The Journal of Computing Sciences in Colleges

Papers of the 16th Annual CCSC Southwestern Conference

April 1st, 2023
University of California – Irvine
Irvine, CA

Baochuan Lu, Editor
Southwest Baptist University

Megan Thomas, Regional Editor
California State University, Stanislaus

Volume 38, Number 10 **April 2023**

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

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

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

Scott Sigman, President (2024), ssigman@drury.edu, Mathematics and Computer Science Department, Drury University, 900 North Benton Avenue, Springfield, MO 65802.

Karina Assiter, Vice President/President-Elect (2024), KarinaAssiter@landmark.edu, Computer Science, Landmark College, 19 River Road South, Putney, VT 05346.

Baochuan Lu, Publications Chair (2024), blu@sbuniv.edu, Division of Computing & Mathematics, Southwest Baptist University, 1600 University Ave., Bolivar, MO 65613.

Brian Hare, Treasurer (2023), hareb@umkc.edu, School of Computing & Engineering, University of Missouri-Kansas City, 450E Flarsheim Hall, 5110 Rockhill Rd., Kansas City MO 64110.

Cathy Bareiss, Membership Secretary (2025), cathy.bareiss@betheluniversity.edu, Department of Mathematical & Engineering Sciences, Bethel University, 1001 Bethel Circle, Mishawaka, IN 46545.

Judy Mullins, Central Plains Representative (2023), Associate Treasurer, mullinsj@umkc.edu, UMKC, Retired.

Michael Flinn, Eastern Representative (2023), mflinn@frostburg.edu, Department of Computer Science & Information Technologies, Frostburg State University, 101 Braddock Road, Frostburg, MD 21532.

David R. Naugler, Midsouth Representative (2025), dnaugler@semo.edu, 5293 Green Hills Drive, Brownsburg, IN 46112.

David Largent, Midwest Representative(2023), dllargent@bsu.edu, Department of Computer Science, 2000 W. University Avenue, Muncie, IN 47306.

Mark Bailey, Northeastern Representative (2025), mbailey@hamilton.edu, Computer Science Department, 198 College Hill Road, Clinton, NY 13323.

Shereen Khoja, Northwestern Representative(2024), shereen@pacificu.edu, Computer Science, 2043 College Way, Forest Grove, OR 97116.

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

Tina Johnson, South Central Representative (2024), tina.johnson@mwsu.edu, Department of Computer Science, Midwestern State University, 3410 Taft Boulevard, Wichita Falls, TX 76308.

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

Bryan Dixon, Southwestern Representative (2023),

bcdixon@csuchico.edu, Computer Science Department, California State University Chico, Chico, CA.

Serving the CCSC: These members are serving in positions as indicated:
Bin Peng, Associate Editor, bin.peng@park.edu, Department of Computer Science and Information Systems, Park University, 8700 NW River Park Drive, Parkville, MO 64152.
Ed Lindoo, Associate Treasurer & UPE Liaison, elindoo@regis.edu, Anderson College of Business and Computing, Regis University, 3333 Regis Boulevard, Denver, CO 80221.
George Dimitoglou, Comptroller, dimitoglou@hood.edu, Department of Computer Science, Hood college, 401 Rosemont Ave. Frederick, MD 21701.
Megan Thomas, Membership System Administrator, mthomas@cs.csustan.edu, Department of Computer Science, California State University Stanislaus, One University Circle, Turlock, CA 95382.
Karina Assiter, National Partners Chair, karinaassiter@landmark.edu.
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.

Platinum Level Partner

Google Cloud
GitHub
NSF – National Science Foundation

Gold Level Partner

zyBooks
Rephactor

Associate Level Partners

Mercury Learning and Information
Mercy College

Revisiting Academic Probation in CS: At-Risk Indicators and Impact on Student Success*

Barbara Martinez Neda, Max Wang, Hamza Errahmouni Barkam, Jennifer Wong-Ma, and Sergio Gago-Masague
Department of Computer Science
University of California, Irvine
Irvine, CA 92617
{barbarm,maxw4,herrahmo,jwongma,sgagomas}@uci.edu

Abstract

Computer Science (CS) is a competitive field with high demand and low admission rates. In the last three years, the CS program at the University of California, Irvine (UCI) received over 15,000 applications and admitted only 14%. Despite being highly selective and admitting top students, we observed over a 10-year span 57% experienced a period of academic probation. The alarmingly high probation rate motivates us to understand challenges that lead CS students to enter probation at UCI. Some of our results aligned with past findings regarding academic performance: high school GPA and math background level have a negative trend with probation, underrepresented groups experience probation at higher rates than the average, and female students have lower probation but higher attrition rates than men. More importantly, over a third of the students enter probation in freshman year, and students on probation leave the field at higher rates than the average. There is also a positive trend linking probation duration to attrition rates. Our results suggest that current probation practices may not be sufficient for students to return to satisfactory academic standing, highlighting the need for proactive and targeted probation interventions.

1 Introduction

CS is one of the largest and most selective majors at UCI, as reflected by an average admissions rate of under 20% in the last three years [7]. Students must handle the transition into higher education while also performing well academically. Despite the high selectivity of the field, a whopping 57% of students experienced a period of probation, and probation students left the field at higher rates than the average. The Computer Science (CS) program at UCI places students on probation if quarterly or cumulative GPA falls under 2.0, writing requirements are not met, minimum unit progress is not made or if the major's program of study is not followed. Students on probation must sign an academic contract to raise awareness and attempt to get them back on track, and if terms are violated, students are subject to disqualification from the major and/or the university.

In this study, we analyze undergraduate CS student characteristics and performance to identify significant indicators of probation. We also analyze relationships between current probation and CS attrition to assess probation practices' efficacy. We focus specifically on academic probation because it depends on students meeting unit, progress, and GPA requirements, making it a useful indicator of satisfactory academic standing.

2 Goals

In this work, we aim to answer the following questions to better understand undergraduate students' challenges regarding probation in the CS program:

(i) *Are there common characteristics in admissions data which identify an increased likelihood for probation?* Detecting common characteristics such as ethnicity, gender, socioeconomic status or GPA in students who enter probation could assist in developing well-tailored support programs.

(ii) *Are there key aspects of students' academic journey and courses taken that increased likelihood for probation?* If common academic patterns or taken courses constantly lead a large amount of students to enter probation, changes in curricula or specific course support could be implemented to maximize student success.

(iii) *Does being placed on academic probation impact students' chances of succeeding in the CS major?* Probation and attrition relationships could point to crucial characteristics that make students more prone to struggling academically. New interventions could be tailored to assist them.

By answering these questions, we will gain insight to propose new academic advising practices to proactively provide assistance to struggling students prior to entering academic probation, as well as reform probation practices to complement probation contracts with academic support.

3 Related Work

There is limited literature focusing on CS students' likelihood of entering probation and how being on probation impacts their academic success. Instead, related literature has focused on factors associated with students' academic performance. For instance, a commonly explored feature in the literature is math background; strong relationships were discovered between having a stronger math background [8, 2], and students' performance. Also, [8, 5] discussed that high school (HS) GPA was a significant predictor of performance in introductory CS classes. Another study identified gender, prior experience and classroom interactions as strong predictors of academic performance [1]. [3] explored feasibility of machine learning for early detection of probation status and differences in performance were uncovered across different demographics. However, further statistical analysis is needed to explain these disparities. Overall, relationships between student characteristics and academic probation status should be explored similar to past work on academic performance.

4 Student Dataset

We worked with a dataset of 1,762 CS students at UCI. It includes students who enrolled from 2011 to 2015 and graduated from 2012 to 2020. Note, this data was collected prior to the pandemic, with the exception of a few fifth year students. The dataset contains admission application records, student term status, quarterly course list, and graduation degree. Admission records included student demographics and HS data. The next component provided academic status information on a term-by-term basis. Data on courses taken included course name, grade, and quarter taken. The last component contained students' degrees and graduation term. We performed two-sample t-tests and Chi-square tests of independence to understand feature relationships with probation. We also evaluated distributions across student background, course experience, and probation experiences.

5 Results and Discussion

In total, 1009 students (57%) found themselves on probation during their academic career. The attrition rate for those who were never on probation was

	Total	Probation
Asian	984	55% (543)
Not Stated	305	51% (157)
Latino	254	77% (196)
White	192	47% (91)
Black	23	87% (20)
Am. Indian	4	50% (2)

Table 1: Distribution of probation across students' reported ethnicity.

GPA	Total	Probation
2.50 - 3.00	2	100% (2)
3.01 - 3.50	94	68% (64)
3.51 - 3.75	250	58% (145)
3.76 - 4.00	675	60% (409)
4.01 - 4.25	623	55% (346)
4.26 - 4.50	118	36% (43)

Table 2: Distribution of probation across students' UHSGPA.

19%, while students who entered probation had a significantly higher attrition rate of 40% ($\chi^2(1, 1762) = 90.99$, p=.001). Higher attrition rates for probation students suggest that current probation practices are not sufficient to support student success. This indicates that there is room for improvement by reforming the current probation practices. In the following sections, we examine student data to find common characteristics that may aid in developing tailored probation support.

5.1 Admissions Data

Table 1 shows probation student breakdown by ethnicity. There is a significant difference in probation ($\chi^2(5, 1762)=63.04$, p=.001) between the groups, and Latinos and Blacks enter probation at the highest rate (77% and 87%). Additionally, 63% of low-income students and 64% of first-generation students entered probation, both at significantly higher rates than the average ($\chi^2(1, 1762)=11.00$, p=.001, $\chi^2(1, 1762)=26.14$, p=.001). These demographics often overlap, pointing to the "leaky pipeline" which fails to retain under-represented groups (URGs) throughout their entire academic career [4]. These results suggest that probation support programs tailored to URGs may yield higher student success rates.

The data also indicates female students enter probation at lower rates than males, yet they end up leaving CS at significantly higher rates ($\chi^2(2, 1762)=4.81$, p=.090). Our results align with previous findings, reiterating the continued need for comprehensive or alternate support for female students. While women in CS are considered underrepresented, our results suggest that they do not face the same challenges generally present for URGs. For instance, [6] shows that female students tend to have harsher self-perceptions, negative interactions with faculty or classmates, lack of female role models, cultural values, and lack of peer and family support impacting their decisions. In this case,

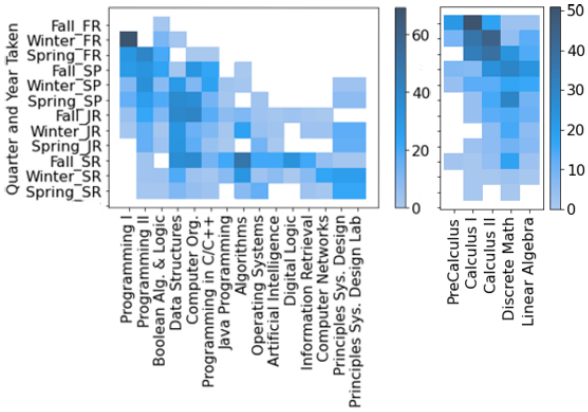

Figure 1: Probation-causing course heatmap and the quarter when the course was taken. FR = Freshman, SP = Sophomore, JR = Junior, SR = Senior.

female students may benefit more from tailored support regardless of probation status.

Lastly, Table 2 displays the probation distribution based on uncapped HS GPA (UHSGPA). Advanced courses (AP or IB) follow a five-point grading scale, allowing UHSGPA to be higher than 4.0. Students with UHSGPA under 3.5 are more likely to be on probation. On the contrary, students with UHSGPAs higher than 4.0 entered probation at rates significantly lower than the average ($t(1760) = -4.464$, $p = .001$). This suggests that students who enrolled in and excelled in advanced courses during HS, represented by a UHSGPA over 4.0, may be better prepared to maintain satisfactory performance in college.

Holistic admission practices allow for students from diverse backgrounds to be admitted. However, URGs and students with lower HS academic achievement enter probation at significantly higher rates. Probation practices could be modified to target challenges that URGs face and support students who may not have had access to the highest levels of academic preparation.

5.2 Course Data

Next, we examined coursework features that may be relevant in early detection of probation. Courses were marked as probation-causing if a student got a C- or lower in the same quarter that they entered probation. We also considered the quarter in which students took the probation-causing course to conclude if students generally struggled more during a specific point in their career.

As Figure 1 shows, we found that for CS courses, most students struggle

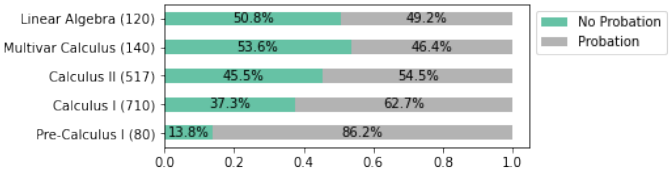

Figure 2: Probation rates of students based on their lowest level of math class taken when entering the CS program (reflects students' math background).

with Programming I. Later on in their studies, they struggle with Algorithms, Data Structures, and Computer Organization. For math, students struggle across all courses early on in their careers. The largest amount of students entered probation after performing poorly in Calculus I and II during their Freshman year.

As Figure 2 shows, 86% of the students who started with Pre-Calculus I, reflecting a weaker math background, entered probation at some point in their careers. This value drops to 46% for students who had a strong calculus foundation and started with Multivariable Calculus. This significant performance difference ($\chi^2(4, 1567)=57.01$, p=.001) suggests that math-focused probation prevention practices may aid students in maintaining or returning to good academic standing. These results highlight the need for assistance mechanisms to give students with less pre-college math knowledge an opportunity to build a strong foundation and increase chances of success in the CS program.

5.3 Academic Probation Data

We analyzed relationships between probation and attrition to gain insight into students' success after being on probation. We found a significant difference in attrition based on probation duration, with higher attrition rates as probation length increased ($t(1760) = -12.227$, $p = .001$). We also evaluated the academic year in which students first entered probation. As illustrated in Figure 3, in total, 41% of probation students first entered probation during their first year, and an additional 30% during their second year. In other words, over two-thirds of probation students first entered probation during the first two years in the program. These findings suggest that there is room for the development of enhancements to aid in increasing CS student success early on in students' careers.

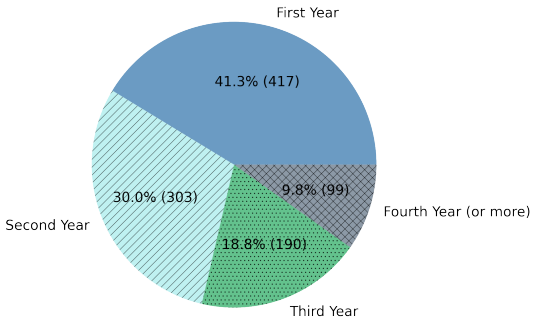

Figure 3: Breakdown of academic year in which students first entered academic probation.

6 Future Work

Future steps include conducting an additional study to obtain surveyed data from CS students to complement our current available data and analysis. Data from student surveys will be examined to consider student experiences in the development of potential future modifications and additions to probation practices. In the long term, students who find themselves struggling academically should have access to resources to gain the necessary skills, habits and knowledge to return to a good academic standing and succeed academically.

7 Conclusion

We analyzed student data to gain insight into academic probation in CS. Based on the results, we are able to answer our initial research questions as follows:

(i) *Are there common characteristics in admissions data which identify an increased likelihood for probation?* We found that URG students experienced higher levels of probation than the average. Also, female students obtained lower probation rates combined with higher attrition rates than males. While alarming, these findings are similar to past findings regarding attrition. GPA has often been considered a strong predictor of academic success, and our results also aligned with previous findings.

(ii) *Are there key aspects of students' academic journey and courses taken that increased likelihood for probation?* The largest amount of students struggled with courses often taken early in their careers, notably Programming I, Calculus I and Calculus II. Additionally, students with a stronger math background were less likely to experience probation.

(iii) *Does being placed on academic probation impact students' chances of succeeding in the CS major?* The majority of students entered probation for the first time early on in their careers. While many students managed to avoid dismissal, they struggled with leaving probation. We observed higher attrition rates for students with more probation quarters. This motivates the need for prompt and tailored support for students to return to satisfactory academic standing soon after entering probation.

These findings suggest that current probation practices may not provide enough support for students to get back on track. Probation contracts could be supplemented with academic support to help students return to satisfactory academic standing. The growing number of student applications and the significant high rate of probation highlight the urgency to better understand the specific challenges that students are facing in CS. The insights and common characteristics found across probation cases presented in this study may set the basis to identify and mitigate the main factors causing students to leave CS.

Acknowledgements

This work has been partially supported by the Division of Teaching Excellence and Innovation at UCI through a Provost Faculty Fellowship. We thank Vice Provost Michael Dennin for guidance and assistance in obtaining the student datasets. We also thank the Admissions Office at UCI for assistance in obtaining and understanding students' admissions data.

References

[1] Lecia J. Barker, Charlie McDowell, and Kimberly Kalahar. "Exploring Factors That Influence Computer Science Introductory Course Students to Persist in the Major". In: *SIGCSE Bull.* 41.1 (Mar. 2009), pp. 153–157. ISSN: 0097-8418. DOI: 10.1145/1539024.1508923. URL: https://doi.org/10.1145/1539024.1508923.

[2] Chen Chen et al. "High School Calculus and Computer Science Course Taking as Predictors of Success in Introductory College Computer Science". In: *ACM Trans. Comput. Educ.* 21.1 (Dec. 2021). DOI: 10.1145/3433169. URL: https://doi.org/10.1145/3433169.

[3] Hamza Errahmouni Barkam et al. “Testing Machine Learning Models to Identify Computer Science Students at High-Risk of Probation”. In: *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2022. Providence, RI, USA: Association for Computing Machinery, 2022, p. 1161. ISBN: 9781450390712. DOI: 10.1145/3478432.3499103. URL: https://doi.org/10.1145/3478432.3499103.

[4] Mica Estrada et al. “Improving underrepresented minority student persistence in stem”. In: *CBE Life Sciences Education* 15.3 (Sept. 2016). ISSN: 19317913. DOI: 10.1187/cbe.16-01-0038. URL: https://www.lifescied.org/doi/10.1187/cbe.16-01-0038.

[5] Terry R. Hostetler. “Predicting Student Success in an Introductory Programming Course”. In: *SIGCSE Bull.* 15.3 (Sept. 1983), pp. 40–43. ISSN: 0097-8418. DOI: 10.1145/382188.382571. URL: https://doi.org/10.1145/382188.382571.

[6] Katarina Pantic and Jody Clarke-Midura. *Factors that influence retention of women in the computer science major: A systematic literature review*. 2019. DOI: 10.1615/JWomenMinorScienEng.2019024384. URL: www.begellhouse.com.

[7] *Undergraduate Admissions Dashboard*. 2022. URL: https://datahub.oapir.uci.edu/Undergraduate-Admissions-Dashboard.php.

[8] Laurie Honour Werth. “Predicting Student Performance in a Beginning Computer Science Class”. In: *SIGCSE Bull.* 18.1 (Feb. 1986), pp. 138–143. ISSN: 0097-8418. DOI: 10.1145/953055.5701. URL: https://doi.org/10.1145/953055.5701.

Active Learning with Code Writing in Large Lectures, In-Person and Online*

Raymond Klefstad[1], Susan Anderson Klefstad[2]
[1]Department of Computer Science
University of California, Irvine
Irvine, CA 92697
klefstad@uci.edu
[2]Independent Researcher
Irvine, CA 92617
saklefstad@gmail.com

Abstract

Programming is a difficult skill for students to master. Although active learning has benefits in STEM, its benefits have been reported mostly for conceptual multiple-choice questions, which do not transfer to code writing. In addition, code-writing questions may increase learning due to the *generation* effect found in cognitive science research, in which there is greater learning when learners generate an answer, rather than recognizing it. To bring the benefits of active learning to teaching programming, therefore, in-class code-writing activities were developed for large university courses. After a short explanation of a concept, students wrote code applying that concept on their own devices, using apps that displayed their responses on the lecture hall screen, and then discussed correctness of their responses, with instructor-led feedback. Code writing activities were effective in teaching students to write correct code, and students found them beneficial. Compared to a traditional course, an active learning data structures course using these activities had far more A's and fewer F's. In-class code writing can bring the benefits of active learning and the generation effect to the skill of code writing.

1 Introduction

Learning programming is difficult for CS students. The "McCracken Group" report in 2001 showed that students' programming skills were low [10], as did a followup report in 2004 [8]. In recent years, teaching programming has become even more challenging, with much larger class sizes, more under-prepared students [6], and more distractions competing for students' attention.

On the other hand, active learning methods have been shown to produce better learning outcomes in many STEM fields, in hundreds of studies over the past couple decades [4], with benefits disproportionately higher for underrepresented groups [6].

Active learning benefits have also been shown in CSE. For example, the active learning methodology Peer Instruction (PI), which uses conceptual multiple-choice questions (MCQs) as a key component of in-class learning, has improved outcomes in CSE [12], [15]. In addition, CSE research has focused on other types of MCQs, including code tracing questions [7] and Parson's problems [3].

1.1 Rationale for Active Learning with Code Writing

However, although work in CSE has focused on teaching concepts through PI and through these other types of MCQs, active learning for the skill of programming has not been as well developed [13]. The conceptual knowledge needed to answer MCQs (a lower level in Bloom's taxonomy [1]) may not transfer to programming skill (a higher level). In fact, in an exploration of students' performance on many different types of exam questions—conceptual questions, Parson's puzzles, code tracing, code reading, code writing, and "explaining in English"— students performed poorly in code writing (29%), even when they performed well in Parson's puzzles (83%) and conceptual questions (79%) [9]. Similarly, code tracing MCQs have weak correlations to code writing [7]. These results concur with 100 years of research in the cognitive and learning sciences showing that transfer of learning is minimal [11].

We therefore wanted to use active learning methods, but with questions that require students to write code. In fact, based upon robust results in the learning sciences, we expect that code writing may be even more effective for learning than MCQs, because cognitive science research has consistently shown a *generation* effect, or the *recall* vs *recognition* effect: greater learning, longer-lasting learning, and increased transfer occur when learners attempt to generate answers themselves, rather than simply recognize them from a list of choices, as with MCQs [2].

2 Goals and Methods for In-Class Code Writing

Considering the benefits of active learning and the benefits of the generation effect, in-class code-writing questions were designed to replace most lecturing in large, programming intensive university courses. With this method, students first generate answers on their own, then optionally (i.e., sometimes) engage in discussion with a peer, and always end with whole-class discussion and feedback, with the following goals:

- Bring the benefits of active learning to code writing.
- Scale well for large classes with hundreds of students.
- Be easy to prepare and integrate into existing lectures.
- Increase students' ability to apply concepts to write code, during lecture.
- Increase students' related ability to read and analyze others' code.
- Better prepare students to later complete larger programming homework problems on their own, with less help from friends or the Internet.

Many classroom response tools are available to support and present MCQs, including hardware clickers and various software apps, but tools to collect and present code writing are needed.[1] A readily-available online discussion app, Piazza, which we were already using for course communication and which supports code formatting, was therefore used for code writing activities in lecture. Subsequently, after positive results with Piazza, a custom web app was developed to support the activities. In addition, because of the remote classes required by the worldwide pandemic, the custom app was also used for online classes meeting synchronously via Zoom.[2]

These in-class code writing activities were used in several different courses, at several levels, all with medium to large size enrollments: a large CS1-Python course with 450 students, a C++ course with 250 students, a data structures course with 200 students, a programming languages course with 200 students, and a Masters degree-level course in advanced programming and problem solving with 120 students.

For all these large-enrollment courses, traditional lectures were replaced with mini-lectures interspersed with in-class coding activities, inspired by the method of PI, with the following format:

- **Mini-lecture** The instructor presents a short explanation of a concept, typically 10 minutes long.
- **Students code individually** Students write code applying that concept on their own devices.
- **Peer discussion** In some cases, students discuss solutions with a peer.

[1] In [13], a custom app was developed to facilitate code writing during class, but it was no longer available at the time of this work.

[2] Details of implementing the activities with Piazza and the custom app are given in Appendix A.

Figure 1: *Students write code on their laptops and discuss as a class in a large lecture hall.*

- **Class discusses code** The instructor and students select interesting responses to analyze for correctness. (Figure 1 shows a picture of this process.) Often, the instructor copies and pastes an interesting solution into the code visualization tool Python Tutor (which also supports JavaScript, C, C++, and Java) to help students see the execution state evolve, illuminating what is wrong in the code [5]. Students are interested to see others' solutions, including different ways of coding correct solutions. The whole class benefits from expert feedback, as found in [14] (which compared instructor-led feedback to peer discussion alone).
- **Repeat** two or more times, depending on time available.

3 Results

Initial experiences proved the code writing activities in lecture to be highly beneficial. The instructor observed positive benefits; students perceived learning benefits; and data assessing learning was also positive.

3.1 Benefits observed by the instructor

The instructor has observed the following benefits:

- **Attention** Students pay close attention to the "just-in-time" concepts presented in the mini-lecture that they need to apply immediately.
- **Scalability** Coding activities work in large classes of 120-450 students.
- **Prompt feedback** The instructor can see students' first attempts to write code and can immediately correct misconceptions, which are not as discernible after students have spent days working on code for an assignment, getting help from teaching assistants, friends, or the Internet.

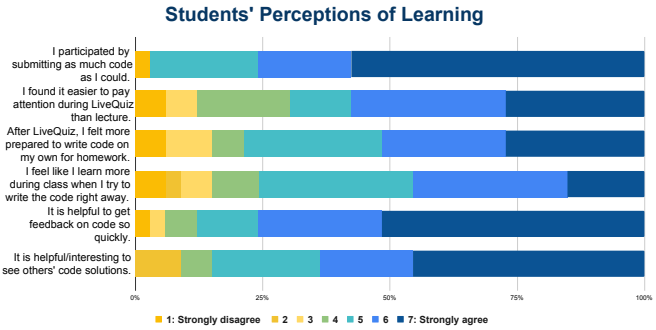

Figure 2: *Students found in-class coding activities to be beneficial, as shown by the teal and blue bars; results were similar for CS1 and upper-division students.*

- **Efficient feedback** Students who make similar mistakes can all learn from instructor-led expert feedback at the same time.
- **Practice in other coding-related skills** Students practice both writing code and reading and analyzing others' code, also valued skills.
- **Faster preparation** Effective coding problems are faster to prepare than effective MCQs, easing the transition to active learning.

3.2 Benefits perceived by students: Survey Results

Students completed surveys of their perceptions of learning for in-class coding activities. As shown in Figure 2, students reported positive learning experiences:

- They participate in the activities, even though they are not graded.
- They learn more in class when participating in coding activities.
- They learn more by trying to write code shortly after hearing a concept.
- They believe it is helpful to analyze others' code.
- They can complete more of their homework on their own, with less help.
- They find it instructive to see that there are many correct solutions.

Open-ended comments were overwhelmingly positive (e.g., *"Always do this!"*) In-class coding activities are frequently mentioned in the standard course evaluations administered by the university (therefore not specifically soliciting feedback about these activities), in answer to the course evaluation question, *"Which course activities contributed most to your learning?"*

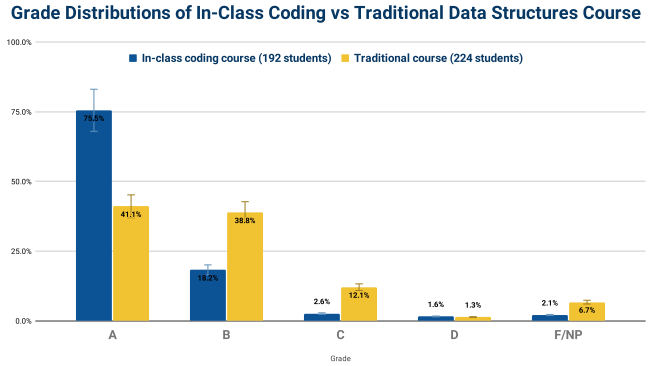

Figure 3: *Compared to a traditional lecture course, an active learning course with in-class coding activities had more A's and 70% fewer F's.*

3.3 Results: Measures of Learning

In-class coding activities were effective in helping students to learn to write correct code. For example, the percentage of correct sequential search functions rose from 36% after lecture alone to 96% after the coding activity.

An active learning version of a Data Structures in C++ course, in which in-class coding activities were always used and often included the peer discussion step, had dramatically more A's (75% vs 41%), and fewer B's (18% vs 39%), C's (3% vs 12%), and F's (2.1% vs 6.7%) compared to a traditional lecture course taught by the same instructor, as shown in Figure 3. That is, the failure rate was reduced by almost 70%.

4 Conclusion

In-class coding activities bring the benefits of active learning and the generation effect to the teaching of programming, even in very large university courses. This method of in-class coding requires active generation of an answer by the students, while being quick and easy to implement for the instructor, with benefits seen across different levels of experience and across many different courses. Active learning has shown robust effects increasing students' learning, and this experience with in-class coding shows that it is a promising way to bring the benefits of active learning to the important skill of writing correct code.

References

[1] Lorin W Anderson. *Bloom's Taxonomy*. University of Chicago Press, 1994.

[2] Elizabeth L Bjork and Robert A Bjork. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. *Psychology and the Real World: Essays Illustrating Fundamental Contributions to Society*, 2(59-68), 2011.

[3] Paul Denny, Andrew Luxton-Reilly, and Beth Simon. Evaluating a new exam question: Parsons problems. In *Proceedings of the Fourth International Workshop on Computing Education Research*, pages 113–124, 2008.

[4] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt, and Mary Pat Wenderoth. Active learning increases student performance in science, engineering, and mathematics. *Proceedings of the National Academy of Sciences*, 111(23):8410–8415, 2014.

[5] Philip J Guo. Online Python tutor: Embeddable web-based program visualization for CS education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pages 579–584, 2013.

[6] David C Haak, Janneke HilleRisLambers, Emile Pitre, and Scott Freeman. Increased structure and active learning reduce the achievement gap in introductory biology. *Science*, 332(6034):1213–1216, 2011.

[7] Amruth N Kumar. A study of the influence of code-tracing problems on code-writing skills. In *Proceedings of the 18th ACM conference on Innovation and Technology in Computer Science Education*, pages 183–188, 2013.

[8] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, and et al. A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bull.*, 36(4):119–150, June 2004.

[9] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. Relationships between reading, tracing, and writing skills in introductory programming. In *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, page 101–112, New York, NY, USA, 2008. Association for Computing Machinery.

[10] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR ' 01, page 125–180, New York, NY, USA, 2001. Association for Computing Machinery.

[11] David N Perkins and Gavriel Salomon. Transfer of learning. *International Encyclopedia of Education*, 2:6452–6457, 1992.

[12] Leo Porter, Cynthia Bailey Lee, and Beth Simon. Halving fail rates using peer instruction: A study of four computer science courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 177–182, New York, NY, USA, 2013. Association for Computing Machinery.

[13] Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. Facilitating code-writing in PI classes. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 585–590, New York, NY, USA, 2013. ACM.

[14] Daniel Zingaro and Leo Porter. Peer instruction in computing: The value of instructor intervention. *Computers & Education*, 71:87–96, 2014.

[15] Daniel Zingaro and Leo Porter. Tracking student learning from class to exam using isomorphic questions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, pages 356–361, New York, NY, USA, 2015. ACM.

A In-Class Coding with Piazza and *Codings*

Piazza. To prepare in-class activities using Piazza so that all students' responses are seen on the instructor's projected screen, then discussed by the whole class, before-class preparation is as follows:

- Post an announcement that students should bring a laptop.
- Create coding problems for class, posting each as a *Note*, organized by date into *Folders*, ideally posting to Piazza just before class.
- Insert links to each coding problem into the lecture slides.
- Create and log in to a *Student* account for the instructor to use during lecture, so as to maintain students' anonymity to increase participation.[3]

[3] We normally choose Piazza's setting that shows students' identities in the instructor's account. If the anonymous option is chosen instead, this step is unnecessary.

Using their own laptops, students navigate to the *Note* with the problem in Piazza, then post their *Followups* with empty code blocks, in order to create screen space for their responses so as to minimize screen redrawing later. They then write their solutions in formatted code blocks and re-post their *Followups*.

For the problems, small functions are used, and the instructor has fluency in the language, and therefore is quick to evaluate solutions while scrolling through them. Even without fluency, however, an instructor could still model the process of evaluating correctness, to help the students improve their own ability to do so. However, an optimal app would cluster solutions so that it is not necessary to scroll through many to find a different one, speeding the flow of the activity.

The instructor continues to scroll through randomly until several different correct and incorrect solutions have been evaluated. Then when the discussion of each problem is complete, the instructor asks students to mark their *Followups Resolved*, so that the instructor's account can still easily find the out-of-class discussions needing responses.

Codings. While Piazza can be used for in-class coding, it is not optimal because of frequent screen redraws. We therefore developed a custom web app called *Codings* to support real-time, in-class coding activities during large lectures, as well as polling questions, pre- and post-quizzes, and data analytics.

Codings deploys client apps for instructors and students, which require accounts to log in, and a server with a database of questions and student submissions. Instructors and students can select a programming language, with support for syntax highlighting and help. The instructor creates a session for each class, which can display multiple problems. All questions and students' anonymous submissions can be viewed during class, using real-time tools.

The instructor app supports creating and editing questions and analyzing submissions both before and during a lecture. Offline, the instructor can view students' submission history and analytics. Codings can generate a report in a CSV file of the data for each student for any specified subset of problems.

The student app allows students to view questions shared by the instructor, and to create, edit, submit, and update their answers with unlimited numbers of revisions. Students can also browse their history of previous submissions, see changes made between revisions, and send submissions to the visualizer for further illumination.

Codings has been used for a number of courses, some in-person and some online during synchronous Zoom lectures during the pandemic. In-class coding worked well during Zoom meetings, with perhaps even more students participating in whole-class discussion through the chat in Zoom.

Early experience with Codings has been extremely promising, and its maintenance and development are ongoing.

CodeVid Studio: Coding Videos with Multimodal Instruction

Kevin Buffardi
Computer Science Department
California State University, Chico
Chico, California, USA
kbuffardi@csuchico.edu

Abstract

This paper describes a novel technique for capturing educational videos that teach coding with improved support for multiple modes of presenting information. I developed "CodeVid Studio" using a combination of lightboard, luma key, integrated development environment (IDE), and digital video capture technologies to innovate how code can be taught through on-demand and streaming video. The paper describes hardware and software configurations to reproduce a style of video that enables demonstration of development within an IDE along with drawing, highlighting, and gesturing to aide instruction. A usability test included comparing a CodeVid Studio recording to one using conventional technologies and found unique advantages of CodeVid Studio.

1 Introduction

There is increasing demand for learning how to code via online and hybrid modalities that use educational videos. Online tutorials (e.g. Khan Academy [7]) and Massive Online Open Courses (MOOC) [3][4][13]) deliver asynchronous tutorials for many computer science topics. Likewise, the Covid-19 pandemic forced many universities to transition to teaching courses via web-conferencing services (such as Zoom [14]) for instruction. Even with in-person courses, some educators have adopted techniques such as "flipped classrooms" [5] that involve students watching video instruction at home before engaging in hands-on activities in the classroom.

However, video recordings of coding instruction can leave both instructors and students wanting. Effective teaching involves sharing information in different ways, including text, images, animation, and audio [2] but current video

formats rarely enable effective use of all these modes of presentation. Conventional methods for coding videos involve trade-offs between these different modes of communicating information. For example, if the instructor records their screen they can show the text of their code, but it is not easy to simultaneously accompany the code with illustrations nor to gesture to explain relationships between code and their illustrations.

Consequently, I designed a studio for recording and streaming video instruction that enables multiple modes of communication for teaching how to code. I built *CodeVid Studio* to demonstrate live coding in an IDE while empowering the instructor to draw and gesture to explain their code. This paper details the implementation of *CodeVid Studio* and provides initial insights into students' experiences with *CodeVid Studio* recordings in comparison to traditional video recording methods.

2 Background

Cognitive load is the amount of information being processed by the brain's working memory; working memory has limited resources so excessive cognitive load overwhelms those resources and detracts from learning. Fortunately, evidence suggests that cognitive load decreases when information is distributed across different modes of communication [11]. Accordingly, Clark and Mayer summarized evidence-based practices for multimedia instruction in a collection of learning principles [2].

In particular, the *Multimedia Principle* emphasizes that humans learn better from words with graphics than words alone. In terms of coding, graphics can illustrate concepts (e.g. memory, data structures, control flow, etc.) to supplement the actual code (words). The *Modality Principle* encourages supplementing auditory explanations with graphics because "the psychological advantage of using audio presentation is a result of the incoming information being split across two separate cognitive channels—words in the auditory channel and pictures in the visual channel—rather than concentrating both words and pictures in the visual channel" [2].

Meanwhile, preliminary research of lightboards—transparent marker boards that allow the instructor to face the camera while drawing and gesturing—has found no burdensome cognitive load while demonstrating potential for effective video presentations [9]. Moreover, recent research on instructional videos revealed that when a student witnesses the instructor's positive emotions through hearing vocal cues and seeing their gestures, they are more likely to reflect the same emotion when learning the subject [8].

However, conventional video instruction tends to violate these principles by emphasizing either textual *or* graphical presentation alone, and having to

choose between graphical presentations *or* a good view of the instructor. For example, sharing one's screen on Zoom while coding in an IDE presents information only through text without complementary graphics or the instructor's gestures to things on screen. In contrast, presenting graphics via digital (e.g. tablet) or marker boards (e.g. dry erase boards or transparent lightboards) only present information graphically without a view of the corresponding code. An instructor may opt to switch between screen-sharing and illustrating on a marker board, but doing so violates the *Contiguity Principle*, which advises that presenting related text and graphics *simultaneously* side-by-side supports improved learning [10]. Consequently, I designed a solution that incorporates IDE development (text) with marker board annotation (graphics) while the instructor provides spoken explanation (audio) and gestures.

3 Design and Implementation

Lightboards are transparent marker boards that allow the presenter to draw in front of themselves while facing the camera. When using a lightboard, the instructor can write and draw as they normally would, but the image (captured on the camera on the other side of the board) is flipped horizontally so that the video captures the writing and drawings in the same orientation as the instructor's view. To capture video, I connected a digital video camera's HDMI output to the studio computer's video capture card.

I flipped the video source horizontally and recorded/streamed with negligable latency using Open Broadcast Software (OBS) [12], a cross-platform Free and Open Source Software (FOSS) program. I also installed Visual Studio Code (VS Code) as the IDE for demonstrating coding skills but other IDEs—or even just a command line interface—would also work as long as they support "dark mode" interface themes with dark backgrounds and high contrast text.

OBS supports screen capture so I configured it to capture a full-screen VS Code window. I arranged the screen capture to match the dimensions of the camera's flipped video and layered the screen capture over it. I superimposed the images via OBS' luma key filter. Luma key removes pixels that are not as bright as a specified threshold, so anything dark in the foreground is replaced by another video source.

I set the threshold to remove the IDE's dark background so that only the bright text and icons remained. Consequently, the instructor and any markings on the lightboard appear behind the IDE text. **Figure 1** shows an example of an instructor drawing an illustration of recursion, alongside the code for the recursive function. Finally, I placed a microphone above the lightboard, angled toward where the instructor stands behind the lightboard. OBS records audio by feeding the microphone's XLR cable to the computer's audio capture card.

The instructor has a wireless keyboard and trackpad on the lightboard table so they can interact with the IDE while also gesturing and drawing on the board. Two televisions are mounted behind the camera–one above the other. The top monitor provides a full screen view of the IDE while the lower monitor previews OBS' combined video sources. The lower monitor displays the compound image (as flipped by the software) and is positioned at roughly the instructor's height so they can see a mirror image—this allows the instructor to see where they are drawing and gesturing while seeing both the lightboard and the IDE.

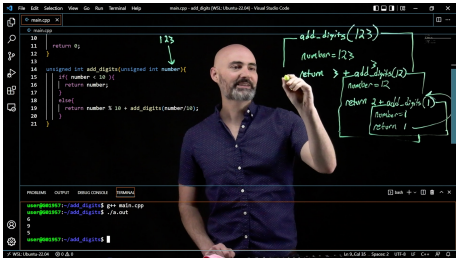

Figure 1: CodeVid recording featuring IDE, marker drawings, and gestures.

Reducing the brightness of these TVs helps mitigate reflections on the lightboard and interference with the camera's lighting. I also added a third monitor that duplicates the screen of the lower TV (mirror recording monitor) in case the instructor needs a closer view of small details and so they can see the recording in its full brightness. **Figure 2** shows the CodeVid Studio from behind the lightboard.

Recording a lightboard requires a relatively dark room so the fluorescent markers contrast with a dark background. Likewise, the instructor stands in front of a black backdrop but has two subject back-lights to help them (i.e. the *subject* of the image) stand out from the backdrop; this is particularly helpful for instructors with dark clothes, skin, and/or hair since the subject lights give detail to the subject while obscuring the backdrop. The lightboard also features integrated subject lights that illuminate the instructor's front. **Figure 3** illustrates the layout.

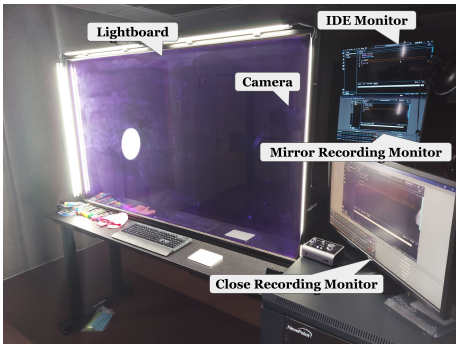

Figure 2: CodeVid Studio components

Chroma key is typically used to remove objects from the background, but can accidentally remove parts of the subject if they wear a similar color (i.e. green clothes in front of a green screen). On the other hand, luma key only

removes dark parts of the IDE and allows the instructor and lightboard to appear *within* the IDE. Consequently, there is not a risk of the lightboard illustrations nor the illustrator being replaced by the luma key filter.

I spent $18,205.71 USD to purchase CodeVid Studio's major hardware (excluding cables, markers, cleaning supplies, furniture, etc.). Note that I purchased the largest lightboard available and chose hardware that exceeded requirements to allow flexibility and future-proofing in case we want to demonstrate resource-intensive computation or record in higher video resolution/frame rate.

In particular, the PTZ camera was much more expensive because our CodeVid Studio shares a room with another video recording setup and the technicians preferred to share a camera for both setups that can be automated to Pan/Tilt/Zoom according to the situation. On a lower budget, a setup could use smaller lightboard (at the sacrifice of drawing space) or even build one from scratch, which may be done (with limitations) for as little as $100 [6]. On a small budget, one may be able to reproduce the features of CodeVid Studio at an order of magnitude lower cost by sacrificing audio/video quality with a less-performant computer.

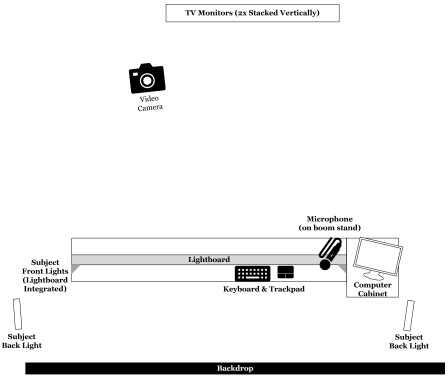

Figure 3: Diagram of room layout for Codevid Studio, with approximate scale.

4 Usability Test

I elicited feedback during a usability test of students' experience learning from CodeVid Studio videos. The usability test involved observing ten students (n=10) using *Codewit.us*[1][1], which features embedded tutorial videos with interactive coding practice problems. I recorded two videos that used the same example to teach recursion. One video used a CodeVid Studio recording (as shown in figure 1) and the other used a typical web-conferencing layout: screencast of an IDE with a picture-in-picture box in the corner with the instructor's face.

I recruited computer science majors who had passed the introductory programming course and enrolled to take the subsequent data structures course

[1]https://Codewit.us/

in the following semester. Participants were randomly assigned to either the CodeVid Studio treatment or the picture-in-picture treatment. I observed the students while using *Codewit.us* to learn about recursion and then practice coding a recursive problem. At the end of the session, each participant was asked about their experience, including: *Rate on a scale from 1-5 (where 1 is strongly disagree and 5 is strongly agree) the statement "The video feature was helpful."* as well as open-ended questions of what they liked most/least.

Student feedback was overwhelmingly positive. Unanimously, all five students in the CodeVid Studio group rated the video helpfulness as 5 and did not provide any negative feedback about the video. The students in the picture-in-picture group also provided positive feedback, but averaged rating the video a 4.8. Moreover, when asked about what they liked least about *Codewit.us*, one (picture-in-picture) student suggested that the video would be better with visuals/graphics to illustrate the topic. The results suggest at least a more positive response to CodeVid Studio in comparison to traditional picture-in-picture videos.

5 Conclusion

CodeVid Studio is a video recording/streaming setup that enables an instructor to demonstrate coding within an IDE while also using spoken explanations, gestures, and drawings to provide rich multimedia presentations. A usability study of videos recorded using this studio found that students responded more positively to them than a picture-in-picture screencast with a "talking head" confined to a box in the corner of the video.

Evidence-based multimedia principles also suggest that CodeVid Studio videos may even improve learning, so future experiments will measure students' cognitive load during the videos as well as their performance of the skills demonstrated after watching the videos. The information provided in this paper should also enable others to recreate the studio on different budgets while using cross-platform Free and Open Source Software.

6 Acknowledgements

This material is based upon work supported by the Department of Education's (DOE) American Rescue Plan Higher Education Emergency Relief Fund and by the Learning Lab, an initiative of California Governor's Office of Planning and Research (OPR). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of either DOE or OPR.

References

[1] Kevin Buffardi, Elena Harris, and Richert Wang. "Codewit.Us: A Platform for Diverse Perspectives in Coding". In: *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2022. Providence, RI, USA: Association for Computing Machinery, 2022, pp. 780–786. ISBN: 9781450390705. DOI: 10.1145/3478431.3499398. URL: https://doi.org/10.1145/3478431.3499398.

[2] Ruth Colvin Clark and Richard E. Mayer. "Applying the Multimedia Principle: Use Words and Graphics Rather Than Words Alone". In: *e-Learning and the Science of Instruction*. John Wiley & Sons, Ltd, 2011. Chap. 4, pp. 66–89. ISBN: 9781118255971. DOI: https://doi.org/10.1002/9781118255971.ch4. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118255971.ch4. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118255971.ch4.

[3] *Coursera*. https://www.coursera.org/. Accessed: 2023-11-13.

[4] *edX*. https://www.edx.org/. Accessed: 2023-11-13.

[5] Michail N. Giannakos, John Krogstie, and Nikos Chrisochoides. "Reviewing the Flipped Classroom Research: Reflections for Computer Science Education". In: *Proceedings of the Computer Science Education Research Conference*. CSERC '14. Berlin, Germany: Association for Computing Machinery, 2014, pp. 23–29. ISBN: 9781450333474. DOI: 10.1145/2691352.2691354. URL: https://doi.org/10.1145/2691352.2691354.

[6] *How to Make a Lightboard for Less Than $100*. https://flippedlearning.org/how_to/how-to-make-a-lightboard-for-less-than-100/. Accessed: 2023-11-13.

[7] *Khan Academy*. https://www.khanacademy.org/. Accessed: 2023-11-13.

[8] Alyssa P. Lawson and Richard E. Mayer. "Does the emotional stance of human and virtual instructors in instructional videos affect learning processes and outcomes?" In: *Contemporary Educational Psychology* 70 (2022), p. 102080. ISSN: 0361-476X. DOI: https://doi.org/10.1016/j.cedpsych.2022.102080. URL: https://www.sciencedirect.com/science/article/pii/S0361476X2200039X.

[9] Mark Lubrick, George Zhou, and Jingsheng Zhang. "Is the future bright? The potential of lightboard videos for student achievement and engagement in learning". In: *EURASIA Journal of Mathematics, Science and Technology Education* 15.8 (2019), em1735.

[10] Roxana Moreno and Richard E Mayer. "Cognitive principles of multimedia learning: The role of modality and contiguity." In: *Journal of educational psychology* 91.2 (1999), p. 358.

[11] Seyed Yaghoub Mousavi, Renae Low, and John Sweller. "Reducing cognitive load by mixing auditory and visual presentation modes." In: *Journal of educational psychology* 87.2 (1995), p. 319.

[12] *Open Broadcaster Software*. `https://obsproject.com/`. Accessed: 2023-11-13.

[13] *udemy*. `https://www.udemy.com/`. Accessed: 2023-11-13.

[14] *Zoom*. `https://www.zoom.com/`. Accessed: 2023-11-13.

7 Appendix 1: Hardware

Hardware	Cost	Notes
Revolution 95" lightboard	8570.11	Lightboard with adjustable-height table + integrated subject lights
Alienware Alienware Aurora R12	3455.16	CPU: Intel i7 (8-Core, 16MB Cache, 3.6GHz) Video: NVIDIA GeForce RTX 3070 8GB Memory: 64GB, DDR4, 3200MHz Storage: 2TB M.2 PCIe NVMe SSD + 2TB 7200RPM SATA
Canon CR-N300 4K NDI PTZ	2938.54	PTZ camera
Dual TV stand	944.14	
Samsung TU7000 50" TV (2x)	910.17	IDE Monitor and Mirror Recording Monitor
Sennheiser MKE 600	359.23	Microphone: shotgun condenser including mount and pop filter
ASUS VP28UQG 28"	269.00	16:9 4K UHD close mirror monitor
Audient iD4 MKII	216.66	Audio interface with Mic preamp, phantom power, and USB C
Logitech G915	217.74	Wireless keyboard
AVerMedia Live Gamer ULTRA	195.96	Video capture card: 4Kp60 HDR Pass-Through, 4Kp30 Capture Card, Ultra-Low Latency
Apple Magic Trackpad 2	129.00	Wireless trackpad
TOTAL	18205.71	

Table 1: Primary Hardware Components and Specifications

Skill Development and Self-Efficacy for High School Girls via Virtual Hackathons*

Sreedevi Gutta[1], Youwen Ouyang[1], Moses K. Ochanji[2]
[1]Computer Science
[2]School of Education
California State University San Marcos
San Marcos, CA, USA
{sgutta,ouyang,mochanji}@csusm.edu

Abstract

This paper presents a virtual hackathon model for high school girls that was designed to mitigate the barriers to the gender gap in computer science. Through building Apps that solve real-world problems relevant to their communities, participants worked collaboratively to build competency, self-efficacy, and identity with computing. Six virtual hackathons were offered since summer 2020, successfully supporting 167 students, among them, 33% Latinas and 7.2% African Americans. Analyses of post-event survey responses, the Apps submitted by project teams, and judging scores and feedback pointed to students' growth in technical and professional skills, computing identity, and self-efficacy. Participants with little to no prior computing experience successfully demonstrated improved coding skills such as using variables, conditional and iterative statements, functions, and data tables. They also developed software engineering skills in teamwork, project management, and effective communication.

1 Introduction

Despite concerted efforts to increase girls' interest in programming, women remain underrepresented in computing education and careers. In 2020, for

instance, while 57% of bachelor's degree recipients were women, only 22% of bachelor's degrees in computer and information sciences were awarded to women [5]. Similarly, while women account for 57% of professional occupations in the U.S. workforce, only 26% of computing professional occupations are held by women [7]. The problem is especially severe among women from underrepresented minority groups, such as Hispanic and African Americans. In 2020, twelve states had no black females taking AP Computer Science A (CSA) while twenty-nine states had less than 5. Eight states had no Hispanic/Latinx females take AP CSA and twenty-nine states had less than 10 [9]. Among the factors perpetuating gender gaps in computing include gender stereotypes, male-dominated cultures, and fewer role models for girls. A promising approach to improving women's experiences is the implementation of women-only education initiatives [6].

Literature on the underrepresentation of women and minorities in STEM reveals that self-efficacy, identity, and awareness of the social value afforded, are critical in sustaining their pathway to a STEM career [4]. Vicarious experiences from women role models and social persuasion in terms of how STEM can help girls achieve their social goals have been identified as the most influential sources of STEM self-efficacy for girls and women [10]. Rankin and Thomas [8] introduced the term "Intersectional Computing" to characterize the overlapping interplay among the approaches for promoting diversity, inclusion and equity within the field of computing. It shed light on the plurality of intersectional experiences that exist among the broader computing demographic. It is within the framework of intersectional computing [3] that we designed a virtual hackathon experience for high school girls.

Section 2 highlights the structure and main activities of the hackathons. Analyses of data collected from these events are presented in Section 3. The paper concludes in section 4 with a discussion on the broader impact of the intervention on the participants.

2 The Virtual Hackathon

The first virtual hackathon was launched in summer 2020 in response to the vacuum of after-school programs caused by the COVID-19 pandemic-related school closure and social distancing. Five more hackathons were held since then, taking on two formats, one during summer and the other during the school year. Dates and themes for the six events are presented in Table 1.

These events provided a girls-only working environment where girls of diverse ethnic and academic backgrounds were teamed up from multiple high schools. They were provided with challenges to address social issues directly connected to their interests. We chose to use App Lab [1] as our development

Table 1: Hackathon dates and themes

	Dates	Theme
Summer 2020	6/22 - 6/25	Emotional support for teens during the pandemic
Fall 2020	10/17 & 10/24	Helping teens navigate education during the pandemic
Spring 2021	4/17 & 4/24	Saving the Earth one game at a time
Summer 2021	6/28 – 7/1	Helping teens maximize their summer experience
Fall 2021	10/16 & 10/23	Helping people in need during the holidays
Spring 2022	3/5 & 3/12	Promoting awareness of and solutions for climate change issues

platform, as it afforded easy development by participants with no prior programming experience. The invitation to participate specifically stated that no computing skills were required. This was aimed to present computer science as a field that is potentially open and inviting to all women. The girls were supported and mentored by near-peer college computer science women who had been in high school in recent years. The design of the hackathon also enabled the girls to showcase their work and have their work evaluated by professionals (mostly women) from local high-tech industries. This was meant to serve as exposures to both women role models in the computing industry and the breadth of careers in the computing fields. This work builds on previous evidence that programs that have mentors and peers who are reflective of the girls in the program have shown to increase participants' confidence and overall interest in tech-related courses [6].

The summer virtual hackathons ran for four consecutive days Monday to Thursday. Each day, participants were supported with various activities via Zoom from 10:00 AM to 3:00 PM, including team-building activities, skill development workshops, and career awareness sessions. In addition, a 1-hour kickoff was held Monday from 9:00 – 10:00 AM and a 2-hour showcase was held on Thursday from 4:00 – 6:00 PM.

To provide a similar timeline and experience for students, the virtual hackathon during the school year spanned two consecutive Saturdays. The one-hour kickoff was held from 9:00 to 10:00 AM on the first Saturday. Participants were then engaged in project activities until 3:00 PM. They returned the next Saturday at 10:00 AM to continue working on their projects and the event was wrapped with the 2-hour showcase in the late afternoon from 4:00 to 6:00 PM of the second Saturday. In addition, after-school workshops were offered on

Tuesday and Wednesday leading to the first Saturday while after-school office hours were provided Monday through Thursday between the two Saturdays.

3 Data Analyses

A total of 167 girls from 29 schools in Southern California participated in six hackathons. Our targeted effort in recruiting Hispanic and African Americans was successful. Figure 1 compares the ethnicity distribution among participants against the demographics of the region, focusing on the groups underrepresented in computing. The 32.9% participation for Latinas is the highest of all ethnic groups and higher than the 31.3% Hispanic population in the region. The 7.2% participation for African Americans is also higher than the region's 4.7%. Participation from Native American/American Indian and Native Hawaiian & Other Pacific Islander was also higher than their share of the respective populations in the region [2].

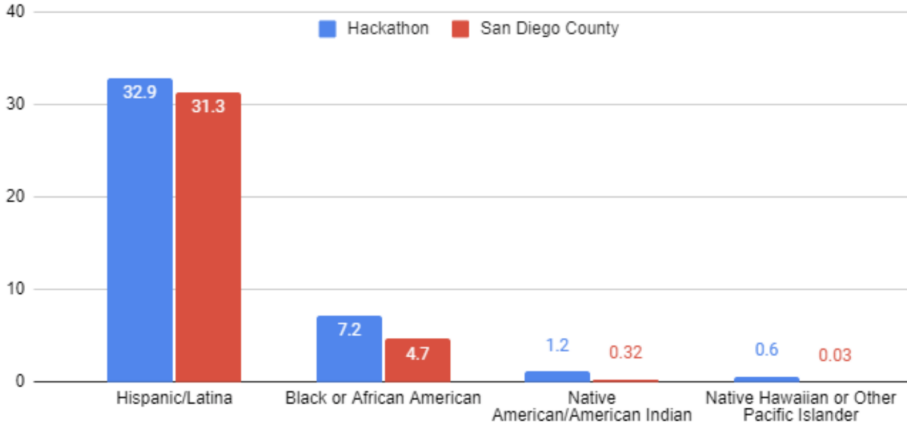

Figure 1: Ethnicity comparison between the hackathons and the region

At the start of each event, a rubric was shared with students to help them understand how their learning would be evaluated at the showcase. The rubric includes seven categories and uses a 4-point scale for each category:

A1: Appropriateness to the theme A2: User experience
A3: Appearance and design A4: Originality and impact
A5: Code complexity A6: Team collaboration A7: Oral presentation

A total of 34 Apps were designed and presented across six hackathons. The Apps were evaluated by tech-professionals from prominent local companies using the above rubric. The average scores provided by the judges in each of

the events were given in Table 2. It was evident that the judges were impressed by the skills showcased by the girls during the event.

Table 2: Average rating given by judges for each hackathon

	A1	A2	A3	A4	A5	A6	A7
Summer 2020	3.65	3.38	3.13	3.35	3.13	3.60	3.30
Fall 2020	3.77	3.28	3.23	3.41	3.13	3.77	3.61
Spring 2021	3.75	3.12	3.50	3.50	3.23	3.58	3.70
Summer 2021	3.54	3.38	3.46	3.54	3.21	3.58	3.63
Fall 2021	3.88	3.31	3.69	3.63	3.63	3.19	3.69
Spring 2022	3.66	3.36	3.32	3.02	3.09	3.53	3.53

Though most of the participants had no prior coding experience, by the end of the event, they were able to incorporate key coding features such as variables, conditional and iterative statements, functions, data tables. These features are illustrated by a representative App created in the Fall 2021 hackathon to address the theme "Helping people in need during the holidays". It utilizes a data table (Figure 2b) to organize items that have been donated to provide gifts for vulnerable people. Figure 2a shows the design of a Gift Cart feature that allows users to browse, add, and check out desired items based on information from the data table. The sample code presented in Figure 2c demonstrates the dynamic update to the data table.

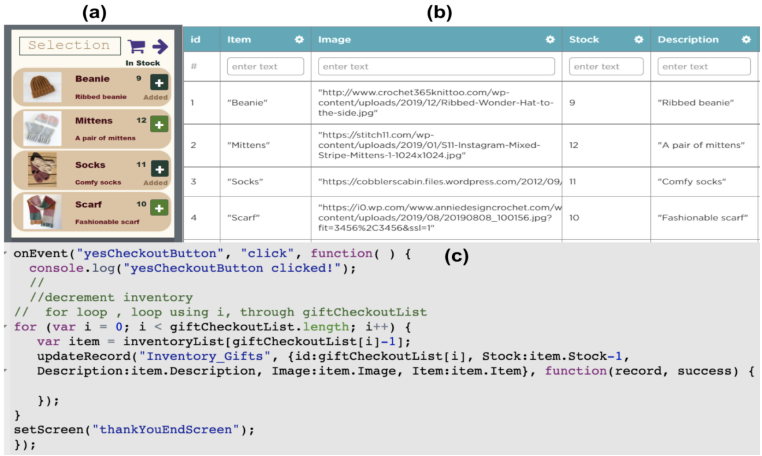

Figure 2: (a) User Interface for a gift cart feature (b) Data table for donated items (c) Event-driven dynamic update.

An anonymous survey was sent to the participants after the events to gauge their self-satisfaction with the program. A total of 86 participants responded to the survey across the six hackathons. Using a Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree), the participants were asked about their growth in the following six categories. The average rating for all the events was reported in Table 3.

B1: Friendships/Networks B2: Programming B3: Communication
B4: Teamwork B5: Problem Solving B6: Leadership Skills

Table 3: Responses for "How much have you benefited from the program"

	B1	B2	B3	B4	B5	B6
Summer 2020	3.92	4.00	4.42	4.25	4.67	4.42
Fall 2020	3.73	3.73	3.92	4.08	4.15	4.00
Spring 2021	3.75	4.25	4.67	4.92	4.50	4.92
Summer 2021	4.38	4.74	4.63	5.00	4.75	4.63
Fall 2021	4.10	4.40	4.20	4.70	4.60	4.20
Spring 2022	4.61	4.17	4.72	4.83	4.67	4.28

The girls assessed their level of satisfaction in the following six categories using a 5-point scale. The average rating reported for each of the events was shown in Table 4.

C1: I felt supported during the program
C2: There was always someone to help me
C3: I felt comfortable asking questions
C4: I learned new things during the process
C5: How satisfied are you with the outcome of your team project?
C6: The hackathon challenge was relevant to me

Table 4: Responses for "How you felt during program?"

	C1	C2	C3	C4	C5	C6
Summer 2020	4.50	4.67	4.33	4.08	4.25	-
Fall 2020	4.46	4.65	4.46	4.31	4.00	4.31
Spring 2021	4.92	4.83	4.75	4.58	4.33	4.67
Summer 2021	5.00	5.00	3.75	5.00	4.75	4.75
Fall 2021	4.80	4.80	4.40	4.60	4.50	4.40
Spring 2022	5.00	4.44	4.67	4.78	4.22	4.56

Girls also reflected on how this event helped them to get excited and pursue their career in computer science. The following quote resonates shared opinions

among participants. "Overall, I had a great experience doing this workshop. Originally, I had little plan to pursue CS in my future, I have a newly gained respect for CS as well as am excited to possibly learn more about coding."

4 Concluding Discussions

In this paper, we have presented a virtual hackathon model for high school girls that was designed to attract a diverse group of girls and engage them in activities that inspire them toward a computing career. While there have been many women's hackathons held in the recent past with similar goals of enhancing young women's participation in computing, what sets this hackathon apart is the consideration factors to enhance female self-efficacy and computer identity development. Additionally, the design aimed to promote inclusion and diversity, as well as to introduce and sustain practices that encourage women to stay in the field.

Starting with a broad range of prior computing experiences, young women were able to gain and articulate a wide range of skills in computing during this dedicated time when working collaboratively with their peers. Even within a virtual context, we were able to create an environment in which the girls felt supported and had a sense of belonging. As a result, the students demonstrated performance in quality technical skills. They reported increased interest in CS and technical competency as a result of these events. Their desire to return to and invite friends to future programming events is an indication of their growing sense of belonging to and identifying with computer science. The girls showed evidence of growth in self-efficacy through their developing abilities in programming skills and their ability to create desired Apps that not only showed their skill development but also the ability to address social issues that were pertinent to the girls themselves. Our project contributes to the evidence that it is possible to recruit and attract a diverse group of young women to computer science.

5 ACKNOWLEDGMENTS

Special thanks to the participants, mentors, and judges for the virtual hackathons. This material is based upon work supported by the National Science Foundation under Grant No. 1615255. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] `https://studio.code.org/users/sign_in` [Accessed 18 August 2022].

[2] `https://datausa.io/profile/geo/san-diego-ca` [Accessed 18 August 2022].

[3] Khalia M Braswell et al. "Pivoting during a pandemic: Designing a virtual summer camp to increase confidence of black and latina girls". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 2021, pp. 686–691.

[4] Martin M Chemers et al. "The role of efficacy and identity in science career commitment among underrepresented minority students". In: *Journal of Social Issues* 67.3 (2011), pp. 469–491.

[5] Scott A Ginder, Janice E Kelly-Reid, and Farrah B Mann. "2015-16 Integrated Postsecondary Education Data System (IPEDS): Methodology Report. NCES 2016-111." In: *National Center for Education Statistics* (2020).

[6] Shahnaz Kamberi. "Enticing women to computer science with es (Expose, engage, encourage, empower)". In: *2017 IEEE Women in Engineering (WIE) Forum USA East*. IEEE. 2017, pp. 1–5.

[7] Department of Labor Bureau of Labor Statistics. *Employed and Experienced Unemployed Persons by Detailed Occupation, Sex, Race, and Hispanic or Latino Ethnicity, Annual Average 2021*. Unpublished table from Current Population Survey, 2021.

[8] Yolanda A Rankin and Jakita O Thomas. "The intersectional experiences of Black women in computing". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 2020, pp. 199–205.

[9] I. Walton. *A Deeper Look at Women of Color (Black, Hispanic/Latinx, and Native American) and AP CS in 2020*. `https://cs4all.home.blog/.` [Accessed 17 August 2022].

[10] Amy L Zeldin and Frank Pajares. "Against the odds: Self-efficacy beliefs of women in mathematical, scientific, and technological careers". In: *American educational research journal* 37.1 (2000), pp. 215–246.

2023 CCSC Southwestern Conference Committee

Michael Shindler, Conference Chair ..... University of California, Irvine
Megan Thomas, Papers Chair ..... California State University, Stanislaus
Mariam Salloum, Authors Chair ..... University of California, Riverside
Michael Shindler, Speakers Chair ..... University of California, Irvine
Todd Gibson, Posters Chair ..... California State University, Chico
Joshua Gross, Panels/Tutorials Chair .. California State University, Monterey Bay
Bryan Dixon, Lightning Talk Chair ..... California State University, Chico
Michael Shindler, Partner's Chair ..... University of California, Irvine

Regional Board — 2023 CCSC Southwestern Region

Michael Doherty, Treasurer/Registrar ..... University of the Pacific
Bryan Dixon, Regional Representative .... California State University, Chico
Megan Thomas, Webmaster ..... California State University, Stanislaus
Michael Doherty, Past Region Chair ..... University of the Pacific
Michael Shindler, Past Conference Chair ..... University of California, Irvine