, the students are first asked to implement the graph in the form of adjacent list with Python list. Since each state is a 4-tuple of characters, a single list cannot implement the graph. It turns out that adding another list

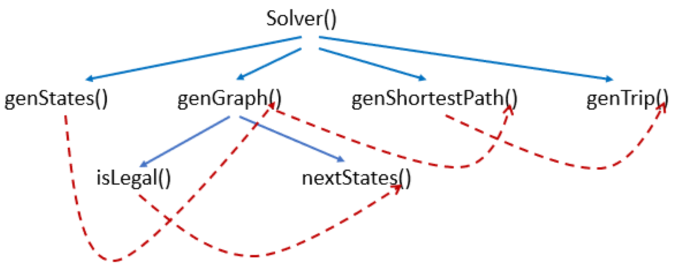


Figure 4: The structure of abstract functions to solve the MCGW problem. A solid blue line AB shows that function A calls function B. A dotted red line AB shows that the output of executing function A is passed to function B as an argument.

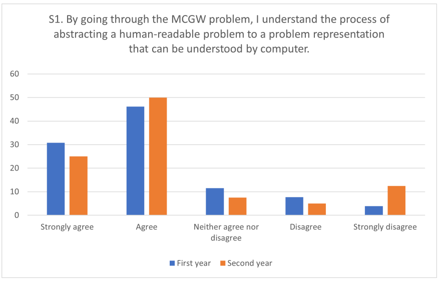
will be able to implement it. The students are then shown that Python dictionary can implement the graph in a much more straightforward manner. This self-exploration process helps them appreciate the dictionary data type.

4 Evaluation

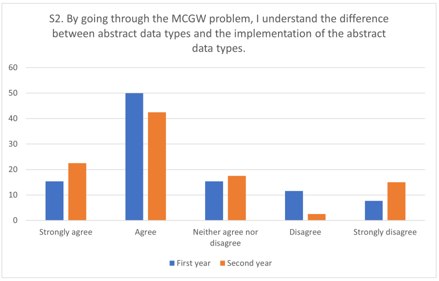
This section presents the evaluation results for two years. Both classes were taught the MCGW problem with several differences. First, although the MCGW problem was presented in both classes, it was given as a class project only for the first year. The students were required to code the solution by themselves. In the second year, the class project is an extended version of the MCGW problem. They were given the code for the MCGW problem as the base.

A set of 6 statements (S1, S2, $\cdots$, S6) were given to both classes in an online survey. The statements are given in Figure 5, 6, and 7. For the first year, 26 responses were received out of around 140 students (response rate: 19%). For the second year, 40 responses were received out of around 160 students (response rate: 25%). The difference in the response rates is not too large to compare them. Furthermore, the author could reach only the CS majors in the first year but not the students in the business side of Financial Technology who generally need more help. Therefore, the survey results could possibly be biased by these uncontrollable factors.

Figure 5a shows the responses to S1. Around 70% of the responses agree or strongly agree that the MCGW problem helps them understand the process of abstracting a human-readable problem to a machine-readable problem. The results for both classes are surprisingly similar. More than 10% in the second year strongly disagree to this statement. By examining the individ-

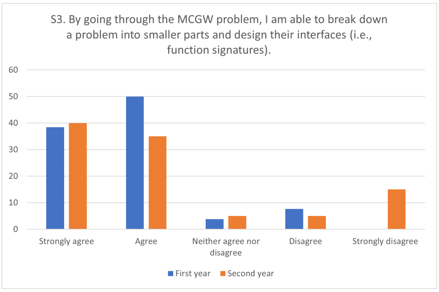


(a) Survey results for statement 1.

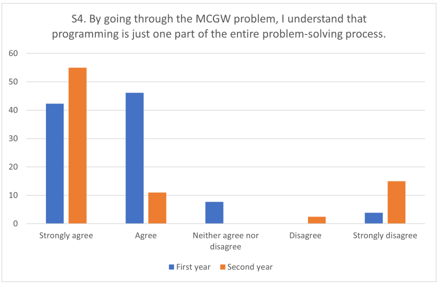


(b) Survey results for statement 2.

Figure 5: Survey results for statement 1 and 2. The vertical axis is the percentage of the responses.



(a) Survey results for statement 3.

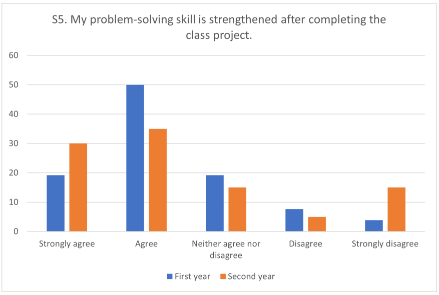


(b) Survey results for statement 4.

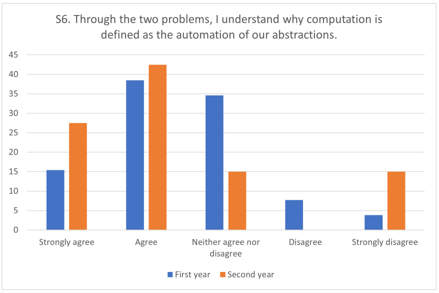
Figure 6: Survey results for statement 3 and 4. The vertical axis is the percentage of the responses.

ual responses, the author found that five responses rated all statements with ‘strongly disagree.’ Therefore, there is always more than 10% ‘strongly disagree’ responses in other figures. Moreover, S2 is designed to measure their understanding of abstract data types. The results in Figure 5b are less favorable, but there are still more than 60% in ‘strongly agree’ and ‘agree’.

The next two figures, Figure 6a and Figure 6b, are overwhelmingly positive for S3 and S4, respectively. S3 assesses whether the students understand the difference between abstract data types and their implementation through the MCGW problem. Almost 40% of them strongly agree. For the first class, almost 90% of them strongly agree or agree to this statement. The results for the second year would be similar if excluding the five always-strongly-disagree responses. The responses to S4 are even more positive. By working through the entire problem-solving process, it is not difficult for students to



(a) Survey results for statement 5.



(b) Survey results for statement 6.

Figure 7: Survey results for statement 5 and 6. The vertical axis is the percentage of the responses.

see that coding is just one of the steps.

As shown in Figure 7a, their responses to S5 about the class project is not as positive as for the last two statements. They are more similar to that for S1. There are still between 65% and 70% indicating 'strongly agree' and 'agree'. Finally, the last statement S6 is to link the MCGW problem and the stock span problem (not discussed in this paper) to the understanding of computation defined as the automation of our abstractions. We unfortunately cannot separate the results for the MCGW problem. Here we see a more significant distinction between the two years. The second year that was attended only by CS students responded much more positively in the category of 'strongly agree'. For the first year, slightly over 50% (strongly) agree to that statement.

5 Conclusions

In this paper, the author presented his two years of experience of using a problem-driven learning approach to teach a CS1 course on the main elements in CT: data abstraction, algorithm design, problem decomposition, and coding. The student survey results from 66 students in two years indicate that the approach is effective. By going through the MCGW problem, the students are taught the whole process of problem-solving and understand the meaning and importance of data abstraction, algorithm design, and problem decomposition. Future exploration of this approach includes trying different kinds of problems and introducing more opportunities for student feedback at different milestones in the project.

Acknowledgement

The author would like to thank Prof. Joel Adams of Calvin University to encourage me to submit this work and to help me improve the readability of this paper.

References

[1] *Brain Teasers: The Wolf, Goat, and Cabbage*. `https://illuminations.nctm.org/BrainTeasers.aspx?id=4992`. 2018.

[2] Peter J. Denning. "Remaining Trouble Spots with Computational Thinking". In: *Commun. ACM* 60.6 (May 2017), pp. 33–39. ISSN: 0001-0782. DOI: `10.1145/2998438`. URL: `http://doi.acm.org/10.1145/2998438`.

[3] S. Hambrusch et al. *Teaching Computational Thinking to Science Majors*. `https://www.cs.purdue.edu/homes/cmh/distribution/PapersChron/secant.pdf`. 2018.

[4] Anna Lamprou and Alexander Repenning. "Teaching How to Teach Computational Thinking". In: *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE 2018. Larnaca, Cyprus: ACM, 2018, pp. 69–74. ISBN: 978-1-4503-5707-4. DOI: `10.1145/3197091.3197120`. URL: `http://doi.acm.org/10.1145/3197091.3197120`.

[5] Hector Levesque. *Thinking as Computation a First Course*. Cambridge, MA: MIT Press, 2012.

[6] Tun Li and Ting Wang. "A Unified Approach to Teach Computational Thinking for First Year Non-CS Majors in an Introductory Course". In: *IERI Procedia* 2 (2012), pp. 498–503.

[7] Michael J. O'Grady. "Practical Problem-Based Learning in Computing Education". In: *ACM Trans. Comput. Educ.* 12.3 (2012).

[8] *The Stock Span Problem - GeeksforGeeks*. `https://www.geeksforgeeks.org/the-stock-span-problem/`. 2018.

[9] Jeannette M. Wing. "Computational Thinking". In: *Commun. ACM* 49.3 (Mar. 2006), pp. 33–35. ISSN: 0001-0782. DOI: `10.1145/1118178.1118215`. URL: `http://doi.acm.org/10.1145/1118178.1118215`.

[10] Osman Yaşar. "A New Perspective on Computational Thinking". In: *Commun. ACM* 61.7 (June 2018), pp. 33–39. ISSN: 0001-0782. DOI: `10.1145/3214354`. URL: `http://doi.acm.org/10.1145/3214354`.

Reviewers — 2024 CCSC Mid West Conference

Imad Al Saeed	Saint Xavier University, Orland Park, IL
David Bunde	Knox College, Galesburg, IL
Jennifer Jo Coy	Ball State University, Muncie, IN
Khaled Elzayyat	Salt Lake Community College, Salt Lake, UT
Paul Gestwicki	Ball State University, Muncie, IN
Hector Hernandez	Benedictine University, Waukegan, IL
Deborah Hwang	University of Evansville, Evansville, IN
David Largent	Ball State University, Muncie, IN
Jeff Lehman	Huntington University, Huntington, IN
Edward Lindoo	Regis University, Denver, CO
Jean Mehta	Saint Xavier University, Orland Park, IL
Michael Rogers	University of Wisconsin Oshkosh, Oshkosh, WI
Dave Surma	Indiana University South Bend, South Bend, IN
Paul Talaga	University of Indianapolis, Indianapolis, IN
James Vanderhyde	Saint Xavier University, Chicago, IL
Valerie Vann	Florida State College, Jacksonville, FL