The Journal of Computing Sciences in Colleges

Papers of the 18th Annual CCSC Southewestern Conference

March 28th-29th, 2025
University of San Diego
San Diego, CA

Bin Peng, Associate Editor
Park University

Megan Thomas, Regional Editor
California State University, Stanislaus

Volume 40, Number 9 **April 2025**

The Journal of Computing Sciences in Colleges (ISSN 1937-4771 print, 1937-4763 digital) is published at least six times per year and constitutes the refereed papers of regional conferences sponsored by the Consortium for Computing Sciences in Colleges.

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

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

Bryan Dixon, President (2026), bcdixon@csuchico.edu, Computer Science Department, California State University Chico, Chico, CA 95929.

Shereen Khoja, Vice President/President-Elect (2026), shereen@pacificu.edu, Computer Science, Pacific University, Forest Grove, OR 97116.

Abbas Attarwala, Publications Chair (2027), aattarwala@csuchico.edu, Department of Computer Science, California State University Chico, Chico, CA 95929.

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

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

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

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

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

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

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

Ben Tribelhorn, Northwestern Representative (2027), tribelhb@up.edu, School of Engineering, University of Portland, Portland, OR 97203.

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

Mika Morgan, South Central Representative (2027), mikamorgan@wsu.edu, Department of Computer Science, Washington State University, Pullman, WA 99163.

Karen Works, Southeastern Representative (2027), keworks@fsu.edu, Department of Computer Science, Florida State University - Panama City Panama City, FL 32405

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

Serving the CCSC: These members are serving in positions as indicated:
Bin Peng, Associate Editor, bin.peng@park.edu, Computer Science and Information Systems, Park University, Parkville, MO 64152.
Brian Hare, Associate Treasurer & UPE Liaison, hareb@umkc.edu, School of Computing & Engineering, University of Missouri-Kansas City, Kansas City, MO 64110.
George Dimitoglou, Comptroller, dimitoglou@hood.edu, Department of Computer Science, Hood College, Frederick, MD 21701.
Megan Thomas, Membership System Administrator, mthomas@cs.csustan.edu, Department of Computer Science, California State University Stanislaus, Turlock, CA 95382.
Karina Assiter, National Partners Chair, karinaassiter@landmark.edu, Landmark College, Putney, VT 05346.
Ed Lindoo, UPE Liaison, elindoo@regis.edu, Anderson College of Business and Computing, Regis University, Denver, CO 80221.
Deborah Hwang, Webmaster, hwangdjh@acm.org.

CCSC National Partners

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

Gold Level Partner

Rephactor
ACM2Y
CodeGrade
GitHub

Silver Level Partner

CodeZinger

Welcome to the 2025 CCSC Southewestern Conference

On behalf of the University of San Diego, we extend a warm welcome to the attendees of the 18th Annual CCSC Southwest Region Conference. We are honored to host the conference on March 28-29, 2025. The Conference Steering Committee is grateful for the authors, presenters, speakers, attendees, and students participating in this year's conference.

This year we received 13 paper submissions on a variety of topics, of which 7 papers were accepted for presentation in the conference. Multiple reviewers, using a double-blind paper review process, reviewed all submitted papers for the conference. The review process resulted in an acceptance rate of 54%. In addition to paper presentations, we are looking forward to a wide variety of lightning talks, nifty tool and assignments talks, panels/tutorials and student posters. We truly appreciate the time and effort put forth into the reviewing process by all the reviewers. Without their dedicated effort, none of this would be possible.

We hope you enjoy the conference and take the opportunity to interact and engage with your colleagues and leave with new connections and knowledge!

Sophia Krause-Levy and Saturnino Garcia
University of San Diego
Co-Chairs, CCSC:SW 2025

2025 CCSC Southewestern Conference Committee

Conference Co-Chairs
Sophia Krause-Levy ..... University of San Diego
Sat Garcia ..... University of San Diego
Papers Chair
Megan Thomas ..... California State University, Stanislaus
Speakers Chair
Michael Shindler ..... University of California, Irvine
Posters Chair
Todd Gibson ..... California State University, Chico
Panels/Tutorials Chair
Olivera Grujic ..... California State University, Stanislaus
Lightning Talk Chair
Shirin Haji Amin Shirazi ..... University of California, Riverside
Partner's Chair
Michael Shindler ..... University of California, Irvine
Nifty Tools & Assignments Chair
Joshua Gross ..... California State University, Monterey Bay

Regional Board — 2025 CCSC Southewestern Region

Regional Chair
Joshua Gross (acting) ..... California State University, Monterey Bay
Regional Representative
Michael Shindler (2023) ..... University of California, Irvine
Treasurer/Registrar
Michael Doherty (2021) ..... University of the Pacific
Current Conference Co-Chairs
Sophia Krause-Levy ..... University of San Diego
Sat Garcia ..... University of San Diego
Past Conference Chairs
Joshua Gross (2024) ..... California State University, Monterey Bay
Webmaster
Megan Thomas (2021) ..... California State University, Stanislaus

Treachery and Deceit: Detecting and Dissuading AI Cheating*

William Kerney
Clovis Community College
Fresno, CA 93730
william.kerney@cloviscollege.edu

Abstract

Last semester, 75% of the author's data structures students were caught cheating at least once, with Generative AI technologies being the most common means by which they cheated. While it may be tempting to move back to in-person pen-and-paper evaluations to ensure students have retained material, the author has found is possible to detect and discourage the use of cheating via various tricky methods. Finally, the author looks at attempts by students to conceal their use of AI in cheating, and how successful off the shelf AI detection tools are at finding the use of AI in coding assignments before and after being rewritten by hand.

1 Introduction

Cheating in computer science has been a topic that has been studied for a long time. The ACM Digital library shows over 13,000 papers tagged with "cheating" or "plagiarism" [1]. Historical approaches to detecting cheating involving similarity detection (comparing different students' submissions to see if they are the same) such as with MOSS [12] or anomaly-detection systems that look for unusual patterns in style, typing speed, backspaces, and so forth [5]. More recently, with the rise of AI cheating, automated AI detection systems have been developed [3] with mixed results [2, 6, 7].

AI cheating, which this paper will define as "any student use of AI contrary to the directions of their instructor", has been shown to harm student learning outcomes due to the mental struggle of working with tough materials being an essential part of the learning process [13]. Unlike copying off of friends or paying people to do assignments, which are common cheating approaches in the past [14], AI cheating is much easier to do – it is simply a matter of hitting the tab key with Microsoft Copilot [10] integrated into an IDE or pasting a homework assignment into Google Gemini [8] or OpenAI ChatGPT [11]. In light of this new reality, professors have to think about how to restructure their assignments and assessments so as to minimize AI cheating [4].

This paper is an experience report going over some novel tactics the author developed to detect and discourage AI cheating. By using these approaches, the percentage of students cheating using AI has dropped significantly.

The approaches the author took to discourage cheating are as follows:

1. Exploit the Weakness of AI
2. Dropping Traps for AI
3. Nega-Grade Discipline Systems
4. Detecting Stealthing

2 Methodology

This experience report is based on the experiences the author had with detecting and dissuading students from AI Cheating in 29 sections of classes, across five computer science classes and two information science / game development classes during the six semesters from Spring 2023 to Fall 2024, including summers. Average attendance per class was about 35 students. These classes encompassed the entire lower division computer science curriculum of Clovis Community College and a CS0 class at Fresno State, with approximately one thousand students total across the sections. Half the sections were online, half in person. The online sections had far more significant problems with cheating, with 75% of the online students in Spring 2024 caught cheating, but only 10% of the in-person. They were caught using the techniques described in this paper along with a confirmation pass done with the commercial AI detector GPTZero [9].

Detecting cheating, especially AI Cheating, can be controversial [2, 6, 7]. Some students deny using AI to cheat. To get around this issue, AI Cheating is only considered "caught" for this paper by either a student admitting guilt or at least not denying it when marked down on an assignment. Validation of the author's methods was done by giving an anonymous survey to all students to ask if they cheated with AI. The estimates of cheating in found in the Results Section were fairly accurate, with a false negative rate of 12%, assuming students

were being honest on the surveys. To validate off the shelf AI detectors, the author asked several former students to create human-made and AI-generated solutions to various homework projects to see how the AI detection systems held up against assignments known to be categorized correctly.

3 Results

Different approaches were taken to try to detect and dissuade the use of AI Cheating at the two institutions. A description of each approach, and their relative effectiveness, will be discussed in each bullet point below.

3.1 Exploit Their Weakness

Generative AI tools have an accuracy rate of roughly 80% [9], with the accuracy depending on how difficult the question is and how much training material the AI has to work from. As such, for any given pairing of an assignment, a generative AI tool a professor can run the AI tool over the assignment and see what sort of answers students trying to cheat will turn in. For online multiple choice quizzes, a canny professor can pick out a subset of questions that AI always gets wrong, and then build quizzes from them. When students try cheating with AI and get a 0%, the problem becomes self-correcting without any need for disciplinary measures. The cheating carries its own punishment, and they stop using it – without any paperwork needed.

When presenting this approach at a special NSF Workshop on AI Cheating in Denver in August 2023, the author was given feedback from other professors that it would take too much time to sort through all of one's assignments and quiz questions to find those that AI can't answer, to which the author here offers two solutions:

- Only take the effort for a select few questions on a quiz or exam instead of the entire assessment and then scan for students who get the selected questions wrong all the same way that AI does. Any student that matches that signature gets disciplined (see Discipline section below). This approach does not take very much time, and gives one good confidence that, if for example they gave the AI's specific incorrect response five questions in a row, that they probably cheated.
- If you have a question test bank, you can install one of the AI tools that will answer questions directly from an LMS [15], have it answer all the questions, and then see which set of questions it got wrong. This allows you quickly get a list of AI-hostile questions with only a minute or two of effort. You can put together entire exams of nothing but these questions.

Results: This approach is excellent at detecting students using a specific GenAI tool, with the other methods below confirming that students caught using this approach were in fact cheating using AI. However, if different AI tools give different results, it becomes too time consuming finding problems they all get wrong. But if all students are predominantly using, for example, ChatGPT to cheat, then this approach works well. The author would catch about 10% of an online section cheating on a midterm each semester using this approach.

Not one student caught cheating by these approaches denied using AI to cheat. Analysis of those students caught cheating on later exams shows that approximately three-quarters of cheaters stopped using AI after getting burned by it once, presumably since it hurt their grade and they stopped trusting it to solve problems for them. The remaining quarter of cheating students continued to cheat and tanked their grade even further each time, and finished with a low grade in the class.

The best advantage to this approach is that it allows one to continue doing online exams in a world where many professors have gone back to pen and paper [16], and if students just shoot themselves in the foot by cheating, there is no need for paperwork or contentious disciplinary meetings with the student, or anything like that. They cheat – they get a 0%. The problem solves itself with a minimum of hassle.

3.2 Dropping Traps for AI

The most common way students cheat using AI is to copy the problem wholesale into an AI and then paste the results back to their professor. This can be detected by putting a **trap** for the AI in the problem description. It is possible in any system that supports HTML questions to embed zero-point font text into the question. A human cannot see the text, but ChatGPT will. It is simple to add something innocuous to the problem so that even if students skim the results they won't notice anything out of place, but when a professor reads their students' submissions they can easily see who cheated.

For example, students might be given a couple paragraphs of description of a homework problem that boils down to, "Write a function that takes a string as input and outputs a string that upper cases every character in an odd index and lower cases every character in an even index". The professor works in a couple extra words in zero-point font so that what ChatGPT sees is actually: "Write a function *named twiddles* that takes a string as input and outputs a string that upper cases every character in an odd index and lower cases every character in an even index". So when the professor is looking through the responses, every single student who has a function named "twiddles" automatically gets flagged for cheating.

Another approach is to carefully control what exact information is in the problem description. For example, a professor could create a problem that specifies that a default value should be "the mascot of your university", and imply that the student is at UC San Diego. An AI would fill in "King Triton" for the default value, but no human (outside of UCSD students) would do so, instead filling in the name of their own mascot, making it easy to catch.

Results: This approach is highly effective, catching 75% of the author's online students in Spring 2024. Only a single student disputed that he had used AI to cheat, claiming instead that he had *a friend cheat for him.* This friend, of course, used AI to write the code and got caught.

Caution must be taken with this approach as visually impaired students or other who use screen readers will pick up the zero-point font.

3.3 Nega-Grade Discipline Systems

When the author began studying this issue, he believed that the main problem would be detecting the use of AI cheating in students. However, the above approaches were able to successfully detect 88% of the students who cheated in the class (based on the end-of-year survey). The real problem actually was not detection, but dissuading them from using it. It is so easy to use AI to cheat, that students who are struggling with a homework assignment often just reach for it even if they know they might get in trouble for it. Given that 75% of the author's students cheated in one semester, it was logistically impossible to send them all to disciplinary hearings, and in any event the author's objective was to have them quit cheating and learn the material instead. The author uses competency exams, so students have multiple attempts to demonstrate competency. The hope was that even if they got caught cheating on one exam, and were notified that they got caught, that they wouldn't do it on the next. The results of this approach were mixed. Indeed, the majority of the students who got caught cheating stopped cheating after they realized the author could catch them. However, about 10% of students kept cheating anyway, presumably because they had not learned anything due to their cheating, and couldn't solve the problems honestly. Their reasoning was that the worst that could happen to them was a 0 on a test that could be retaken, so why not cheat and hope they don't get detected?

In Fall 2024, the discipline system was changed so that cheating resulted in a *negative 10% grade* on the exam, worse than not taking the test at all, and this mark could not be removed. Results: This changed the mental calculus for cheating, and sharply reduced cheating to less than 2% of online assignments. A second offense resulted in being sent to the disciplinary committee; so far only one of the students has done so.

3.4 Detecting Stealthing

Based on the survey, 1 out of 8 cheaters got away with it by "stealthing" their code (hiding the AI origins of their code). The author recruited three former students[1] to explore how they would go about stealthing their code, and to see if it could still be detected by GPTZero. The students submitted 21 homework assignments they: A) did by hand, B) generated by AI, and C) stealthed the AI origins. GPTZero was run on all the submissions. GPTZero gives a score (between 0% and 100%) that can be thought of as the confidence that an AI wrote the text passed into it, with 60

The changes students made to conceal the use of AI most commonly involved changing the names of variables and functions, adding or removing comments, and revising loops to different equivalent versions.

Results: Despite multiple reports of inaccuracy [5][6][7] in 2023, GPTZero had 100% accuracy at classifying human and AI text before it was stealthed. All human text scored at 40% and below, with most below 15%. Before being stealthed, all AI generated files were detected with a score of 60% or higher. After being stealthed, only 9% of the assignments were detected by GPTZero as being AI generated. GPTZero is thus *not* effective at detecting stealthed code and other methods should be used such as approaches as described in [4] which work based on the AST of the submitted code. Caution must be taken with interpreting GPTZero scores as definitive proof of cheating, and should always be considered holistically with other factors

4 Discussion and Conclusion

AI cheating since Spring 2023 has become extremely prevalent. The good news is that 7 out of 8 students did not conceal their use of AI-generated code, and can be detected using software like GPTZero, or through clever tricks like embedding zero point font text in problem descriptions or cherrypicking questions that AI is known to get wrong. When students stealth their code to hide from AI detection, GPTZero stops being an effective tool and students might get lucky and detect and remove the traps from their code before submitting it. However, it will not help the AI answer questions better, and so the author's recommendation is to use a combination of all approaches to detect cheating or other AI detectors.

Once cheating is detected, the focus switches to deterrence. Experiments with using nega-grades, where students are worse off if caught cheating than if they did nothing, have been successful at changing the risk/reward balance of cheating. Additionally, setting up quizzes with questions that AI reliably get

[1]Thanks to Hazelton, Prado, and Red

wrong has been found to be effective at discouraging students from using AI without the paperwork and time overhead of sending students to disciplinary hearings, and reforms them into actually doing the work instead of cheating in the majority of cases.

References

[1] ACM. *ACM Digital Library*. https://dl.acm.org/action/doSearch?fillQuickSearch=false\&target=advanced\&expand=dl\&AllField=AllField\%3A\%28cheating\%29+OR+AllField\%3A\%28plagiarism\%29.

[2] Ian Bogost. "The First Year of AI College Ends in Ruin". In: *The Atlantic* (May 2023).

[3] Binglin Chen et al. "Plagiarism in the Age of Generative AI: Cheating Method Change and Learning Loss in an Intro to CS Course". In: *Proceedings of the Eleventh ACM Conference on Learning @ Scale*. Atlanta, GA, USA: Association for Computing Machinery, 2024, pp. 75–85. DOI: 10.1145/3657604.3662046.

[4] Carleton College. *AI-Resistant Assignments*. URL: https://www.carleton.edu/writing/resources-for-faculty/working-with-ai/ai-resistant-assignments/ (visited on 11/11/2024).

[5] Benjamin Denzler et al. "Style Anomalies Can Suggest Cheating in CS1 Programs". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2024. Milan, Italy: Association for Computing Machinery, 2024, pp. 381–387. DOI: 10.1145/3649217.3653626.

[6] Benj Edwards. *Why AI writing detectors don't work*. June 2023. URL: https://arstechnica.com/information-technology/2023/07/why-ai-detectors-think-the-us-constitution-was-written-by-ai/ (visited on 11/11/2024).

[7] Geoffrey A Fowler. *We tested a new ChatGPT-detector for teachers. It flagged an innocent student*. 2023. URL: https://www.washingtonpost.com/technology/2023/04/01/chatgpt-cheating-detection-turnitin/ (visited on 11/11/2024).

[8] Google. *Gemini*. URL: https://gemini.google.com/app/ (visited on 11/11/2024).

[9] GPTZero. *GPTZero*. URL: https://gptzero.me/ (visited on 11/11/2024).

[10] Microsoft. *Microsoft CoPilot*. URL: https://copilot.microsoft.com/ (visited on 11/11/2024).

[11] OpenAI. *ChatGPT*. URL: https://chatgpt.com/ (visited on 11/11/2024).

[12] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. "Winnowing: local algorithms for document fingerprinting". In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD '03. San Diego, California: Association for Computing Machinery, 2003, pp. 76–85. DOI: 10.1145/872757.872770.

[13] Esther Shein. "The Impact of AI on Computer Science Education". In: *Commun. ACM* 67.9 (Aug. 2024), pp. 13–15. ISSN: 0001-0782. DOI: 10.1145/3673428.

[14] Frank Vahid et al. "Impact of Several Low-Effort Cheating-Reduction Methods in a CS1 Class". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 486–492. DOI: 10.1145/3545945.3569731.

[15] Wizard. *Canvas Wizard*. URL: https://canvaswizard.co/ (visited on 11/11/2024).

Systematic Review on Gamified and Ungraded Pedagogical Approaches*

Dan Houston[1] and Miranda C. Parker[2]
[1] San Diego State University
San Diego, CA 92182
dhouston7516@sdsu.edu
[2] University of North Carolina, Charlotte
Charlotte, NC 20000
miranda.parker@uncc.edu

Abstract

With growing interest in alternative approaches to computing education aimed at improving student outcomes, we focused on two key strategies: gamified learning and ungraded pedagogy. These methods promote student engagement and intrinsic motivation by creating flexible, student-centered environments—gamification through game-like elements and ungraded pedagogy through a focus on mastery rather than performance. To explore how these approaches align and can be integrated, we conducted a systematic literature review of interventions combining both strategies. Our findings highlight key themes—*Reimagined Classroom Experience*, *Gamified Grades*, and *Platformized Class*—and provide insights into current practices, trends, and future research directions.

1 Introduction

In recent years, alternative approaches in education have gained traction for their potential to improve student outcomes by fostering engagement and motivation. Among these approaches, ungraded pedagogy and gamified learning

stand out for their capacity to create dynamic, student-centered learning environments. Ungraded pedagogy seeks to shift the focus from performance metrics to mastery, reducing student anxiety and promoting intrinsic motivation by eliminating traditional grading structures [3]. Gamified learning, which incorporates game-like elements such as points, levels, and challenges, aims to boost motivation and deepen engagement by transforming learning into an interactive experience [12]. By integrating these two approaches, the goal is to reduce external pressures associated with traditional grading, enabling a shift towards intrinsic motivation where the enjoyment of the learning process itself becomes the primary driver of deep engagement.

We chose to investigate the intersection of these two methods to better understand when they are used together and how they may complement each other in enhancing educational experiences. By systematically reviewing the literature, we aim to provide insights into the implementation, outcomes, and challenges associated with integrating gamified and ungraded pedagogy. Specifically, we address the following research question: *How are the concepts of gamification and alternative grading aligned and used in combination with one another?*

The following sections detail our methodology and findings, which are organized around three primary themes: *Reimagined Classroom Experience*, *Gamified Grades*, and *Platformized Class*. Each theme examines the potential for gamified and ungraded practices to foster engagement, motivation, and a supportive learning environment.

2 Methodology

Our methodology follows the guidelines for systematic reviews as proposed by Preferred Reporting Items for Systematic reviews and Meta-Analyses (PRISMA) [19]. We conducted searches across ACM Digital Library, IEEE xPlore, Scopus, World Wide Science, Science Direct, Eric, Wiley, ProQuest, and the university's library database. Each database was searched up to June 4th, 2024. We systematically searched the literature, following methods and suggestions from Booth et al. [4], identifying 380 articles that met our search terms in Table 1. We assessed the relevance of the studies found to our research question, following a two-stage screening process by one reviewer. In the first stage, we screened titles and abstracts, and in the second stage, we reviewed full-text articles using a standardized form and ultimately analyzed 18 publications.

Studies were included if they examined gamified or ungraded pedagogical approaches within educational contexts; were published, peer-reviewed studies, dissertations, or poster abstracts; involved any age group or education level;

Table 1: Search Terms

OR		OR
"game-based learning" "game elements" gamif*	AND	"alternative grading" "competency grading" "contract grading" "mastery grading" "specifications grading" "standards grading" "equitable grading" ungrad*

implemented an intervention applied or put into practice within the study; measured any type of outcome related to the intervention; were conducted in any country; were written in English; and focused on academic contexts, including in-school, after-school, and summer camp settings.

Exclusion criteria included studies without an intervention, such as theoretical papers; studies outside the education sector, such as athletics programs or other non-academic contexts; non-peer-reviewed presentations; full conference proceedings; full books or volumes; and studies not written in English.

Data extraction focused on author, year, sample size, methodology, key findings, and definitions of grading method and gamification. Studies were analyzed using thematic analysis [4].

3 Results

A total of 18 studies met the inclusion criteria. The earliest publication was from 2011 and the latest was from 2024, the year the search was performed. The results are grouped into three key themes: *Reimagined Classroom Experience*, *Gamified Grades*, and *Platformized Class*. Each theme is discussed in terms of methodologies, findings, and implications.

3.1 Reimagined Classroom Experience

The theme of *Reimagined Classroom Experience* encompasses a range of innovative pedagogical approaches that blend alternative grading methods with immersive, game-like learning environments [1, 5, 6, 7, 10, 13, 17, 18, 21]. These classrooms are designed to foster student engagement, motivation, and agency by rethinking traditional classroom roles and systems. Instructors are often positioned as *game masters* or *game designers*, each framing their roles in unique ways to guide, structure, and orchestrate the educational experience.

A *game master*, for instance, adjusts the narrative and difficulty of activities to keep students actively engaged, while a *game designer* establishes the underlying structures, challenges, and rewards that shape student interactions and learning pathways. Each of these roles integrates a balance of structure and flexibility, allowing students to take on meaningful challenges while developing autonomy and self-regulation.

Within these classrooms, alternative grading approaches—such as contract grading, mastery-based assessments, and competency-based grading—enable a shift from traditional metrics to more formative, student-centered systems. In this context, grading becomes part of the immersive learning environment, supporting risk-taking and learning from failure without the constraints of rigid grade penalties. For example, instructors using contract grading set clear criteria tied to specific achievements, such as "quests" or "side missions," enabling students to explore content through various paths. This framework aligns with self-determination theory [11], enhancing intrinsic motivation by fostering autonomy and competence in the learning process.

The studies under this theme demonstrate a shared focus on fostering immersion and flow—a psychological state of deep engagement and focus described by Mihaly Csikszentmihalyi [9]. To achieve flow, educators carefully balance task difficulty with student skill levels, creating a dynamic learning environment where students experience continuous growth without the frustration of overly challenging tasks or the boredom of under-stimulation. For example, Bryant's mathematics course frames assignments as quests and study groups as guilds, providing students with agency in how they approach tasks and interact with peers [6]. This gamified approach allows instructors to adjust challenges, ensuring that students remain within their Zone of Proximal Development [23].

The *Reimagined Classroom Experience* combines elements such as core game loops and side quests to enhance learning. Core game loops, as described by Caravella, involve cycles of challenge, feedback, and reward, mirroring formative assessment to reinforce mastery and build confidence [7]. Side quests, or optional tasks, encourage deeper engagement without grade penalties, fostering curiosity and reducing the pressures of traditional grading [7].

The studies reviewed in this theme provide evidence that gamified and game-based classrooms, structured through flexible grading and guided by immersive principles of game design, can enhance student engagement, motivation, and self-regulation. While alternative grading systems like contract and mastery grading may not always correlate directly with academic performance improvements, they consistently foster a sense of agency and resilience among students. This theme, *Reimagined Classroom Experience*, highlights the transformative potential of integrating game-based design principles into edu-

cational settings, suggesting that a balance of immersive elements, alternative grading, and structured challenges can create an enriching and ethical learning environment.

3.2 Gamified Grades

The *Gamified Grades* theme explores the transformative potential of integrating game mechanics into grading practices to improve student motivation, transparency, and engagement [8, 22, 24, 25]. The studies in this theme demonstrate a range of gamified grading approaches, including point systems, badges, and experience (XP), used to reframe student interactions with assessment.

Topîrceanu's research, for example, highlights the benefits of using an XP-based grading system in computer science courses, where students demonstrated higher attendance and increased participation in assignments. This system engaged students by incorporating role-playing elements, thereby creating an interactive and motivational grading experience that promoted academic involvement [22]. Similarly, Zhao's implementation of a points-based grading approach in teacher training programs allowed students to track their academic progress, which improved transparency and engagement; however, the study found limited improvement in actual academic outcomes, suggesting gamification may primarily enhance engagement rather than overall performance [25].

Cooper's study in early literacy education shows the community-building potential of digital badges, where badges served not only as motivators for young students but also facilitated communication among families, teachers, and administrators. This holistic integration of gamified elements fosters a collaborative and supportive learning environment, especially for younger learners [8]. On the other hand, West-Puckett's exploration of badging in higher education reveals a mixed response from students; while many appreciated the transparency and focus on skill development, others remained more comfortable with traditional grading methods, suggesting that gamified grading may not be universally preferred across educational levels [24].

In summary, gamified grading offers substantial potential to create engaging and transparent learning experiences. Although its impact on performance may vary, gamified grades can effectively support a more active learning environment when implemented with attention to student needs and preferences.

3.3 Platformized Class

The *Platformized Class* theme examines the role of digital platforms, such as Learning Management Systems (LMSs) and mobile technologies, in reshaping educational delivery [2, 14, 15, 16, 20]. These platforms facilitate alternative grading approaches and gamification through features like real-time feedback,

customizable assessment options, and progress tracking, allowing for a more individualized learning experience. The studies summarized in this theme offer insights into how platforms support active learning and student autonomy.

Beaudoin and Avanthey's longitudinal study in computer science courses demonstrates how a platform-centered approach, which includes real-time progress tracking and automated feedback, enhanced both engagement and academic performance over time. This study underscores the advantages of integrating digital tools with active learning to promote student skill development while highlighting the importance of maintaining in-person interaction to avoid platform fatigue [2]. Similarly, Holman's analysis of the GradeCraft LMS in an honors course shows how allowing students to choose their own assessment types can foster ownership and engagement, especially among highly motivated students, though the flexibility may not be equally effective for all learner groups [15].

Kime's use of dashboard tools in calculus classes allowed teachers to monitor individual progress and proactively address learning gaps, supporting a data-driven approach to individualized instruction that encouraged timely feedback and reinforced skill mastery [16]. Heath's study on iPad integration for students with disabilities highlights the role of platforms in promoting engagement and autonomy but emphasizes that teachers need targeted training to effectively incorporate such technology into lesson plans [14]. Finally, Petrovic-Dzerdz's gamified Moodle implementation demonstrates the impact of repeated practice and mastery-based progression, showing that students engaging frequently with the platform's gamified assessments performed better academically [20].

Overall, platform-based learning tools offer notable benefits in fostering engagement and personalization. However, the successful integration of these tools relies on educator support, balancing digital and human elements, and adapting platforms to meet diverse student needs.

4 Discussion

The literature reveals a growing interest in gamified and ungraded pedagogy, with evidence suggesting that both approaches can enhance engagement and motivation. Nonetheless, challenges persist, especially in scalability and consistency across diverse educational contexts.

For practitioners, adopting gamified and ungraded methods requires careful planning. Combining these approaches may be particularly effective for cultivating a student-centered environment. Educators are encouraged to consider both extrinsic and intrinsic motivators when designing gamified, ungraded activities.

Gamified and ungraded pedagogy offer promising avenues for increasing

student engagement and motivation. This review underscores the importance of continued research into the nuanced effects of these pedagogies and provides recommendations for implementing blended approaches to enhance student-centered learning.

References

[1] Dmitriy Babichenko et al. "Implementation and Assessment of Three Gamification Strategies Across Multiple Higher Education Disciplines". English. In: Last updated - 2024-08-27. Reading: Academic Conferences International Limited, Oct. 2019, pp. 41–47, XIII.

[2] Laurent Beaudoin and Loïca Avanthey. "How to help digital-native students to successfully take control of their learning: A return of 8 years of experience on a computer science e-learning platform in higher education". In: *Education and Information Technologies* 28.5 (2023), pp. 5421–5451. DOI: 10.1007/s10639-022-11407-8.

[3] Susan Debra Blum, ed. *Ungrading: Why Rating Students Undermines Learning (and What to Do Instead)*. First edition. West Virginia University Press, 2020.

[4] Andrew Booth, Diana Papaioannou, and Anthea Sutton. *Systematic Approaches to a Successful Literature Review*. Sage Publications, Jan. 2012.

[5] Jessica E. Broussard. "Playing Class: A Case Study of Ludic Pedagogy". PhD thesis. United States – Louisiana, 2011, p. 167. ISBN: 9798802787274.

[6] Albert Bryant. "Student Perceptions of Engagement With Secondary Mathematics in a Gamified, Standards-Based Grading System: A Basic Qualitative Study". In ProQuest Dissertations and Theses (2937271364). Ed.D. Dissertation. American College of Education, 2024.

[7] Elizabeth Caravella. "Teaching Gamefully: Proceduralizing the Classroom through Possibility Space Pedagogy". In ProQuest Dissertations and Theses (2243789108). Ph.D. Dissertation. George Mason University, 2019.

[8] Amy M. Cooper. "Teacher Perceptions of the Digital Badge in Kindergarten Reading Attainment". In ProQuest Dissertations and Theses (2211480168). Ed.D. Dissertation. Concordia University (Oregon), 2019.

[9] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. First edition. New York, NY, USA: Harper & Row, 1990.

[10] Mariam Dabbous et al. “Instructional educational games in pharmacy experiential education: a quasi-experimental assessment of learning outcomes, students’ engagement and motivation”. In: *BMC Medical Education* 23.1 (Oct. 2023), p. 753. ISSN: 1472-6920. DOI: 10.1186/s12909-023-04742-y.

[11] Edward L. Deci and Richard M. Ryan. *Intrinsic motivation and self-determination in human behavior*. Perspectives in social psychology. Includes bibliographical references (pages 335-358). New York: Plenum, 1985. ISBN: 0306420228.

[12] Sebastian Deterding et al. “From game design elements to gamefulness: defining "gamification"”. In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. MindTrek ’11. Tampere, Finland: Association for Computing Machinery, 2011, pp. 9–15. ISBN: 9781450308168. DOI: 10.1145/2181037.2181040.

[13] Daniel J. Harrold. “Game on: A qualitative case study on the effects of gamified curriculum design on student motivational learning habits”. In ProQuest Dissertations and Theses (1673159776). Ph.D. Dissertation. Robert Morris University, 2015.

[14] Katie Louise Heath. “Use and Integration of iPads with Students with Low Incidence Disabilities in Elementary Schools”. Ph.D. Dissertation. Syracuse University, 2018. URL: https://surface.syr.edu/etd/900.

[15] Caitlin Holman. “Building a Better Game: A Theory of Gameful Learning & the Construction of Student Personas with Agency”. In ProQuest Dissertations and Theses (2080292528). Ph.D. Dissertation. University of Michigan, 2018.

[16] Kristian Kime. “Insightful Learning Systems: Supporting STEM Education with Accessible Cyberlearning”. In ProQuest Dissertations and Theses (2577749089). Ph.D. Dissertation. Brandeis University, 2021.

[17] Mary Kruger. “A Comparative Study of Student Performance When Using Minecraft as a Learning Tool”. In ProQuest Dissertations and Theses (1881841972). Ed.D. Dissertation. Northcentral University, 2016.

[18] Lindsay Kistler Mattock. “Using Gamification to Overcome Anxiety and Encourage Play in the Graduate Classroom”. In: *ELearn* 2023.7 (July 2023). DOI: 10.1145/3609266.3603505.

[19] Matthew J. Page et al. “The PRISMA 2020 statement: An updated guideline for reporting systematic reviews”. In: *International Journal of Surgery* 88 (2021), p. 105906. ISSN: 1743-9191. DOI: https://doi.org/10.1016/j.ijsu.2021.105906.

[20] Maristela Petrovic-Dzerdz. "Gamifying Online Tests to Promote Retrieval-Based Learning". In: *International Review of Research in Open and Distributed Learning* 20.2 (2019). URL: https://doi.org/10.19173/irrodl.v20i2.3812.

[21] Shannon J. Saluga et al. "Inter-Twine-d: Combining Organic Chemistry Laboratory and Choose-Your-Own-Adventure Games". In: *Journal of Chemical Education* 99.12 (2022), pp. 3964–3974. DOI: 10.1021/acs.jchemed.2c00481.

[22] Alexandru Topîrceanu. "Gamified learning: A role-playing approach to increase student in-class motivation". In: vol. 112. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference, KES-20176-8 September 2017, Marseille, France. 2017, pp. 41–50. DOI: 10.1016/j.procs.2017.08.017. URL: https://www.sciencedirect.com/science/article/pii/S187705091731356X.

[23] Lev S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Massachusetts: Harvard University Press, 1978.

[24] Stephanie West-Puckett. "Making Classroom Writing Assessment More Visible, Equitable, and Portable through Digital Badging". In: *College English* 79.2 (2016), pp. 127–151. ISSN: 00100994, 21618178. URL: http://www.jstor.org/stable/44805914.

[25] George Zhao. "Using a Gamified Points-Based Grading System in Technology Courses for Pre-Service Teachers". In ProQuest Dissertations and Theses (2478622680). Ph.D. Dissertation. University of Houston, 2019.

Implementation of Alternatively-Graded, Gamified Pedagogy in a Data Structures Course*

Dan Houston[1] and Miranda C. Parker[2]
[1] San Diego State University
San Diego, CA 92182
dhouston7516@sdsu.edu
[2] University of North Carolina, Charlotte
Charlotte, NC 20000
miranda.parker@uncc.edu

Abstract

This paper presents a single-masked controlled study on the effectiveness of an alternatively-graded, gamified pedagogy in a Data Structures course at San Diego State University. The study compares an experimental group experiencing an alternatively graded and gamified assignment against a traditional control group to assess impacts on student engagement and perceptions of the classroom. The study found statistically significant results, with the experimental group responding with more positive feedback.

1 Introduction

In recent years, educators have increasingly explored innovative pedagogical approaches to enhance student engagement and learning outcomes in technical courses. Gamification and alternative grading systems are among these approaches, which aim to increase motivation and reduce the anxiety often

associated with traditional grading. This study investigates the impact of such an approach on students in a Data Structures course, a core component of computer science (CS) education that poses unique challenges due to its complexity and abstract content. The course is an introductory component of the CS curriculum, typically taken by students who are new to the major and have limited prior experience with CS concepts.

To assess the effectiveness of an alternatively graded, gamified (AGG) assignment, this paper presents a controlled study comparing an experimental group that participated in an AGG assignment to a control group that followed traditional teaching methods. Our primary research question is: *What is the impact of an alternatively graded, gamified pedagogy on student engagement and perceptions in a Data Structures course?*

2 Study Design

This study employed a single-masked, experimental design to evaluate the impact of AGG pedagogy against traditional grading methods in a Data Structures course at San Diego State University. Ethics board approval was obtained to ensure adherence to ethical standards, and transparency and voluntary participation were prioritized throughout the recruitment process. All students enrolled in the course were invited to participate, and those who consented were informed that their participation was optional and would not impact their grades. Non-consenting students completed standard course surveys for self-assessment but were excluded from data analysis.

Four course sections were randomly assigned to either the experimental group (AGG) or control group (traditional). The experimental group (two sections, 123 students, with 92 consenting) engaged with AGG assignments featuring narrative-driven tasks, mastery-based grading, and adaptive feedback. The control group (two sections, 129 students, with 102 consenting) followed traditional assignment formats and grading language. To minimize bias, students were not informed of their group assignment.

In this course, students were introduced to fundamental data structures, including arrays, linked lists, stacks, and trees. Assignments, exams, and online discussions comprised the primary course content. For the study, the linked lists unit was selected as the focus due to its relative difficulty for students and its central role in the curriculum. Post-surveys with open-ended questions were administered to collect feedback on students' perceptions on engagement, performance, expectations, and applicability of the course content.

3 Intervention Design and Implementation

This study contrasted AGG pedagogy in the experimental group with traditional approaches in the control group. Both groups followed identical course content, differing primarily in the linked list assignment framing, grading language, and feedback methods.

Previous approaches struggled to engage students, especially those not already familiar with CS (e.g., potentially underrepresented groups such as women in the field [4]). Prior assignments for this unit included the creation of a simulated Central Processing Unit (CPU) scheduler, which was more accessible to students already familiar with computers.

Following Morschheuser et al.'s guidelines for effective gamification design, we collaborated with the course instructor to brainstorm and refine a narrative-driven assignment approach [5]. For instance, we replaced the CPU scheduler assignment with a Monopoly-themed task, using Monopoly spaces as nodes in a linked list. This change aimed to provide a boundary object [8] that made abstract programming concepts more relatable, particularly for students less familiar with CS, thus making the content more equitable.

AGG Approach: In the experimental group, assignments were presented as quests within a cohesive storyline, designed to enhance engagement and learning through narrative. These quests emphasized mastery-based grading, an approach informed by the concept of Learning Edge Momentum (LEM), which posits that successful or unsuccessful learning experiences tend to reinforce themselves over time [6].

Our approach was designed during a brainstorming session that reviewed prior studies that paired gamified and alternative grading methods as interventions. For example, each assignment became part of an immersive storyline to enhance student engagement and learning [1, 7, 9]. These studies highlighted positive impacts on student motivation, engagement, and mastery, while also identifying potential challenges. This background guided our pedagogy choices, helping to integrate effective strategies and keep in mind known limitations. The experimental curriculum adopted gamification principles, including "Leveling Up," "Badge Collection," and "Dynamic Difficulty Adjustment," to create an engaging environment intended to facilitate Csikszentmihalyi's concept of "flow" [2].

Following Morschheuser et al.'s recommendations for iterative, user-centered design [5], the curriculum incorporated adaptive elements that responded to varying skill levels, providing mastery-oriented, motivational feedback. This approach transformed traditional assessment into a series of achievements, de-emphasizing grades and encouraging persistence and experimentation in a game-like, low-stakes environment [3].

Traditional Approach: In the control group, assignments maintained

fixed requirements, traditional grading language and in-person feedback. This structure provided a consistent, traditional approach aligning with typical programming course standards.

Table 1 outlines differences in assignment features, focusing on how wording varied between the experimental and control groups. Each feature row specifies both the experimental and control phrasing, with the "Method of Delivery" column clarifying whether the difference was a direct wording change within the assignment ("Assignment") or a broader (not explicitly documented by the researchers) verbal adjustment made during in-person discussions ("In-person"). Otherwise, both the AGG and control assignments were identical.

Table 1: Comparison of Experimental and Control Group Assignments

Feature	**Experimental Group**	**Control Group**	**Method of Delivery**
Assignment Structure	Core Tasks and Optional Tasks	Mandatory Functions and Optional Functions	Assignment
Extra Credit	Expert Programming Levels	Extra Credit Options	Assignment
Student Autonomy	Select optional tasks to reach 80 points	Mix-match basic and advanced functions	Assignment
Focus	Focus on mastery through expert levels	Completion via fixed mandatory/optional points	In-person
Feedback	Narrative-driven (levels, badges, adaptive hints)	Standard comments and grade-based scores	In-person

4 Data Analysis

Student feedback was thematically analyzed as positive, mixed/neutral, or negative. Table 3 shows the frequency of feedback types across groups. The qualitative survey conducted post-intervention addressed topics such as (Q1) student motivation, (Q2) engagement, (Q3) assessment impact, (Q4) academic challenge, (Q5) course expectations, (Q6) learning support, (Q7) practical preparedness, and (Q8) confidence in technical problem-solving, with eight optional questions. Responses were grouped into positive and non-positive feedback for comparison purposes.

Table 2: Demographic Summary of Experimental (AGG) and Control Groups

Category	Subcategory	AGG (n=92)	Control (n=102)
Race	White	40	40
	Asian	29	35
	Other	7	10
	Black	3	4
Ethnicity	Not Hispanic	68	73
	Hispanic	21	24
	NA	3	4
Socioeconomic	Middle Income	65	67
	Low Income	12	26
	High Income	7	5
	NA	7	4
Employment	Unemployed	52	41
	Part-Time	37	54
	Full-Time	3	5
	NA	0	2

Survey Questions

Q1 In what ways did you find yourself more or less motivated to participate in course activities?

Q2 Describe how the course design impacted your involvement in class discussions and assignments.

Q3 In what ways did the course's assessment methods (e.g., assignments, quizzes) impact your learning and performance?

Q4 Can you provide examples of how the course content challenged you academically?

Q5 How did the course design align with or differ from your expectations of a computer science course?

Q6 How did the course environment support or hinder your learning process?

Q7 How prepared do you feel to use the knowledge gained from this course in practical applications?

Q8 Can you describe how this course contributed to your confidence in tackling technical problems?

Table 3: Responses for Experimental and Control Groups

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8
Experimental Group (AGG)								
Positive Feedback	39	41	44	41	37	44	39	39
Neutral or Mixed Feedback	12	3	2	1	0	1	1	3
Negative Feedback	4	3	2	7	9	2	6	4
Control Group								
Positive Feedback	43	48	46	47	42	40	43	44
Neutral or Mixed Feedback	9	9	9	7	4	9	6	7
Negative Feedback	11	4	6	7	14	11	12	7

A Chi-squared test on positive and non-positive feedback distributions revealed a significant difference ($X = 17.472, df = 2, p < 0.001$), indicating that the AGG group provided a statistically higher proportion of positive feedback when observing all open-ended questions, which showed that the AGG group consistently displayed more favorable attitudes compared to the control group.

5 Conclusion

This study evaluated the impact of an AGG approach compared to traditional grading on student perceptions of engagement, performance, expectations, and applicability in a programming course. Students in the AGG group reported enhanced motivation and engagement, attributing this to the flexible, mastery-based structure that allowed creative exploration and incremental learning. Some students did, however, mention occasional ambiguity in instructions, suggesting a need for clearer guidance within the AGG format.

While traditional grading practices in the control group provided familiarity, some students reported frustration with rigid assessment, which they felt sometimes constrained their motivation. In contrast, the AGG group appreciated the focus on mastery and progression, which felt more aligned with professional goals and strengthened their confidence in problem-solving.

Both groups found assignments relevant to real-world applications, though the AGG structure better supported students in connecting course concepts to practical applications. Overall, the AGG approach shows promise for fostering an engaging, skill-oriented environment, though future studies should assess long-term impacts and demographic variations. Further research is needed to explore how AGG pedagogy may influence diverse student populations, as well as its potential to integrate with various course formats and disciplines.

Understanding these factors will be key to optimizing this approach for broader educational contexts.

References

[1] Jessica E. Broussard. "Playing Class: A Case Study of Ludic Pedagogy". In ProQuest Dissertations and Theses (2674873586). Ph.D. Dissertation. Louisiana State University and Agricultural & Mechanical College, 2011.

[2] Mihaly Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. First edition. New York, NY, USA: Harper & Row, 1990.

[3] Caitlin Holman. "Building a Better Game: A Theory of Gameful Learning & the Construction of Student Personas with Agency". In ProQuest Dissertations and Theses (2080292528). Ph.D. Dissertation. University of Michigan, 2018.

[4] Briana B. Morrison et al. "Evidence for Teaching Practices that Broaden Participation for Women in Computing". In: *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education*. ITiCSE-WGR '21. Virtual Event, Germany: Association for Computing Machinery, 2022, pp. 57–131. ISBN: 9781450392020. DOI: 10.1145/3502870.3506568.

[5] Benedikt Morschheuser et al. "How to Gamify? A Method For Designing Gamification". In: Jan. 2017. DOI: 10.24251/HICSS.2017.155.

[6] Anthony Robins. "Learning edge momentum: a new account of outcomes in CS1". In: *Computer Science Education* 20.1 (2010), pp. 37–71. DOI: 10.1080/08993401003612167.

[7] Kyle W. Scholz, Jolanta N. Komornicka, and Andrew Moore. "Gamifying History: Designing and Implementing a Game-Based Learning Course Design Framework". In: *Teaching & Learning Inquiry* 9.1 (2021). Publicly Available Content Database, pp. 99–116. URL: https://doi.org/10.20343/teachlearninqu.9.1.9.

[8] Susan Leigh Star and James R. Griesemer. "Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39". In: *Social studies of science* 19.3 (1989). ObjectType-Article-1, pp. 387–420. ISSN: 0306-3127. DOI: 10.1177/030631289019003001.

[9] George Zhao. "Using a Gamified Points-Based Grading System in Technology Courses for Pre-Service Teachers". In ProQuest Dissertations and Theses (2478622680). Ph.D. Dissertation. University of Houston, 2019.

Adaptable Metrics to Inform Introductory CS*

Yuan Garcia, Jenny Ngo, Florence Rui Lin, Zachary Dodds
Computer Science Department
Harvey Mudd College
Claremont, CA 91711
{ygarcia,jngo,fllin,dodds}@g.hmc.edu

Abstract

Metrics have long been used to assess and guide successful software projects. Traditionally these metrics have measured software's *professional* rather than its *educational* suitability. This work proposes six adaptable, reproducible *pedagogical* metrics. With these metrics, we track an Introductory CS course's capstone projects, 2018-2024. The results suggest both year-over-year evolution and a more sudden, LLM-correlated impact on students' relationship with their early computing work. We have begun adapting our curriculum to these signals, and we foresee future refinements and broader applications to metrics-based reproducible curricular assessment.

1 Introduction: Comp1's "Springboard" Projects

Creative final projects are, rightfully, an often-used capstone for introductory computing courses. Usually, they are the largest and most open-ended software artifacts built by introductory computing students. As such, they invite personal investment smaller assignments can't always match. At its best, that personal investment can be a strong foundation for confidence and comfort with future computing, whether in a CS program or elsewhere.

Our Computing 1 (Comp1) course has a long tradition of these open-ended capstone projects. Students choose from five options. Nominally from scratch, each is partly scaffolded by its prompt:

1. a 3d physical simulation leveraging an external library
2. a terminal-based, turn-taking game with student-written AI
3. a text-analysis engine using both student-chosen and provided features
4. cellular automata of various rulesets (given and invented)
5. a genetic algorithm that itself evolves programs

Although these projects' popularity varies, the projects' "game balance" across effort, creativity, and complexity is equitable. As a result, all of them have been launching students into their computational future for many years.

2 Motivation: Metrics across Time and "AI Space"

This work asks the question, "What can introductory computing projects say about the student-computing relationship – especially given how rapidly it's evolving?" In the past three years - especially this past year - LLMs have changed the trajectory of computing skill-building. LLMs aside, *all* CS instructors track over time their choice of content, pedagogy, and philosophy. Updating and adapting decades of thought vis-a-vis traditional software metrics, this work proposes – and demonstrates – the insights available using *pedagogical* metrics, as instantiated via six years of Comp1 capstone projects.

3 Background: Adapting traditional software metrics

Although CS1 as a course takes on very different forms [8], it's not uncommon for CS1 courses to be capped by student-shaped final projects. Ideally, a final project affords students the chance to integrate skill-and-knowledge across most-or-all the topics they have practiced through their introductory course. It can also provide open-ended opportunity for creativity, individuality, and pride-of-creation in a software artifact.

Here, we seek reproducible syntheses, i.e., *metrics* that allow year-over-year comparisons of students' final introductory projects. These data help us, as instructors, to help our students, as computational authors, more deeply appreciate their work's clarity, its capabilities, and the expressive individuality imbued in it.

This paper thus layers a *pedagogical* purpose atop the longstanding history of software metrics. Software-quality concerns motivated Halsted [6] and McCabe [12], to design influential early metrics. They did not hold up equally well: Halsted's metrics have attracted widespread criticism [9, 1, 16], while McCabe's *Cyclomatic Complexity* is still widely used, a pattern this work happily continues! Both appear in software's first *maintainability indices*, which combined individual source-code features by fitting their coefficients to the results of a small number of projects [14, 4].

Our work draws inspiration from *next-generation* maintainability metrics, specifically [7]. Rather than averaging or projecting onto a 1D space, their metrics preserve the component software-maintainability features. They demonstrate convincingly that the resulting *maintainability space* is far more useful than a single summative value. Contemporary taxonomies summarize these within decades of past practice [3]. Yet they aspire only to inform professional-grade software quality and maintainability. They do *not* address the needs of CS education or, more specifically, Introductory CS.

Within CS Education, metrics have often been used for comparing submissions' authorship as "plagiarism prevention tool" [10]. This is not our goal. Instead, this work is closer to work by Santos et al., who applied similar techniques to identify programming learning-styles [5]. Yet, for us, *pedagogical responsiveness* is more important than learning style. Thus, we are closest to the work of Price et al. [15] and Nguyen et al. [13], who also incorporated metrics from capstone projects in a single *introductory data science* course.

Our work expands the scale and scope of these prior foundations. On one hand, we use a different set of metrics, as detailed in the next section. We also offer a deeper longitudinal look, spanning across six years and over a thousand students. That timespan allows us to begin to contrast how students and their artifacts have changed in the era of LLMs.

4 Mixing old with new: *Pedagogical* metrics and trends

Here we describe the six metrics we use to track students' CS1 project trends. We emphasize that these metrics are not used for grading projects. Grades are determined via the projects' rubrics, which include the behavioral capabilities of the software and its creativity within the framework of the project, all of which depend on the judgment of a human reader. We have a number of course alums who help the instructors with grading.

Rather, these six metrics allow us a *reproducible* look into the evolution over time of the relationship of our introductory CS cohorts and their final projects. Figure 2 summarizes with histograms the distributions of each of the six metrics over the past seven years. Figure 1 shows the p-values of the

Kolmogorov-Smirnov test, with highlighting relative to a (nominal) threshold of $p = 0.01$.

Kolmogorov-Smirnov tests	WEEKS	CYCCOMP	MAXDEPTH	NUMFUN	COMMENT	LOC
2023 vs 2018	0.0180	0.0000	0.2870	0.0000	0.0030	0.0048
2024 vs 2018	0.0016	0.0001	0.2029	0.0000	0.0003	0.0008
2024 vs 2023	0.0297	0.2047	0.6270	0.0002	0.1813	0.2711

Figure 1: The p-values computed by the K-S test for each of the six metrics comparing three timespans: '23 vs '18 (evolutionary change), '23 vs '24 (LLM advent), and '24 vs '18 (both forces).

4.1 Complexity

As we focus on students and the work they create, the traditional *complexity* metric is beneficial: we use McCabe's Cyclomatic Complexity [12], which counts the number of distinct code paths in a program. As it does for larger-scale software systems, this complexity metric conveys the structural intricacy of students' code and the logic behind their solutions. Higher complexity is *not* an indication of more sophistication. Rather, it is often a signal of ad-hoc thinking, i.e., code that has evolved in real-time. This is inevitable in introductory CS projects, but simpler is, by and large, better. In contrast, we represent structural complexity by measuring the highest level of function nesting present. This metric indicates comfort with function calls and function composition, a key goal of our introductory experience, as well as many others! Our data shows that, with statistical significance, complexity has decreased over time. However, the introduction of Large Language Models has not affected complexity. This is encouraging because this indicates modularity in project code, suggesting increasing separation of ideas and positive problem decomposition.

4.2 Volume

One of the key metrics in determining students' conceptual grasp is the volume of their final project. Though volume can be measured in many ways, we are using the straightforward *Lines of Code* (LOC) along with *Number of Functions* (Num Funcs). In our measurements, we include comments in this metric, because our goal is to get a sense of the sum-total of *student-managed material* in a submitted file. This ensures that our work overlaps with metrics' traditions across all software [2, 7, 11] as well as pedagogical software-creation tasks in CSEd[10]. Comparing both evolutionary change and LLM advent for *LOC* and *Num Funcs*, we see a significant decrease in both measurements.

Both volume metrics from 2018 and 2023 showed a significant decrease overall. Isolating changes due to LLMs (2023-24), only *Number of Functions* experienced significant change. We speculate the time-varying cultural factors and LLM-influence contribute to reduced Volume and reduced code duplication.

4.3 Documentation Quantity

In Introductory CS, comments often reflect the students' ability to document and explain their thought process, making their work accessible to others[7]. In our metrics, we include lines of comment alongside *Lines of Code* to calculate *Percentage of Comments* to assess students' understanding of desired concepts. This way, we can capture the breadth, implementation, and documentation of students' work. We do see a significant decrease in comment quantity over time, but not a significant change after LLMs were introduced. This has prompted updates to our own pedagogy in subsequent offerings.

4.4 Ambition

Capturing the ambition behind students' code is essential for understanding their learning journey and growth. We assess the integration of course topics in projects as a measure of students' deep understanding and mastery of core concepts. To represent this idea, we use (1) the number of learned concepts encapsulated and (2) the depth of function dependency in students' projects. Together, these metrics attempt to provide a holistic view of the students' ambition, showcasing both their problem-solving depth and sustained effort. Over the years, we see a significant change in *Weeks Used* of concepts, which can be attributed to both evolutionary change and LLM usage. However, there is no significant change in *Max Depth*. This suggests that over time and across LLM use, student work continues to illustrate problem-decomposition skills.

5 Preliminary Verdict

Assessments like these best serve students – and instructors – when they reflect and support the relationships at the heart of our work: their interpersonal foundations, as well as the layer of computing confidence and capability that grows from those foundations.

For tracking the "student-computing relationship" in our introductory computing course, this paper's metrics have succeeded in a number of ways:

- they reproducibly ground our informal observations through the raw material of student-submitted projects, which are the most individualized work in our course

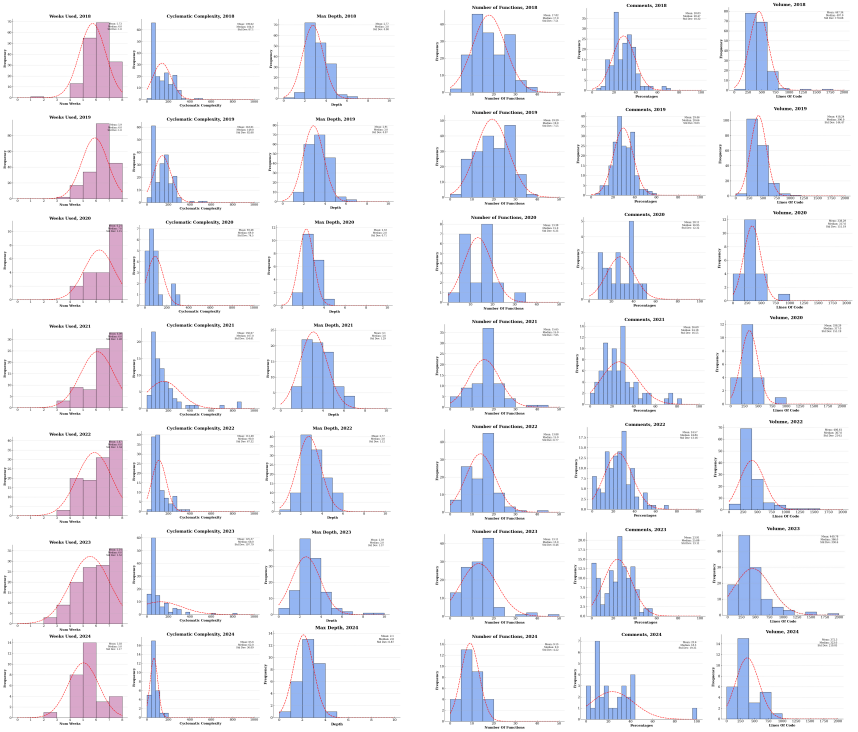

Figure 2: Trends over seven years of Introductory CS final projects measured by six metrics: Weeks Used *(far left)*, an estimate of the completeness of course-integration; Cyclomatic Complexity *(mid left)*, code paths counted per file; Max Depth *(center left)*, the longest nested function call chain; No. Functions *(center right)*, a count of defined functions; Comments *(mid right)*, as a percentage of code lines; and Volume *(far right)*, tracking total lines of code, LoC. Section 4 describes trends observed and addressed, i.e., decreasing complexity, functions, and comments; varying coverage; and constant call-depth and volume, i.e., lines of code.

- they track changes in the student-computing relationship, across timescales short (pre-and-post LLMs) and medium-term (since before Covid at least)
- the *weeks used* metric has spurred new pedagogical approaches in our introductory CS curriculum: in 2024, we began piloting *student-defined deliverables* to better align students' semester-wide integration of topics.

(That effort warrants its own, separate treatment.) Here, *all* of these metrics have helped with our careful, deliberate inclusion of LLM-use in our introductory courses.

Even so, these measures are neither institution- nor curriculum-specific. As a result, this work can inform introductory curricula across a wide variety of institutions and philosophies. For example, quantifying the importance of clear, well-structured, and appropriately complex, i.e., appropriately *expressive*, code helps instructors and students alike. Measures of student-computing engagement open the door to a positive feedback loop as new technologies quickly become everyday expectations.

6 Perspective

With its single snapshot of a single institution's past six years of introductory computing, this work invites further exploration along two paths. The first is a comparison with other introductory computing courses. Each course emphasizes its own set of priorities – as it should – and this paper's metrics are an opportunity to reproducibly cross-compare student artifacts, whether within courses or across them, to support new contexts, collaborations, and experiments. The second opportunity is to further expand the pedagogical measures of computing work – for example, by creating LLM-based metrics that may (or may not) augment our insights about the effectiveness and enjoyment of computing curriculum – or others that correlate with creativity, sophistication, and ambition.

It is an exciting time to invest in the evolving relationships between college students and computing. We look forward to sharing the journey!

7 Acknowledgments

The authors gratefully acknowledge support from Google, Harvey Mudd College, and NSF DUE 2142780.

References

[1] Daniel Afriyie and Yvan Labiche. "Predictors of Software Metric Correlation: A Non-parametric Analysis". In: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. Dec. 2021, pp. 524–533.

[2] Kalev Alpernas, Yotam M. Y. Feldman, and Hila Peleg. “The Wonderful Wizard of LoC: Paying Attention to the Man behind the Curtain of Lines-of-Code Metrics”. In: *Proceedings of the 2020 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software* (Nov. 2020), pp. 146–156.

[3] Fatima Nur Colakoglu, Ali Yazici, and Alok Mishra. “Software Product Quality Metrics: A Systematic Mapping Study”. In: *IEEE Access* 9 (2021), pp. 44647–44670.

[4] Don Coleman et al. “Using Metrics to Evaluate Software System Maintainability”. In: *Computer* 27.8 (Aug. 1994), pp. 44–49.

[5] Francisco Alan de Oliveira Santos and Luis Carlos Costa Fonseca. “Collection and Analysis of Source Code Metrics for Composition of Programming Learning Profiles”. In: *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*. Vol. 2161-377X. July 2019, pp. 173–175.

[6] Maurice H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. USA: Elsevier Science Inc., Apr. 1977.

[7] Ilja Heitlager, Tobias Kuipers, and Joost Visser. “A Practical Model for Measuring Maintainability”. In: *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*. Lisbon, Portugal: IEEE, Sept. 2007, pp. 30–39.

[8] Matthew Hertz. “What do "CS1" and "CS2" mean? investigating differences in the early courses”. In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. SIGCSE ’10. Milwaukee, Wisconsin, USA: Association for Computing Machinery, 2010, pp. 199–203.

[9] Capers Jones. “Software Metrics: Good, Bad and Missing”. In: *Computer* 27.9 (Sept. 1994), pp. 98–100.

[10] Ronald J. Leach. “Using Metrics to Evaluate Student Programs”. In: *SIGCSE Bull.* 27.2 (June 1995), pp. 41–43.

[11] Md Abdullah Al Mamun, Christian Berger, and Jörgen Hansson. “Correlations of Software Code Metrics: An Empirical Study”. In: *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*. IWSM Mensura ’17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 255–266.

[12] Thomas J. McCabe. “A Complexity Measure”. In: *IEEE Transactions on Software Engineering* SE-2.4 (Dec. 1976), pp. 308–320.

[13] Huy Nguyen et al. “Exploring Metrics for the Analysis of Code Submissions in an Introductory Data Science Course”. In: *LAK21: 11th International Learning Analytics and Knowledge Conference*. LAK21. Irvine, CA, USA: Association for Computing Machinery, 2021, pp. 632–638.

[14] Paul Oman and Jack Hagemeister. “Construction and Testing of Polynomials Predicting Software Maintainability”. In: *Journal of Systems and Software*. Oregon Workshop on Software Metrics 24.3 (Mar. 1994), pp. 251–266.

[15] Thomas Price et al. “Using Data to Inform Computing Education Research and Practice”. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE ’20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 175–176.

[16] Rafa Al-Qutaish and Alain Abran. “An Analysis of the Design and Definitions of Halstead’s Metrics”. In: Sept. 2005.

Teaching the TAs: Course-Based Undergraduate TA Training*

Austin Zang, Elshiekh Ahmed, Eleanor Birrell, and Tzu-Yi Chen
Computer Science Department
Pomona College
Claremont, CA 91711
{ayza2020,eoaa2021}@mymail.pomona.edu
{eleanor.birrell,tzuyi.chen}@pomona.edu

Abstract

As research universities and liberal arts colleges hire more undergraduate teaching assistants (TAs) to support growing computer science enrollments, effective training for those TAs becomes increasingly important. In this paper we describe the design and evaluation of a peer-led, discussion-based TA training seminar course. Our results suggest students in the course gained familiarity with terms and techniques relevant to effective and inclusive teaching, and additionally acquired an expanded sense of community. We discuss aspects of the class that worked well and those that could use improvement. We conclude with recommendations for others considering offering a similar course.

1 Introduction

As demand for computer science (CS) classes increases [21], both large research universities and small liberal arts colleges (SLACs) are hiring rising numbers of undergraduate Teaching Assistants (TAs). While many strategies for TA training exist, effects are often reported anecdotally rather than via formal

studies [19]. Additionally the impact of training on TA interactions with underrepresented student groups is not well understood [19].

At Pomona College, a small liberal arts institution, we offered a student-led seminar course in Spring 2024 [25]. The class was centered around small-group discussions of readings in peer mentoring and diversity, equity, and inclusion (DEI) in CS. We then asked whether this kind of lightweight offering could improve TA effectiveness and engagement. We used pre-/post-surveys and semi-structured interviews to investigate the following research questions:

RQ1: Are students more aware of effective teaching practices after taking this course?

RQ2: Are students more aware of DEI issues after taking this course?

RQ3: Do students feel more engaged with the CS department community after taking this course?

2 Related Works

Mirza et al. [19] conducted a systematic literature review of research on undergraduate TA training and observed that TA training programs vary in frequency (weekly [1, 8] vs. annual/semi-annual [2, 3, 4]), content (including learning styles [5, 35], teaching techniques [1, 2, 3, 9, 24, 31, 32, 35], communication skills [1, 2, 3, 7, 8, 23, 24, 32, 35], professionalism [1, 2, 24, 4, 35], grading [2, 29, 28, 31, 32], and other topics [1, 2, 5, 29, 28]), and pedagogical structure (including workshops [29, 31, 32], self-reflection [8, 28], role-playing [1, 2, 29, 32, 4], and practice lectures [8, 29]).

Given the range of contexts in which TAs work, there may be no universal best practice for TA training [30]. However, experience reports observe benefits across a range of institutions and degree programs: from large institutions to SLACs, and from psychology to engineering [26, 2, 13, 20]. That said, despite the importance of addressing gaps in CS support structures for underrepresented students, there is relatively little work on how TA training affects DEI awareness [34]. DEI research more broadly has explored decolonizing university curricula [6], inclusive general education TAs [33], and DEI for graduate TAs [36]. Recent works explore inclusivity card games in a TA workshop [15] and a lecture-based course centering positive classroom climates [14].

3 Course Design

We designed a peer-led, discussion-based seminar course for TA training [25]. Two undergraduates with significant TA experience served as primary course facilitators; two supervising faculty alternated attendance.

The course met once a week for 75 minutes and revolved around peer discussion of topics related to assigned readings. These readings were predominantly research articles but also included a few articles from the popular press. The syllabus was divided into three modules: peer mentoring practices, DEI in STEM, and applications. Each 75-minute meeting had 3 parts: community-building icebreaker (10 minutes), small group discussions (50 minutes), and class debrief (15 minutes). The facilitators and supervisors participated in discussion as equal peers.

Students were expected to complete weekly readings and to submit short written responses relating the reading to their lived experiences. Students also participated in a live mentoring workshop where they discussed mock TA-mentee scenarios. And, finally, students were expected to execute and present a final project that applied course topics towards improving systems and resources in our department. The course was graded on a pass/fail basis.

4 Methodology

Students completed a pre-survey in the semester's first weeks and a matching post-survey after the last week; they were also offered the opportunity to be interviewed at the end of the semester. All protocols received IRB approval.

To assess awareness of teaching practices (RQ1) and DEI (RQ2) terms, we asked students to define and to give a confidence rating of their definitions for 6 terms: *self-explanation*, *goal orientation*, *growth mindset* (RQ1) [16, 12, 17]; *microaggression*, *fallacious archetype*, *defensive climate* (RQ2) [11, 27, 10]. We compared their definitions to that from course readings (Appendix A) and coded each response as *No Answer*, *Uncertain*, *Incorrect*, *Partially Correct*, or *Correct*. Improvement was defined as having a response that changed either from one of the first 3 categories in the pre-survey to one of the latter 2 categories in the post-survey, or from *Partially Correct* to *Correct*.

Next, we asked for both Likert confidence and short answers assessing TA's existing approaches: *Describe what you already do to try and be an effective mentor, describe what you already do to try and create inclusive environments while mentoring, describe how you have incorporated some of these concepts (as you defined) into your own mentoring.* We used thematic analysis via inductive coding [22] to analyze 44 qualitative responses to these prompts across both surveys. Two authors independently open coded all student responses, then grouped them on a virtual affinity diagram for thematic analysis through axial coding [18]. Emergent themes were identified through collaborative review: the two authors then independently re-tagged the initial 44 responses deductively against these themes (Cohen's Kappa 0.26 to 1.0). Disagreements were discussed until consensus was acheived.

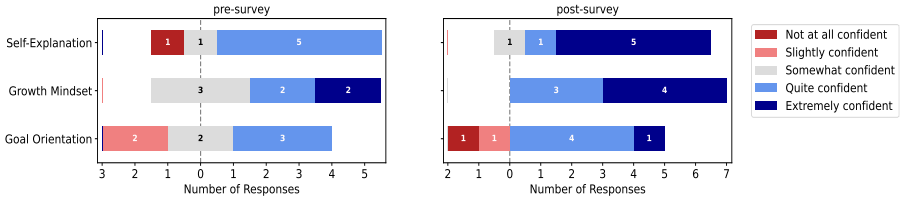

Figure 1: Student pre- and post-survey confidence defining RQ1 terms

Optional 45-minute semi-structured interviews were conducted in person by the course facilitators at the end of the semester. The flexible script examined three topics: **Motivation**, a student's background and reasons for enrolling; **Growth**, a student's self-identified changes over the semester; and **Feedback**, a student's suggestions and opinions regarding the class. Interviewers took notes and transcribed audio via Otter.ai. Each facilitator open coded their interview transcripts by topic. Codes were placed on a virtual affinity diagram for thematic analysis through axial coding, resulting in a final set of emergent themes for each topic, shown in Table 2.

5 Results

Of the 9 students enrolled, 8 consented to data inclusion. They self-identified as follows: 6 male and 2 female; 3 Black or African American, 3 Asian, 1 Hispanic, and 1 White; 1 international student. One student chose not to be interviewed and one did not complete the post-survey.

RQ1: Awareness of Teaching Practices 7 students completed both surveys, and teaching definitions improved as follows: 2 on *self-explanation*, 4 on *growth mindset*, and 2 on *goal orientation*. Additionally their confidence in their definitions increased, as shown in Figure 1. Figure 2 shows student confidence in their own teaching effectiveness also increased. In Table 1, survey short answer mentions of specific teaching techniques increased from 35 to 41. Table 2 shows that in their exit interviews, 5 students said the class improved their ability to evaluate mentoring strategies, and 3 said the class improved their use of self-explanation specifically.

RQ2: Awareness of DEI Of the 7 students, 4 gave an improved definition of *microaggression*, 3 of *fallacious archetype*, and 5 of *defensive climate*. Figure 3 shows increased confidence in their definitions. And Table 2 shows that 5 students mentioned improved awareness of inclusive strategies for TAs,

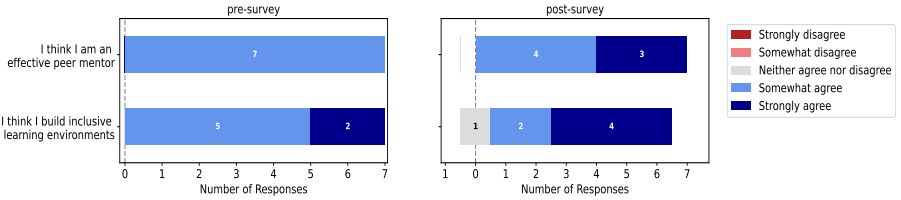

Figure 2: Student pre- and post-survey confidence in own teaching effectiveness

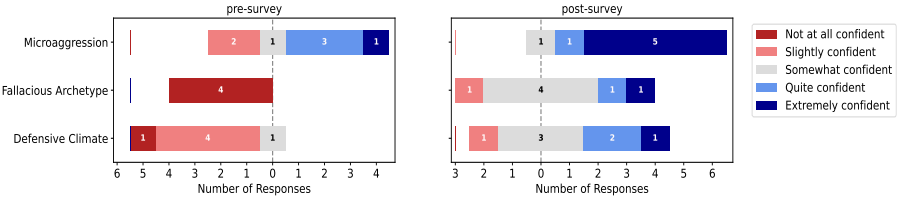

Figure 3: Student pre- and post-survey confidence defining RQ2 terms

with one saying: "I've also been more mindful of general practices that will not alienate certain students . . . of trying to be more mindful of different backgrounds that different students have come from. If there's anything I'm doing unintentionally that might not make [students] feel comfortable . . . I feel like I'm just more cognitive of things like that."

RQ3: Community Building As shown Table 2, many students discussed in their exit interviews the community building aspect of the course: 6 mentioned growth via peer discussion, 5 talked about classroom diversity, and 3 appreciated the safe space for exchanging ideas. One participant reflected: "Nine consistent students plus two professors. We got to know each other, where everyone was coming from, really well. In different discussion groups . . . it was an honest and vulnerable space where I could be real about the struggles I've had TAing, and then also learn from struggles other people have had."

6 Discussion

TA Awareness Overall, students gained awareness of teaching techniques (RQ1). While they began with lower confidence and knowledge of DEI terms, the number of improved definitions suggests the course had beneficial impacts here as well (RQ2). Peer-to-peer discussions in which students could hear directly from classmates were extremely valuable in part because the ideas

Survey Theme	Pre	Post
Self-explanation	3	5
Individual Attention	4	6
Active Listening	4	4
Growth Mindset	4	4
Fairness	3	3
Teaches Concepts	4	4
Embraces Mistakes	4	5
Microaggressions	3	4
Takes Initiative	3	5
Mentor Preparation	3	1

Table 1: Survey Short Answer Codes

Growth Themes	# students
DEI awareness	5
Eval. mentor strategies	5
Use of self-explanation	3
Feedback Themes	
Valued discussions	6
Valued diverse class	5
Valued safe space	3
Lacked hands-on training	3
Valued readings	2
Valued engaging professors	2
Valued engaging non-TAs	2
Wished for more students	2
Often skimmed readings	2

Table 2: Exit Interview Codes

exchanged, mistakes identified, and struggles shared were all directly relevant to student TA experiences in our department.

Community Impact Students deeply valued having a weekly space for authentic discussion with other students and with faculty. We additionally noticed students from the course socializing with each other outside of class, encouraging others to incorporate effective teaching strategies, and speaking proudly of their course participation. The final projects showcased a strong desire to expand community and give back to our department: ideas included implementing structural collection of alumni course notes, proposing a partial-credit student incubator for industry software engineering preparation, starting a weekly email highlighting photos and biographies of CS majors, and more.

Recommendations Due to both the class's small size and tendency to draw interest from students already engaged with the department, reported experiences may be biased. We recommend expanding enrollment and particularly of first-time and prospective TAs. The larger class size and increased background diversity could enrich discussions and broaden course impact, although facilitators must carefully maintain spaces conducive to vulnerable exchanges. We also recommend reducing weekly reading load by assigning selected excerpts, as students reported limited engagement and retention with full-length academic papers. Strict enforcement of attendance policy is key, given the focus on live discussion. Lastly we recommend adding additional experiential learning opportunities, such as role-playing workshops or simulated mentoring sessions, in order to help students explore how to apply course material in practice.

References

[1] Christine Alvarado, Mia Minnes, and Leo Porter. "Micro-classes: A structure for improving student experience in large classes". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 2017, pp. 21–26.

[2] Rebecca Brent et al. "Preparing undergraduates to teach computer applications to engineering freshmen". In: *2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports*. IEEE. 2007, F1J–19.

[3] Shearon Brown and Xiaohong Yuan. "Experiences with retaining computer science students". In: *Journal of Computing Sciences in Colleges* 29.5 (2014), pp. 34–41.

[4] Andries van Dam. "Reflections on an introductory CS course, CS15, at Brown University". In: *ACM Inroads* 9.4 (2018).

[5] Holger Danielsiek et al. "Undergraduate teaching assistants in computer science: Teaching-related beliefs, tasks, and competences". In: *2017 IEEE Global Engineering Education Conference (EDUCON)*. IEEE. 2017, pp. 718–725.

[6] George Dei. "Decolonizing the university: The challenges and possibilities of inclusive education". In: *Socialist Studies/Études Socialistes* 11.1 (2016), pp. 23–23.

[7] Paul E Dickson, Toby Dragon, and Adam Lee. "Using undergraduate teaching assistants in small classes". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 2017, pp. 165–170.

[8] Francisco J Estrada and Anya Tafliovich. "Bridging the gap between desired and actual qualifications of teaching assistants: An experience report". In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. 2017, pp. 134–139.

[9] Meg Fryling et al. "Catch'em Early: Internship and Assistantship CS Mentoring Programs for Underclassmen". In: *Proceedings of the ACM Technical Symposium on Computer Science Education*. 2018.

[10] Kathy Garvin-Doxas and Lecia J Barker. "Communication in computer science classrooms: Understanding defensive climates as a means of creating supportive behaviors". In: *Journal on Educational Resources in Computing (JERIC)* 4.1 (2004), 2–es.

[11] Colin Harrison and Kimberly D Tanner. "Language matters: Considering microaggressions in science". In: *CBE—Life Sciences Education* 17.1 (2018), fe4.

[12] Meredith A Henry et al. "FAIL is not a four-letter word: A theoretical framework for exploring undergraduate students' approaches to academic challenge and responses to failure in STEM learning environments". In: *CBE—Life Sciences Education* 18.1 (2019), ar11.

[13] Thomas P Hogan et al. "Working with and training undergraduates as teaching assistants". In: *Teaching of Psychology* 34.3 (2007), pp. 187–190.

[14] Victor Huang and Armando Fox. "A Climate-First Approach to Training Student Teaching Assistants". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 2023, pp. 423–429.

[15] Audra Lane et al. "Motivating literature and evaluation of the teaching practices game: Preparing teaching assistants to promote inclusivity". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 2021, pp. 816–822.

[16] James M. Lang. *Small Teaching: Everyday Lessons from the Science of Learning*. Jossey-Bass Inc Pub, 2016.

[17] Lisa B Limeri et al. "Growing a growth mindset: Characterizing how and why undergraduate students' mindsets change". In: *International Journal of STEM Education* 7 (2020), pp. 1–19.

[18] Andrés Lucero. "Using affinity diagrams to evaluate interactive prototypes". In: *Human-Computer Interaction–INTERACT 2015: 15th IFIP TC 13 International Conference, Bamberg, Germany, September 14-18, 2015, Proceedings, Part II 15*. Springer. 2015, pp. 231–248.

[19] Diba Mirza et al. "Undergraduate teaching assistants in computer science: a systematic literature review". In: *Proceedings of the 2019 ACM Conference on International Computing Education Research*. 2019, pp. 31–40.

[20] Lakshmy Mohandas et al. "Effectiveness of Undergraduate Teaching Assistants in a First-Year Design Course". In: *2020 ASEE Virtual Annual Conference Content Access*. 2020.

[21] National Center for Education Statistics. *Bachelor's degrees conferred by postsecondary institutions, by field of study: Selected academic years, 1970-71 through 2020-21*. 2022. URL: `https://nces.ed.gov/programs/digest/d22/tables/dt22_322.10.asp`.

[22] Kimberly A Neuendorf. "Content analysis and thematic analysis". In: *Advanced research methods for applied psychology*. Routledge, 2018, pp. 211–223.

[23] Elizabeth Patitsas. "A case study of the development of CS teaching assistants and their experiences with team teaching". In: *Proceedings of the 13th Koli Calling International Conference on computing education research*. 2013, pp. 115–124.

[24] John Paxton. "Undergraduate consultation: opportunities and challenges". In: *Journal of Computing Science in Colleges* 21.1 (2005), pp. 231–238.

[25] Pomona College. *ID009: Peer Mentoring in STEM and Computer Science*. URL: `https://cs.pomona.edu/classes/id009/2024sp/` (visited on 02/04/2025).

[26] Heather Pon-Barry, Becky Wai-Ling Packard, and Audrey St. John. "Expanding capacity and promoting inclusion in introductory computer science: a focus on near-peer mentor preparation and code review". In: *Computer Science Education* 27.1 (2017).

[27] Vahab Pournaghshband and Paola Medel. “Promoting diversity-inclusive computer science pedagogies: A multidimensional perspective”. In: *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*. 2020.

[28] Stuart Reges. “Using undergraduates as teaching assistants at a state university”. In: *Proceedings of the SIGCSE Technical Symposium*. 2003.

[29] Stuart Reges, John McGrory, and Jeff Smith. “The effective use of undergraduates to staff large introductory CS courses”. In: *Proceedings of the SIGCSE Technical Symposium*. 1988.

[30] Kristin A. Ritchey and Shelby Smith. “Developing a Training Course for Undergraduate Teaching Assistants”. In: *College Teaching* 67.1 (2019).

[31] Eric Roberts, John Lilly, and Bryan Rollins. “Using undergraduates as teaching assistants in introductory programming courses: An update on the Stanford experience”. In: *Proceedings of the SIGCSE Technical Symposium*. 1995.

[32] Guido Rößling and Jacqueline Gölz. “Preparing first-time CS student teaching assistants”. In: *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*. 2018.

[33] Umesh Sharma and Spencer J Salend. “Teaching assistants in inclusive classrooms: A systematic analysis of the international research”. In: *Australian Journal of Teacher Education (Online)* 41.8 (2016), pp. 118–134.

[34] Roli Varma. “Making Computer Science Minority- Friendly.” In: *Communications of the ACM* 49 (Feb. 2006).

[35] Dee AB Weikle. “More insights on a peer tutoring model for small schools with limited funding and resources”. In: *Journal of Computing Sciences in Colleges* 31.3 (2016).

[36] Deborah S Willis and Laura N Schram. “Graduate student diversity, equity and inclusion professional development”. In: *Studies in Graduate and Postdoctoral Education* 14.1 (2022), pp. 63–82.

A Correct Definitions of Terms from Course Readings

Term	Definition
Self-Explanation	learners benefit from explaining out loud (to themselves or others) what they are doing during the completion of a learning task [16].
Growth Mindset	students who believe that intelligence is a changeable trait that they can improve with effort and guidance are described as holding a "growth mindset" [17].
Goal Orientation	the goals and aims students tend to hold when approaching a new task. These goals fall into two main orientations: 1) mastery or 2) performance [12].
Microaggression	brief, sometimes subtle, everyday exchanges that either consciously or unconsciously disparage others based on their personal characteristics or perceived group membership [11].
Fallacious Archetype	students who match the hegemonic profiles of 'successful' CS students tend to possess certain values for each component. Those are: male gender, white race, middle or higher socioeconomic [27].
Defensive Climate	describing classroom communication climates, such as whether students feel comfortable asking questions, and what sorts of comments, discussions, and knowledge are valued... exhibits traits like: evaluative, controlling, strategic, neutral, superior, certain [10].

Code Replacement Detection as a Cheating Detection Approach in Programming Classes*

Ashley Pang, Lizbeth Areizaga,
Benjamin Denzler, Mariam Salloum, and Frank Vahid
Computer Science and Engineering
University of California, Riverside
Riverside, CA
{apang024, larei002, bdenz001, msall001, vahid}@cs.ucr.edu

Abstract

Similarity checking has long been the main approach for detecting cheating in programming classes. While still a key approach, similarity checking has increasing limitations, due to more ways for students to copy code from online solutions, low-cost contractors, and artificial intelligence code writing tools - - in such cases, a student may copy a program that is not similar to a program from any classmate. However, the rise in use of program auto-graders, version control, and other tools that capture a student's program history provides new cheating detection approach opportunities. One approach detects when a student's program history includes a code replacement - - an instance where a student's code at one time is followed by new code that is clearly an entirely different program. We manually examined program histories for 5 labs in our CS1 class, for 50 random students per lab, and found code replacement prevalence of 12%, with half not turning up in the similarity checker. Detecting code replacement, and from-the-start solution copying, may become important complements to similarity checking, to catch and preferably prevent cheating in programming classes.

1 Introduction

Cheating in computer science (CS) programming courses is a long-standing problem. The problem has been exacerbated in recent years by growth of online websites and apps, via easy availability of solutions at code-sharing sites like GitHub [7, 12], low-cost "help" websites like Chegg [3] with "tutors" who write solutions for students [1], real-time anonymous student communication apps like Discord [5], AI-driven code writing tools like ChatGPT [4] or GitHub's Copilot [14], and more.

The most common way to detect cheating on programming assignments is via similarity checking [18, 10, 16, 23]. Similarity checking works by detecting that a student has submitted a program similar to another student in the class and/or to a set of known solutions such as those from a previous term. Similarity checkers ignore certain differences, such as variable names, minor reordering of statements or expressions, whitespace, comments, etc.

Today, a new technique is enabled in cases where instructors have access not just to a student's one final submission, but also to a history of versions of a student's program as the student developed their program. Such a version history may be available if, for instance, a class is using an auto-grader, wherein a student might submit code to the auto-grader to receive a score, and then resubmit multiple times to try to gain a higher score. Thousands of CS1 courses today use program auto-graders [8]. Another instance is when students must develop code within a particular cloud environment, and that environment records the history of code [20, 6, 9]. Another instance is when an instructor requires students to regularly commit their code to a private code repository like GitHub as in [19].

With such program history, one possible technique for detecting potential cheating is to detect that a student replaced their solution with an entirely different solution [20]. For example, a student might first try to write a program but, upon repeatedly getting a bad score from an auto-grader, give up and resort to copying, often late at night near a deadline in an act of desperation [13]. Looking at the history of program versions, an instructor can detect that a program has been replaced by another, typically by noticing a different algorithm, different variable names, a change in coding style, etc.

Other researchers in recent years have also begun to mine student behavior on cloud-based development tools to detect cheating [11]. In our own prior work, we proposed a comprehensive means of detecting program cheating, and detecting large code replacements is one facet of our cheating concern metrics [21].

In this work, we sought to determine the prevalence of code replacement instances on programming assignments in our CS1 class at a large state university. Also, we looked at how often the resulting solutions would have not

been noticed by a similarity checker. Thus, we performed a manual and auto-detected analysis on 5 programming assignments. Our goal ultimately is not to catch students cheating, but rather to prevent such cheating. By developing strong multi-faceted automated detection tools and making students aware of such tools, the goal is to prevent cheating by reducing perceived opportunity, one key factor according to Albuwi [2]. Additionally, this is a continuation of our prior work in an effort to prevent students from cheating by making detection/punishment efforts clear[15, 22].

2 Our CS1 course

Our CS1 course serves ∼1500 students per year, about half computing majors, and the other half science and engineering majors. The class is offered every quarter for 10 weeks via 3-5 ∼100 student sections, plus summer sections. The class is taught by experienced instructors, has strong course evaluations, good grades, and yields solid student performance in CS2 and CS3. The class uses pedagogical approaches known to aid student success: flipped lectures, active learning, scaffolding, many-small-programs, auto-grading, peer instruction, allowed collaboration, growth mindset, help normalization and resources (learning assistants, office hours, real-time discussion forum).

Our class uses the zyBooks learning system [24]. Each week, students read and answer ∼100 learning questions (Participation Activities or PAs), complete ∼20 homework code reading and writing problems (Challenge Activities or CAs), and code 5-10 programming assignments (Lab Activities or LAs), all in the zyBook. We expect about 7-9 hours per week of work outside lecture period, for students with no prior experience. Students are required to do all programming in the zyBook (no external tools), to reduce cheating, reward effort, and enable analysis. All zyBook activities are auto-graded, with immediate feedback, partial credit, and resubmissions. Instructors can download reports of activity completion, and logs of all LA program runs.

Our CS1 course grade consists of ∼10% PAs, ∼10-15% CAs, ∼15-20% LAs, ∼5-10% class participation, and ∼50-60% in-person proctored exams. The high exam weight enables gentler policies on allowing some collaboration.

3 Prevalence of code replacement in program histories

3.1 Setup

We selected 5 LAs from our 100-student section of a CS1 course offering in Winter 2019. We chose that quarter due to it being the most recent "normal" quarter taught by one of this paper's authors, prior to the disruptions caused

by the Covid pandemic starting in 2020, with effects of those disruptions still being felt in our most recent quarters.

We chose the 5 labs seeking a spread across chapters in the middle of the course of weeks 3-8, and choosing labs with enough variability in student solutions that a similarity of 90% or above in the zyBooks similarity checker was suggestive of potential copying from a classmate (in contrast to some programs where highly-similar solutions are common due to there not being many different solutions). We chose to do 5 labs based on available time, as each lab took several hours to manually analyze. The 5 selected labs are listed in 1. All the labs chosen are known as "zyBooks Maintained Labs" or ZMLs, provided by zyBooks to instructors for use as programming assignments (80 total are provided, of which we use about 40, and we add more of our own). Being used at hundreds of universities, zyBooks Maintained Labs have a higher likelihood of having solutions available online, and thus we modify some of them slightly each term so online solutions won't work, but those 5 were unmodified. Instructor solution sizes ranged from 20-40 lines and averaged 35 lines.

We trained two senior CS-major undergraduate researchers to manually detect code replacement, involving a judgment that a program in a student's program history was not derived from the immediately-previous program instance in that history. Key indicators included substantial differences in the algorithm, identifier names, statement ordering, expression term ordering, whitespace usage, brace style, comments, and more. But frankly, for people comfortable with programming, code replacement in these CS1 programming assignments is usually glaringly obvious, and easily distinguishable from normal program development even considering large additions to the code between program instances. The researchers were given access to a zyBooks tool that gives instructors/TAs/graders easy access to the program histories, and examined every recorded program instance in those histories, which typically ranged from 5 to 50 instances for a student in one lab. For each lab, they examined the full histories of 50 randomly selected students and recorded every instance of code replacement. Their work was checked by a CS instructor with 10+ years experience of investigating CS1 cheating cases and following through on sanctions and student conduct referrals for dozens of students, and who also happened to chair the university's Academic Integrity committee.

3.2 Results

Table 1 provides results on the prevalence of code replacement. On average per lab, 6 of the 50 students or 12% were detected performing code replacement. On a particularly challenging lab (4.10), 26% were detected performing code replacement.

We ran the zyBooks similarity checker for the 5 labs, to see if the code-

Table 1: Prevalence of code replacement, of 50 students.

LA	# code replacers	% code replacers	# w/out high-sim
3.17: Leap year	5	10%	1
4.10: Name format	13	26%	6
5.10: Count input len	4	8%	2
6.21: Acronyms	2	4%	1
8.15: Word frequency	7	14%	6
Average	6	12%	3

replacing students would have been flagged as having highly-similar code to any other student in our 100-student section. Based on experience, students scoring above a 90% similarity tend to be the suspicious cases, forming a "high similarity" list that might trigger investigation for cheating. Interestingly, 3 of the 6 code replacers (on average) did NOT appear in the high-similarity list, meaning their code was not at least 90% similar to any other student. In our experience, code replacement is strongly suggestive of cheating, and thus 3/50 or 6% of the class was likely cheating on assignments that would not have triggered a cheating investigation using similarity checking. For the other half of code replacers who did appear on the high similarity list (having >= 90% similarity to at least one other student), the existence of a code replacement in their program history strengthens confidence that cheating occurred, vs just happening to have developed similar code to another student.

4 Results

4.1 Initial solution copiers

In the manual analysis of program histories to detect code replacement instances, we noticed some students were clearly copying a complete solution from the very beginning. Some might have a code replacement later if that initial solution didn't work, but others were successful in gaining full points from that first copied solution. Curious about the prevalence of initial solution-copiers, we re-analyzed program histories looking for cases where the student's first program instance was a complete solution – unusual because we require all development to be done in zyBooks and we strongly encourage and teach incremental design. A complete solution from the start is not always due to copying, because some students (against advice) write the entire program at once without any development runs or submissions to the auto-grader along the way. Thus, we looked for more indications that the initial complete solution was copied, including looking for unusual code style not taught by the book or

instructor. Style refers to brace usage, indenting, whitespace, variable naming, chosen constructs, etc. For example, our CS1 does not teach and in fact forbids use of C++ arrays, such as v[i], instead requiring vectors, such as v.at(i). Other examples include 100% similarity with other students or left-aligned code due to pasting copied code.

Thus, solutions using arrays are almost certain to have been copied from online, where array-based solutions are common (partly because tutors on Chegg seem to mostly have a C background and C only has arrays, vs. a C++ background). Another indication included 100% similarity from the similarity checker with other students. Another was left-aligned code, which seems to happen sometimes when students paste copied code.

Table 2 provides results. Initial code copying appears to be an even more substantial problem, with 12 of 50 students or 24% initially copying. These initial copiers turned in code that was not their own from the very first submission.

Table 2: Prevalence of initial copying, of 50 students.

LA	# code replacers	% code replacers	# without high-sim
3.17: Leap year	9	18%	1
4.10: Name format	13	26%	1
5.10: Count input len	17	34%	4
6.21: Acronyms	13	26%	4
8.15: Word frequency	7	14%	4
Average	12	24%	3

Nine of those 12 had high similarity with at least one other student, but 3 of the 12 did not. Furthermore, on average, 3 of the 12 initial copiers had a subsequent code replacement, because their initially-copied solution did not get full credit. We refer to trying multiple copied solutions as "solution hopping". On average, 6% of students in the class solution-hopped, either by trying first on their own and then code-replacing two or more times, or by initial copying and then code-replacing at least once.

In the classes examined, 6% of students were likely cheating via code replacement yet were not suspicious in the similarity checker. This number is quite high when considering that detected cheating rates (primarily via similarity checking) at at CS1 range from 5% to 20% depending on the instructor and term. In other words, code replacement detection alone could potentially increase the detected cases by 30% (6/20) to 120% (6/5), which is quite substantial. Initial copy detection could increase that even further.

5 Automatic detection of code replacements

Manual analysis was time consuming, requiring 2-3 hours per lab for 50 students. Thus, we wanted to know if code replacement could be detected automatically. Following the philosophy of trying simple approaches first, we wrote a script to run a text difference function on every sequential pair of programs in a student's program history. We used the difflib library [17] available in Python, namely the difflib. Differ() function ("diff" for short). From diff's return data, we counted the lines that differed. We experimented and determined 70% was a reasonable threshold to distinguish code replacement from normal development based on evaluation of hundreds of examples, but a more general adaptable approach may be a topic for future work. We compared diff to our manual analysis, which had marked every examined run as code replacement or not.

Our data showed that the simple diff approach has high sensitivity, catching about 98% of all actual code replacements (or about 19 in 20 cases). Its specificity is good at 93.3% but a higher percent would be preferred to reduce false alarms. Thus, simple diff is likely usable for detecting code replacement, but future work would be beneficial to reduce the false alarm rate.

6 Conclusions

Given the growing number of solutions available online, the easy availability of low-cost online programming contractors, and the rise of AI-driven code writing systems, we were concerned that similarity checking might not be catching as high a percentage of cheating as desired. Thus, we investigated the prevalence in our CS1 class of "code replacement" in student program histories, because code replacement is usually highly-suggestive of cheating. We found the rate to be 12%. Importantly, 6% of all students were likely cheating via code replacement but did not show up as suspicious in the similarity checker, which in some terms could double the number of detected cheating cases. In our analyses, from-the-start copying also seemed to be a substantial problem as well, occurring in 24% of the labs, being more commonly detected by similarity checking but with some cases sneaking undetected past similarity checking. We repeated our experiments in a different CS1 at a different university using a different language, and obtained nearly identical results. Additionally, we showed that a simple text difference approach could detect code replacements with reasonable accuracy, but future work is needed to reduce false alarms.

References

[1] Susan Adams. "This $12 billion company is getting rich off students cheating their way through covid". In: *Forbes. Retrieved February* 26 (2021), p. 2021.

[2] Ibrahim Albluwi. "Plagiarism in programming assessments: a systematic review". In: *ACM Transactions on Computing Education (TOCE)* 20.1 (2019), pp. 1–28.

[3] Chegg.com. *Chegg.com*. `https://www.chegg.com`. Accessed: August 2022.

[4] Jianyang Deng and Yijia Lin. "The benefits and challenges of ChatGPT: An overview". In: *Frontiers in Computing and Intelligent Systems* 2.2 (2022), pp. 81–83.

[5] Discord.com. *Discord.com*. `https://www.discord.com`. Accessed: August 2022.

[6] Barbara J Ericson and Bradley N Miller. "Runestone: A platform for free, on-line, and interactive ebooks". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 2020, pp. 1012–1018.

[7] GitHub.com. *GitHub.com*. `https://www.github.com`. Accessed: August 2022.

[8] Chelsea L Gordon, Roman Lysecky, and Frank Vahid. "The rise of program auto-grading in introductory cs courses: A case study of zylabs". In: *2021 ASEE Virtual Annual Conference Content Access*. 2021.

[9] Dianne Hagan and Selby Markham. "Teaching Java with the BlueJ environment". In: *Proceedings of Australasian Society for Computers in Learning in Tertiary Education Conference ASCILITE*. 2000.

[10] JPlag. *Software plagiarism detector*. `https://jplag.ipd.kit.edu/`. Accessed: 2022.

[11] Vedran Ljubovic and Enil Pajic. "Plagiarism detection in computer programming using feature extraction from ultra-fine-grained repositories". In: *IEEE Access* 8 (2020), pp. 96505–96514.

[12] Sad CS Major. *How UCLA Admins Could Stop The GitHub Cheating and Let Us Get Back To Learning CS*. `https://medium.com/@joe_bruined/how-ucla-admins-could-stop-the-github-cheating-and-let-us-get-back-to-learning-cs-8f95e8bf6850`. Accessed: November 2024. 2017.

[13] David J Malan, Brian Yu, and Doug Lloyd. "Teaching academic honesty in CS50". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 2020, pp. 282–288.

[14] Nhan Nguyen and Sarah Nadi. "An empirical evaluation of GitHub copilot's code suggestions". In: *Proceedings of the 19th International Conference on Mining Software Repositories*. 2022, pp. 1–5.

[15] Ashley Pang and Frank Vahid. "Performance Analysis and Interviews of Non-CS-Major Students Sanctioned for Cheating in CS1". In: *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*. 2024, pp. 374–380.

[16] Lutz Prechelt, Guido Malpohl, Michael Philippsen, et al. "Finding plagiarisms among a set of programs with JPlag." In: *J. Univers. Comput. Sci.* 8.11 (2002), p. 1016.

[17] Python Software Foundation. *difflib Python Library*. `https://docs.python.org/3/library/difflib.html`. Accessed: August 2022.

[18] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. "Winnowing: local algorithms for document fingerprinting". In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003, pp. 76–85.

[19] Gina Sprint and Jason Conci. "Mining github classroom commit behavior in elective and introductory computer science courses". In: *The Journal of Computing Sciences in Colleges* 35.1 (2019).

[20] Narjes Tahaei and David C Noelle. "Automated plagiarism detection for computer programming exercises based on patterns of resubmission". In: *Proceedings of the 2018 ACM Conference on International Computing Education Research*. 2018, pp. 178–186.

[21] Frank Vahid, Ashley Pang, and Benjamin Denzler. "Towards Comprehensive Metrics for Programming Cheat Detection". In: *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 2024, pp. 1361–1367.

[22] Frank Vahid et al. "Impact of Several Low-Effort Cheating-Reduction Methods in a CS1 Class". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 2023, pp. 486–492.

[23] Lisa Yan et al. "TMOSS: Using intermediate assignment work to understand excessive collaboration in large classes". In: *Proceedings of the 49th ACM technical symposium on computer science education*. 2018, pp. 110–115.

[24] zyBooks.com. *zyBooks.com*. `https://www.zybooks.com`. Accessed: August 2022.

Data Science Academy: K-12 Broadening Participation Program Conducted by Undergraduate Students*

Shirin Haji Amin Shirazi,Paea LePendu, Mariam Salloum
Computer Science
University of California, Riverside
Riverside, CA 92521
{shaji007,paealp,msall001}@ucr.edu

Abstract

Developing social, management, and academic skills is essential in any undergraduate program. Opportunities to explore future career possibilities, especially in a STEM context, can profoundly impact students' career choices. The Data Science Academy (DSA) is designed to give undergraduates experience in teaching and academia and equip them with valuable personal and professional skills. The DSA hosts coding camps in data science for middle and high school students, led and managed by trained undergraduate leaders. This article shares our experiences with the academy and examines its impact on both the participants and the undergraduate leaders who guide them.

1 Introduction

Research indicates that students enroll in undergraduate programs with diverse motivations, such as a genuine interest in the subject, the aim to prepare for a career or a commitment to others [14]. However, many undergraduate programs prioritize technical proficiency within specific majors but frequently fall short of equipping students with vital social, personal, and professional skills. This gap is particularly noticeable in areas like communication, teamwork, and

problem-solving, all of which are essential for career success. Employers have raised concerns about graduates lacking these key competencies, underscoring a disconnect between academic training and the demands of the modern workplace [1, 15, 20, 17]. In this paper, we propose a data science coding camp designed to provide undergraduates with valuable professional experience alongside technical skills in their studies. Additionally, the camp serves as an outreach initiative for K-12 students, introducing them to the possibilities within the field of data science.

Computer Science continues to be one of the most in-demand skills across various industries [19]. Despite the availability of a wealth of publicly accessible resources, many individuals are only introduced to or gain significant experience with computing after completing their formal education. This delay is often due to economic, geographic, and educational barriers. In response, numerous coding camps [13] have been developed to introduce students to computing earlier in their education, with a focus on topics like robotics and game development.

While the concept of coding camps is not new, offering a structured program that targets and measures the experiences and benefits for those involved in planning and executing these camps could provide fresh insights. Our vision extends beyond teaching data science; we propose a framework where undergraduate students not only lead such camps but also gain meaningful opportunities for professional development, networking, teamwork, and management as part of their academic programs. This approach fosters a collaborative community of students who share an interest in data science, thus enriching their engagement and growth in their chosen fields. Engaging undergraduate students in teaching-related activities is crucial, especially in fields like Computer Science, where the demand for educators is projected to grow significantly. According to the U.S. Bureau of Labor Statistics, employment of postsecondary teachers is expected to increase by 8% from 2023 to 2033, outpacing the average for all occupations [18]. By fostering teaching opportunities early on, institutions can better prepare students for future roles in education, industry, or academia, where mentoring and team leadership are critical competencies.

To our knowledge, the Data Science Academy (DSA) is the first data science-focused program for students in grades 7-12, primarily led by undergraduates. Launched in 2018, DSA has since provided opportunities to over 15 cohorts. The program serves as an introduction to computer science, data science, and their real-world applications. Exposing students in grades 7-12 to data science has the potential to offer long-term career advantages, as computing and data are integral to a wide range of industries.

It is important to note that DSA is structured to introduce participants to the fundamentals of data science and computing before delving into more

technical content. The primary objective is to allow students to explore their interests in these fields and decide whether they wish to pursue them further. As such, the program is designed to be interactive, engaging, and inclusive, appealing to a diverse group of students with varied interests.

2 Related Work

Outreach programs have been widely implemented across various fields and institutions. Gonzalez et al. [5] share their experience with a data science and deep learning-focused code camp for high school students, while Chen et al. [3] explore the introduction of algorithms and computational thinking in code camps designed for middle school students, emphasizing broader humanistic inquiry.

Many of these initiatives align with the CSforALL movement [4], which has been gaining significant momentum over the past decade. CSforALL is a national initiative aimed at integrating high-quality computer science education into K-12 curricula across the United States. Its mission is to ensure equitable access to comprehensive computer science learning opportunities for all students and teachers, paving pathways to college and career success. The organization collaborates with a diverse network of schools, districts, providers, funders, and researchers to advance its goal of providing quality computer science education nationwide. Numerous projects inspired by CSforALL have reported notable improvements in student interest, engagement, and coding proficiency [4, 3, 5, 12, 10, 16].

While the Data Science Academy (DSA) shares similarities with these initiatives, particularly in its outreach to K-12 students—especially those from minoritized backgrounds—DSA takes a distinctive approach. Most notably, DSA prioritizes undergraduate students as the primary participants. By leveraging the successful frameworks of code camps, DSA achieves dual objectives: first, to provide undergraduates with an opportunity to develop and enhance leadership, management, communication, and technical skills; and second, to expose K-12 participants to computer science, data science, and ethical computing.

Efforts to train undergraduates in personal and professional skills have traditionally included internships and formal programs [9, 8, 11, 6]. Kapoor et al. [9] highlight the importance of internships in fostering professional growth, while Hug et al. [8] discuss the benefits of Peer-Led Team Learning (PLTL) in improving confidence, self-efficacy, and technical abilities. However, while internships are invaluable for student development, they are not universally accessible to all undergraduates. Programs like DSA address this gap by offering a more inclusive, structured platform for undergraduate professional develop-

ment.

To our knowledge, DSA is the first initiative to integrate undergraduate training in soft skills, such as leadership and communication, with a K-12 outreach program. This unique model not only prepares undergraduates for future careers but also empowers the next generation of learners with exposure to critical STEM fields.

3 Structure

The program is supervised by a faculty advisor from our department, who oversees the hiring process, engages other faculty members, liaises with schools, and may also participate in teaching sessions. We welcome other interested faculty to join the advisory board to further enrich the program. However, the primary responsibility for organizing, planning, and executing the program lies with undergraduate students, referred to as the DSA Leaders. They take on the main workload to ensure the program's success.

3.1 DSA Leaders

The majority of the activities and lessons are conducted by undergraduate students, called DSA leaders.

- **Application**: The team advertises the program, holds rallies, and encourages students to apply and explore the opportunities within it. Interested students can apply to interview and volunteer at an event to become acquainted with the process and responsibilities of their positions.

- **Interview/Shadowing**: Upon expressing interest, each candidate is interviewed by faculty to ensure they are in good academic standing and genuinely committed to the program. Following the interview, candidates are invited to volunteer at a camp, which provides them with hands-on experience and allows senior DSA Leaders to evaluate their fit for the role. A key criteria is continuity and succession planning for the program over time.

- **Hiring/Training Process**: After successfully completing the interview and camp experience, candidates are officially hired at an hourly rate to support future camps. New hires participate in multiple in-person and online sessions to learn program logistics and practice their responsibilities. These sessions cover a range of topics, including but not limited to teaching methods, fostering a growth mindset, teamwork and responsibility management, effective communication, inspiring younger students, and strategies for classroom management and effective teaching. DSA roles may vary from responsibilities such as coordinating with schools, organizing food and activities,

to delivering lectures, and providing on-site learning support. The selected group meets regularly with the faculty advisor to ensure thorough and timely preparation. Additionally, leaders collaborate independently to plan lessons, brainstorm activity ideas, and conduct practice runs for any scheduled sessions.

4 Program Design

The DSA program offers flexible formats to accommodate various needs, including a two-day weekend event during the school year, a five-day summer camp, and a quarter-long training spread across multiple weekends. Based on our experience, shorter, consecutive camps work best for younger middle school students, as they align well with the primary goal of providing exposure. Conversely, a longer, more detailed, and technical program benefits high school students seeking deeper engagement. Table 1 shows a sample 5-day schedule of a previous DSA offering.

Table 1: 5-day DSA Schedule

Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
Entry Survey	Drop-off	Drop-off	Drop-off	Drop-off
Breakfast	Breakfast	Breakfast	Breakfast	Breakfast
Intro 1	Encryption	DS 1	WordClouds 1	Ethics
Icebreaker	Game	Game	Tour	Game
Intro 2	Chat Bots	DS 2	WordCloud 2	Showcase Prep
Lunch	Lunch	Lunch	Lunch	Lunch
Showcase Prep & Pickup	DS-PATH talk & Pickup	Showcase Prep & Pickup	Showcase Prep & Pickup	Award Ceremony & Exit Survey

Since its inception in 2018, DSA has adapted to different schedules and formats. Initially an in-person program, DSA transitioned online during the COVID-19 pandemic in 2020 to ensure continuity. Currently, the academy operates in person on our campus, with an online option still available as needed.

The program is actively advertised to parents across the school district. In alignment with our goal of reaching students from diverse backgrounds, no prerequisites are imposed, acknowledging the varying levels of computer

and programming familiarity among participants. To address this, lessons are designed with essential coding tools, clear step-by-step instructions, and live feedback to minimize confusion. On-site assistance from DSA Leaders is a crucial component of this supportive structure.

4.1 Language and Platform

Python is selected as the programming language for the camp due to its high-level syntax, extensive library support for data science and machine learning, and widespread applicability across various domains [7]. For an accessible and user-friendly environment, we utilize Google Colaboratory [2], a free, cloud-based platform that students can access through their school accounts. This choice ensures that students can easily retrieve their work and access resources even after the camp concludes, allowing them to revisit lectures and reinforce their learning independently.

4.2 Project Showcase

At the beginning of each camp, students are informed that they will be required to complete and present a project of their choice by the end of the program. They have the flexibility to use any dataset, text, coding technique, or package for their projects. Each day includes dedicated project time, allowing students to apply newly learned concepts and advance their work progressively. This activity has proven to enhance student engagement, leading to a variety of innovative projects, such as analyses with diverse datasets, word clouds, chatbots, and encryption techniques.

This hands-on, project-based approach reinforces learning while encouraging creative problem-solving, preparing students for real-world applications in data science and coding.

While this segment gives students the freedom to explore their interests, DSA leaders play a crucial role in maintaining engagement. They provide follow-up support, assist in generating project ideas, offer technical guidance, and help keep students motivated throughout the process.

4.3 Lessons and Other Activities:

Our primary goal is to design a program that is appealing, engaging, and informative. For many participants, DSA may be their first experience with data science—or even computing in general. Therefore, it is crucial to create an inviting and inclusive environment that encourages students to envision themselves continuing in these fields.

For each lesson, a Google Drive repository is prepared, containing Python code files (Colab Notebooks), data files, and links to helpful resources. (Links to specific lessons have been omitted to maintain the anonymity of this article.) The main DSA Lessons are "Introduction and Setup", "Python Programming Basics", "Encryption", "Data Analysis and Visualization", "Ethical Computing", "Chat Bots", "Word Clouds", and "Social Activities and Icebreakers".

At the end of the program, students present their completed projects and are recognized with awards for their efforts. This final showcase celebrates individual achievements while also reinforcing the collaborative and supportive environment fostered throughout the program.

5 Evaluation Methodology

Since starting with 28 students in 2018 as an in-person program, our Data Science Academy has grown to host up to 150 participants at once, with over 500 students participating in total since its inception. In this section, we discuss the evaluation methodology introduced in Fall 2022 to measure the program's impact.

The goals of the Data Science Academy focus on two main groups impacted by the program: the *"DSA Leaders"*, who are undergraduates responsible for conducting the academy, and the *"attendees"*, the K-12 students who participate in the DSA as an extracurricular activity.

5.1 Attendees:

To evaluate the effectiveness of DSA, we implemented Entry and Exit surveys for recent sessions to capture self-reported impacts on attendees. They are asked to complete a survey designed to gather demographic information as well as insights into their personal, professional, and technical interests relevant to the camp.

Some of the key questions included in the survey are as follows (The students are asked to answer these questions based on a 1-5 Likert Scale):

- "How would you rate your proficiency with coding?"
- "How would you rate your proficiency with data analysis?"
- "How confident are you with programming?"
- "Agree/Disagree: I feel like I belong with other students who like CS and DS."
- "Agree/Disagree: I am interested in doing activities related to CS and DS in the future."

– "Agree/Disagree: I am interested in learning about computers and data science."

5.2 DSA Leaders:

We chose to conduct focus groups to understand the DSA Leader experience because the number of DSA leaders hired in the lifetime of the program (19 undergraduate students) is relatively small, making this approach more advantageous for collecting qualitative data in depth.

In a smaller group setting, we can facilitate conversations that delve into their thoughts, experiences, and perceptions, while encouraging dynamic exchanges that add richness to the data. We believe this approach will allow us to capture a more comprehensive understanding of the experience and its impact on undergraduate student life without introducing biases about expected outcomes.

Some of the main conversation threads and question highlights are as follows:

– "How has being a DSA leader impacted your undergraduate experience?"

– "How has being a DSA leader impacted your academic performance in computer science courses overall?"

– "Has being a DSA leader influenced your interest in pursuing graduate studies in computer science? If so, how?"

– "In what ways has being a DSA leader expanded your network within the computer science department or the broader academic community?"

These questions and topics are introduced in casual conversation, allowing DSA leaders to express themselves more openly and naturally. This informal approach encourages authentic responses and a deeper exploration of their perspectives.

6 Results

In this section, we share our experiences with the Data Science Academy. While the program has been offered since 2018, this study specifically examines the five sessions conducted since Spring 2023.

All of the offerings in this study were structured as two-day, in-person weekend camps, hosting a total of 114 students. During this period, a total of 19 DSA Leaders were actively involved, either hired previously or joining after Spring 2023. On average, 9 DSA Leaders attended and conducted each camp session (not all DSA Leaders are assigned to every camp due to common scheduling conflicts).

6.1 Attendee Experience

We categorized the questions on the entry and exit surveys into key impact categories – "Sense of Belonging," "Proficiency in Data Science," "Proficiency in Programming," "Confidence in Programming," and "Interest in DS and CS" – and measured the improvement between the two surveys.

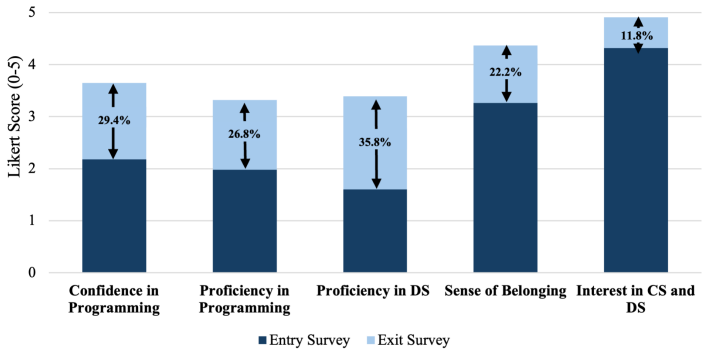

Figure 1: Entry vs. Exit Survey Results Comparison

Our results show a statistically significant (independent paired t-test $p < 0.05$) average increase of 28.55% in "Confidence in Programming," "Sense of Belonging," "Proficiency in Programming," and "Proficiency in DS" among participants in the most recent year of DSA as demonstrated in Figure1. Attendees reported an 82% satisfaction rate with the overall experience, along with an impressive 97% satisfaction score when asked about their experience with the DSA Leaders.

While studying the objective long-term career effects of DSA on attendees may not be feasible, we have observed an encouraging trend: several former participants have reached out post-graduation to express sustained interest in data science, report involvement in educational projects, or reconnect with the program. We plan to establish a framework for maintaining contact with former attendees to assess the program's long-term impact over time.

6.2 DSA Experience

The following notes have emerged from DSA focus meetings:

- **Invaluable Community of Peers Interested in Similar Topics**: Being a DSA Leader allowed students to connect with others who share an interest in data science and teaching. This role provided opportunities to exchange

information about programs, seminars, and career connections, and fostered valuable friendships, enhancing their sense of belonging and community.

- **Increased Interest in Involvement in Other Events and Communities**: Observing how the DSA leadership role opened new doors, students became more interested in participating in other programs and opportunities. For instance, many went on to work as undergraduate tutors in other peer-teaching initiatives at our institution.

- **Enhanced Communication Skills and Interest in Teaching**: Leaders reported that teaching in a classroom setting helped them develop communication skills, enabling them to explain problems in simpler terms, discuss technical issues effectively, and seek solutions collaboratively. This experience also inspired some leaders to consider teaching as a potential career path.

- **Improved Communication with Faculty**: Due to their hands-on responsibilities, leaders developed friendly and approachable relationships with supervising and participating faculty members. This connection helped demystify faculty roles and opened doors for mentorship, advice, and additional opportunities.

- **Significant Interest in MS/BSc Program Applications**: Many senior DSA Leaders went on to apply for our extended MS/BSc program. While this may not be solely attributed to the DSA experience, leaders noted that working alongside high-achieving peers and discussing graduate opportunities with faculty positively influenced their decisions.

7 Conclusion

In this paper, we report our experience with Data Science Academy, an initiative to provide undergraduate students with the opportunity to develop professional skills and experience teaching while offering an outreach DS and CS program to middle school and high school students.

References

[1] Karen Anewalt and Jennifer Polack. A curriculum model featuring oral communication instruction and practice. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 33–37, 2017.

[2] Ekaba Bisong and Ekaba Bisong. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64, 2019.

[3] Yesheng Chen, Zhen Chen, Shyamala Gumidyala, Annabella Koures, Seoyeon Lee, James Msekela, Halle Remash, Nolan Schoenle, Sarah Dahlby Albright, and Samuel A Rebelsky. A middle-school code camp emphasizing digital humanities. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 351–357, 2019.

[4] CSforALL. Csforall: Computer science for all, 2024. Accessed: 2024-11-19.

[5] Rosemarie Santa Gonzalez, Tsion Fitsum, and Michael Butros. High school summer camps help democratize coding, data science, and deep learning. *arXiv preprint arXiv:2410.02782*, 2024.

[6] Shirin Haji Amin Shirazi, Mariam Salloum, Annika Speer, and Neftali Watkinson. An experience report: Integrating oral communication and public speaking training in a cs capstone course. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, pages 450–455, 2024.

[7] Mingzhe Hu and Yu Zhang. An empirical study of the python/c api on evolution and bug patterns. *Journal of Software: Evolution and Process*, 35, 2022.

[8] Sarah Hug, Heather Thiry, and Phyllis Tedford. Learning to love computer science: Peer leaders gain teaching skill, communicative ability and content knowledge in the cs classroom. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 201–206, 2011.

[9] Amanpreet Kapoor and Christina Gardner-McCune. Understanding cs undergraduate students' professional development through the lens of internship experiences. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 852–858, 2019.

[10] Paea LePendu, Cecilia Cheung, Mariam Salloum, Pamela Sheffler, and Kelly Downey. Summer coding camp: Curriculum, experiences, and evaluation. In *2021 ASEE Virtual Annual Conference Content Access*, 2021.

[11] Gary McDonald and Merry McDonald. Developing oral communication skills of computer science undergraduates. In *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer science education*, pages 279–282, 1993.

[12] Bamshad Mobasher, Lucia Dettori, Daniela Raicu, Raffaella Settimi, Nasim Sonboli, and Monica Stettler. Data science summer academy for chicago public school students. *ACM SIGKDD Explorations Newsletter*, 21(1):49–52, 2019.

[13] Jari Porras, Antti Knutas, Jouni Ikonen, Ari Happonen, Jayden Khakurel, and Antti Herala. Code camps and hackathons in education - literature review and lessons learned. *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2019.

[14] Anya Skatova and Eamonn Ferguson. Why do different people choose different university degrees? motivation and the choice of degree. *Frontiers in psychology*, 5:1244, 2014.

[15] Society for Human Resource Management. Employers say students aren't learning soft skills in college, 2023. Accessed: 2024-11-12.

[16] Shashank Srikant and Varun Aggarwal. Introducing data science to school kids. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pages 561–566, 2017.

[17] The Times. We've got good degrees but we can't find good jobs, 2023. Accessed: 2024-11-12.

[18] U.S. Bureau of Labor Statistics. Postsecondary teachers, 2023. Accessed: 2024-11-12.

[19] U.S. Bureau of Labor Statistics. Computer and Information Research Scientists: Occupational Outlook Handbook, 2024. Accessed: 2025-02-03.

[20] Wall Street Journal. In demand: The colleges where students start jobs right away, 2023. Accessed: 2024-11-12.

Reviewers — 2025 CCSC Southewestern Conference

Mahesh Balkrishna Chaudhari ..... University of San Francisco, San Francisco, CA
Bryan Dixon ..... California State University Chico, Chico, CA
Zachary Dodds ..... Harvey Mudd College, Claremont, CA
Michael Doherty ..... University of the Pacific, Stockton, CA
Joshua Gross ..... California State University Monterey Bay, Seaside, CA
Essa Abubaker Imhmed ..... Eastern New Mexico University, Portales, NM
Michael Shindler ..... University of California Irvine, Irvine, CA
Megan Thomas ..... California State University Stanislaus, Turlock, CA