The Journal of Computing Sciences in Colleges

Papers of the 24th Annual CCSC Northwestern Conference

November 4-5, 2022
Portland Community College – Sylvania Campus
Portland, OR

Baochuan Lu, Editor
Southwest Baptist University

Sharon Tuttle, Regional Editor
Cal Poly Humboldt

Volume 38, Number 1 **November 2022**

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:

Chris Healy, President (2024), chris.healy@furman.edu, Computer Science Department, 3300 Poinsett Highway Greenville, SC 29613.

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

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

Cathy Bareiss, Membership Secretary (2025), cathy.bareiss@betheluniversity.edu, Department of Mathematical Engineering Sciences, 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, Park University - Department of Computer Science and Information Systems, 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

Welcome to the 2022 CCSC Northwestern Conference Portland Community College

On behalf of the Conference Steering Committee and Northwest Regional Board, I welcome you to the Twenty Second Annual Consortium for Computing Sciences (CCSC) Northwestern Regional Conference, held at the Portland Community College, Sylvania campus. We are very excited to be back on campus and meet you all in person again this year.

Many individuals and groups helped coordinate and support this year's conference and I want to thank them for all their time and effort. This year's conference includes 10 papers, 8 panels/tutorials, and the keynote. All papers, panels, and tutorials went through the regular peer review process. The steering committee accepted 9 out of 17 papers (53% acceptance rate). We had colleagues across the region serve as professional reviewers and we recognize their generous efforts in providing time and guidance in the selection of our conference program.

Our Keynote speaker, Peter-Michael Osera, Associate Professor and co-chair of the Department of Computer Science at Grinnell College will talk about why and how we teach Computer Science more intentionally.

Of course, none of this would be possible without the support of our national and local partners. A final thank you goes out to you the attendees whose participation is essential not only to the continuance of conferences such as this, but also for the continued communication and collegiality you provide between all of us involved in the advancement and promotion of our discipline.

We are excited to invite you to our campus, and hope you enjoy the conference and the chance to interact with your colleagues at our annual gathering.

Gayathri Iyer
Portland Community College
CCSC-NW 2022 Conference Chair

2022 CCSC Northwestern Conference Steering Committee

Gayathri Iyer, Conference Chair/Site Chair .... Portland Community College
Bob Lewis, Program Chair ..... Washintgon State Universtity, Tri-Cities
Razvan Mezei, Papers Chair ..... Saint Martin's University
Gina Sprint, Panels & Tutorials ..... Gonzaga University
Ben Tribelhorn, Partners Chair ..... University of Portland
Richard Weiss, Student Posters Chair ..... The Evergreen State College

Regional Board — 2022 CCSC Northwestern Region

Shereen Khoja, Regional Representative ..... Pacific University
Registrar, Registrar ..... Lewis Clark University
Dan Ford, Treasurer ..... Linfield College
Sharon Tuttle, Editor ..... Cal Poly Humboldt University
Mario Guimaraes, Past Conf. Chair ..... Saint Martin's University
Gayathri Iyer, Next Conf. Chair ..... Portland Community College
Razvan Mezei, Webmaster ..... Saint Martin's University

Automation to Soul: Reflections on Tooling Support for Mathematics and Programming Education*

Keynote

Peter-Michael Osera
Associate Professor of Computer Science
Grinnell College
Grinnell, IA 50112

We recognize that quality education is challenging to deliver at scale. Technology has democratized education to a certain degree, e.g., connectivity and online education, machine learning and intelligent tutoring systems. However, these technological solutions have always lacked a "human" element that makes the educational experience special. Such educational experiences are, ultimately, personal; we meet students where they are and guide them to the next step on their educational journey in a challenging yet humane manner.

In this talk, I reflect on the lessons I have learned pursuing the dream of educational tools from the world of programming languages and systems. In particular, in designing tools that remove some of the barriers to mathematics education in computer science, I have been forced to confront, perhaps, the "soul" of our educational endeavors: do undergraduate computer scientists really need a deep understanding of mathematics to be successful? Have we set the question: how does mathematical reasoning relate to computational thinking? Do our assessment methods set students up for success when engaging with mathematics? While I have no definitive answers, I hope to begin a necessary conversation about these foundational questions as we reflect on our instruction in a post-COVID era.

Peter-Michael Osera is Associate Professor of Computer Science at Grinnell College. Peter-Michael's mission is to help people harness the power of computation in its many forms, in particular, through computer programming. He does this by working at the intersection of programming languages and systems, human-computing interaction, and computer science education. Peter-Michael

currently investigates new foundations for constraint-based program synthesis and how we can use this technology to build tools that fundamentally change the relationship developers have with their tools. He received his B.S. in Computer Science and Applied and Computational Math Sciences and B.A. in the Comparative History of Ideas from the University of Washington and received his Ph.D. in Computer Science from the University of Pennsylvania in 2015.

Alumni as Adjunct Faculty and Mentors for Computer Science I Students*

Tammy VanDeGrift
Shiley School of Engineering
University of Portland
Portland, OR 97203
vandegri@up.edu

Abstract

As computer science grows in popularity, more faculty are needed to staff computer science courses. Approval for full-time tenure-track faculty and permanent instructor positions can lag student enrollment demands. One common strategy to meet student enrollment needs is to expand sections sizes. Another strategy is to staff courses with adjunct faculty. This paper describes the experiences of alumni serving as adjunct faculty members to teach introductory computer science labs. These alumni learned introductory programming by completing these labs as students, and then served as role models for computer science majors by providing professional and career advice. The paper provides an overview of the introductory computing labs, ``professions´´ talks the alumni gave each week, and perspectives from the alumni after teaching the labs.

1 Introduction and Related Work

Computer science education prepares students to enter many sectors in the technology, engineering, and science industries. The job outlook is strong in the United States [6, 12, 15]. Collectively, colleges and universities provide many of the educational pathways to these careers, so increasing undergraduate

enrollment in computer science over the past ten years is not surprising. Over half of US high schools now offer computer science at the high school level [4], so students have the opportunity to try computer science before starting college. All of these factors lead to increased demand for computer science courses at the college level [1, 5]. Not only has CS major enrollment increased, the number of students who want programming skills has increased, creating more demand for introductory CS courses. Even though the student demand has risen, the number of tenure-track faculty per unit has not grown at the same rate, according to the Computing Research report Figure B.4 [5]. Departments have handled this mismatch by teaching larger classes, hiring visiting faculty, hiring adjuncts, and using more graduate students as teaching assistants.

The University of Portland has managed enrollment growth by increasing section sizes and hiring adjunct faculty to teach CS elective courses and CS labs. Creating larger section sizes adds extra load to faculty, and the University prioritizes small class sizes. Hiring adjuncts is preferred and can even produce positive learning experiences. A study from 2004 showed that computer science adjuncts helped students pass future courses and increased interest in the major [3]. Engineering programs have also utilized industry adjuncts [2, 7, 9]. A recent panel at a Consortium in Computing Sciences discussed using industry adjuncts in a tech-in-residence model [10]. The CS program did not create a full tech-in-residence model, but adjunct faculty were encouraged to provide feedback about the labs. Another model is an industry fellow program where a faculty member and industry professional collaborate in co-designing and teaching a course [18]. This paper summarizes the experiences of alumni as adjuncts in the introductory programming lab and workshop. Also, starting in Fall 2021, the alumni were asked to share about their professional pathway, their careers, and advice they have for early-college students.

Following the introduction, the institution and introductory computer science courses are described in more detail to provide more context. The remaining sections provide information about the study, the alumni participants, results, and a discussion. This paper focuses on the adjuncts' experience. Due to paper length guidelines, a separate paper reports on the student experience.

2 Institutional and CS 1 Context

The University of Portland is a comprehensive, private, Catholic university on the west coast of the United States serving about 3500 undergraduate students, of which 160 are computer science majors. There are no CS graduate students to teach courses and labs.

CS majors take a three-credit Introduction to CS course (CS 1) and a one-credit companion lab (CS1 Lab) as the entry point to the BSCS degree. Most

CS majors take the CS1 course/lab during the fall semester of their first year. However, there are a few common exceptions: i) students who switch to the CS major after they start at the University take it either fall or spring, and ii) students who get a passing grade on the CS AP exam and/or earn college credit for a CS 1 course start in an advanced CS course. The CS1 course and lab are also required of math, physics, and electrical engineering majors.

CS 1: CS 1 uses the Java programming language with typical introductory language constructs. The course meets three times per week: 55-minutes on Mondays, Wednesdays, and Fridays. The CS 1 class meetings typically include short lecture overviews, code demonstrations, and in-class practice activities.

CS 1 Lab: The CS 1 lab is a one-credit co-requisite for the CS 1 course. The lab meets for three hours per week (one three-hour block or two 1.5-hour blocks). There is little time devoted for lecturing during the lab. Students typically work through lab exercises in pairs or groups of three, using the pair programming model [8]. The labs are designed to be completed and checked by the instructor during the lab period. If students do not finish, they can submit work during the next week. Here are the lab topics schedule by week: 1. Intro to Java and BlueJ, 2. Strings, 3. Graphics, 4. Selection, 5. Iteration, 6. Arrays, 7. Arrays continued, 8. Methods, constructors, classes, 9. Debugger, 10. Inheritance, 11. Libraries, 12. ArrayLists and Files, 13. ArrayLists and Files continued, 14. Review, 15. Coding Exam.

Opt-In CS 1 Workshop: The CS program offers an optional one-credit pass/no pass workshop to give structured and supported individual programming practice time for students. Students with no programming experience and students who get poor scores on the CS 1 quizzes are encouraged, but not required, to take this workshop. The weekly exercises are different than those given in the CS 1 lab, but the topics are similar.

2.1 Course Delivery in Fall 2021 and Spring 2022

During the covid-19 pandemic, instructional modes varied. CS 1 was taught by a full-time faculty member in person in Fall 2021 and Spring 2022. The CS faculty were unsure if students could sit near each other to pair program and follow social distancing protocols when course scheduling was completed in March 2021. Thus, the CS 1 lab and CS 1 workshop sections were taught entirely online using Microsoft Teams in Fall 2021. Eight alumni served as adjunct faculty for the CS 1 lab and one alumnus taught the CS 1 workshop; section sizes ranged from five to nine students for lab and workshop. No student teaching assistants were hired for the CS 1 lab in the fall since each faculty member had at most nine students. Additionally, one full-time CS faculty member served as the CS 1 lab coordinator in Fall 2021: he did not teach sections but he managed the code repository (starter code and solutions), posted

materials to the course management site, and managed the grading spreadsheets. He created short video explanations for each lab to provide consistency across the lab sections and students watched these prior to attending the lab sessions. Using video, rather than live, explanations provided about 15 minutes of lab time for professions talks (see below). The coordinator answered questions from adjuncts – technical questions about course outcomes and lab exercises, university policies such as midterm grades and attendance, and how to engage with students. Because videos introduced the lab exercises, there were no formal teaching presentations during the lab sections. The lab exercises were provided to the adjuncts, and the adjuncts were asked to complete the labs themselves prior to each session. During the lab session, the adjunct faculty rotated through breakout rooms (in the fall) and in-person groups (in the spring) to see how students were doing and to review their work.

In Spring 2022, all courses and labs were taught in person, except for the first two weeks during the omicron spike. One section of CS 1 lab and one section of CS 1 workshop were offered in the spring. Enrollment in the CS 1 lab was 27 students, with one adjunct instructor and two undergraduate teaching assistants. Enrollment in the CS 1 workshop was four students.

2.2 Professions and Life Talks

Because students watched short videos to prepare for lab each week, the alumni replaced these 15-minute blocks per week with career and life advice talks. It is common for faculty in the CS program to invite guest speakers, often alumni, to speak in the capstone course and elective courses. In this context, alumni shared with CS 1 students early in their college education to help them make the most of college and to prepare for their lives and careers. Figure 1 shows the weekly profession talk topics the paper author provided to the adjunct faculty. Social cognitive career theory provided the foundation for developing these topics [13, 14]. The original theory has three aspects: 1) how academic and career interests develop, 2) how educational choices are made, and 3) how academic and career success is obtained. The professions talks were designed so alumni spoke about all three of these items during the semester. No slides were expected – these talks could be more informal for students. Because the course met online in Fall 2021, the alumni recorded this part of the class session; videos of alumni from all sections were posted to the common course site, so students could learn from eight different software professionals.

3 Evaluation

All adjunct faculty were invited to complete an IRB-approved survey at the end of the semester to help answer the following research questions:

Wk	Topic	Prompts
1	Introduction	Short biography and timeline: Who am I? When did I attend University of Portland? Why did I major in computer science? Where have I worked or gone to school? Where do I work or go to school now? What is my favorite thing about computer science?
2	Day at work	What does a typical day of work or a week of work look like for me? What are the many responsibilities? (Give students perspective that they may not be coding all day long as a software engineer.)
3	Teamwork	With whom am I expected to work? What are my strategies for effective teamwork and communication? What advice do I have for you in this lab when pair programming?
4	Debugging	Describe a bug that I have encountered, how I found it, and how I resolved it. What advice do I have regarding debugging software in CS 1?
5	Favorite project	Describe my favorite work or hobby project. What made it fun? What made it challenging? What made it meaningful? What made it memorable?
6	Being resilient	This is usually a tough point in the semester for students. Describe a time where I had to be resilient (could be work or life) and overcome a challenge or setback.
7	Office tour	If I work in an office, show the students around with the camera. If I do not work in an office, describe what working remotely is like.
8	Advice in college	This is usually the week that students are choosing courses for the next semester. Why did I major in computer science? What are some opportunities (electives, clubs, service, study abroad, work) I did as a student? What advice do I have about selecting CS and non-CS courses? How did the core curriculum give me breadth of knowledge/perspective?
9	Getting help	What advice do I have for getting help and finding mentors? Who has helped me as a student and professional? Asking for help is part of the process – try to normalize this for the students.
10	First job and resume	How did I find my first internship or first position? What advice do I have about building a resume? (If so inclined, the instructor could invite students to submit their resumes and provide feedback about them.)
11	Work/life integration	How do I integrate work with life? How do I create and maintain balance between professional and personal goals? What do I do to keep myself refreshed outside of my job?
12	Interviewing	What is my advice about interviewing for internships and jobs? What types of questions have I been asked or have asked? What advice do I have for preparing for interviews?
13	Student choice	Give students time to ask the instructor what they want to know about the profession.
14	Gratitude and care	For what am I grateful in my work and life? How do I show gratitude? How can you show gratitude and appreciation toward your peers, classmates, mentors, supervisors, and faculty?

Figure 1: Profession and Life Talk Topics

1. Was being an adjunct a rewarding experience?
2. How did teaching online work or not work (or teaching in person for spring)?
3. How did students engage with the adjuncts, especially regarding professional development?

Survey: Survey questions included rating questions (strongly agree to strongly disagree) and free-text-response questions. The survey questions can be found in Appendix A. Note that the survey for the workshop adjunct was altered to use ``workshop´´ instead of ``lab´´. The rating questions were designed to force agreement or disagreement – there was no neutral rating; however, there was a not-applicable option for each statement.

Participants: Eight alumni taught the CS 1 lab and one alumnus taught the CS 1 workshop in the Fall. Demographic information about the alumni can be found below (subject ID, grad year, gender, prior adjunct experience).

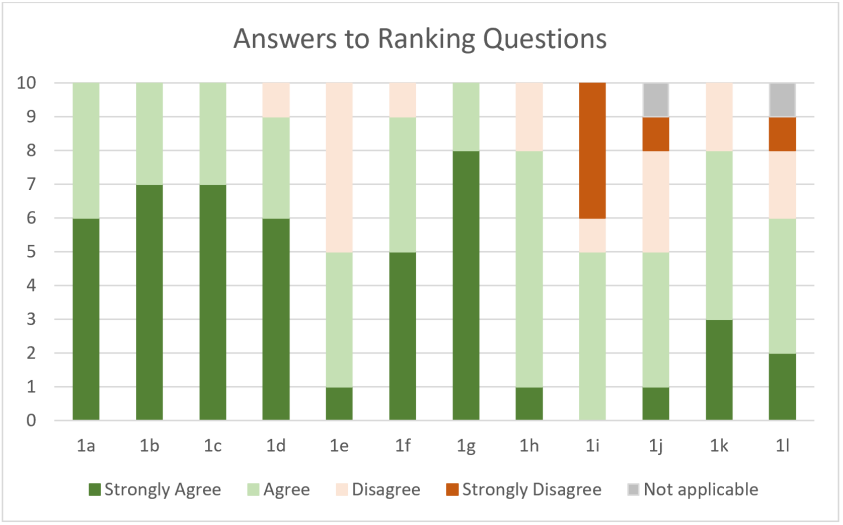
- CS1W1, 2012, Male, Yes; also taught in spring
- CS1L1, 2008, Male, Yes
- CS1L2, 2019, Female, No
- CS1L3, 2020, Male, No
- CS1L4, 2018, Female, No
- CS1L5, 2007, Male, No
- CS1L6, 2019, Male, No
- CS1L7, 2019, Female, No; also taught in spring
- CS1L8, 2020, Female, No

4 Analysis and Results

Figure 2 shows the distribution of ratings for ten surveys (nine from fall and one from spring). responses were coded into emergent themes [17]. Overall, adjunct faculty had a positive experience. Their perception of the student experience was more varied. The free-text

Q1: Rewarding Experience: Was the experience rewarding for the alumni to serve as adjunct faculty members? Yes. Based on the survey responses for question 1a, 100% of the alumni would be an adjunct again for the CS 1 lab or workshop. Additionally, all agreed that they enjoyed being an adjunct (question 1c) and enjoyed working with students (question 1g).

The free-text answers for question 7 (most meaningful about experience) resulted in the following themes: i) appreciate opportunity to try teaching, ii) interacting with students, iii) explaining ideas to others, iv) helping students, and v) reflecting on own development and growth as a programmer. The most common theme was getting to interact with students and watching them



Responses to Rating Questions on Survey, green hue is agreement, orange hue is disagreement, and grey hue is not applicable. [a: would do again; b: prepared; c: enjoyed being adjunct; d: online worked well for me; e: online worked well for students; f: prefer in person; g: enjoyed students; h: students utilized lab time; i: students utilized office hours; j: students worked well together; k: students communicated well with me; l: students communicated well with each other]

Figure 2: Survey Results By Question

grow. For example, CS1L2 stated, ``I enjoyed working with the students and watching them have the aha! moments when they figured out course material. It was great to watch them grow as programmers.´´ CS1L4 stated, ``the things I think were most meaningful were the times I was able to explain something to the students (how to use points to draw a shape, how to use a loop to solve a problem, etc.), and see that they really got it.´´ Others also commented on seeing students reach pivotal moments as a programmer.

Q2: Teaching Online vs In Person: The covid pandemic forced online learning for much of the world. Perhaps teaching some CS courses and labs online permanently makes sense (greater flexibility for students and hiring adjuncts). Did teaching online work or not work for the adjunct faculty? Based on the ratings for question 1d, most alumni felt that teaching online worked well for them. However, responses to question 1e show that they were split (5 agreed and 5 disagreed) about online learning working well for students. Almost all stated that they would prefer teaching in person versus online.

Free-text comments about ratings showed that online teaching had mixed reviews. CS1L1 stated, ``I got the sense that not everyone loved online classes. Also, I didn't monitor the groups at all times, so I don't know if there was a ton of collaboration going on between the team members.´´ CS1L2 stated, ``Often times I found that the students would not work in their groups or work on the lab during class time at all.´´ CS1L3 stated, ``teaching online was very convenient considering I have another job, and did not have to commute to the University, but I feel like I would be more effective at helping the students if I were in person.´´ CS1L5 stated, ``I felt like online made communication more difficult because students didn't have their cameras on, or they were doing the lab from noisy locations or from the library where they said they couldn't talk.´´ CS1L7 thought their section was more interactive and kept cameras on, but noticed that the small section size limited the group pairings. Respondent CS1L8 saw both positives and negatives of online: ``The amount of collaboration in the 'groups' was definitely less online than in person, so general communication did probably suffer from the nature of the course. Playing around with the way class was held did help with this (breakout rooms in teams where I rotated every 10-15 minutes through the rooms worked best). However, it wasn't all bad: I definitely felt like some students coded a lot more, and a lot more independently, than they would have in an in person environment.´´

CS1L7 taught in fall and spring and being in person was much more effective: ``Being in person was huge. I saw so much more engagement once we began in person classes this semester. I experimented with longer term groups instead of changing weekly. Some individuals really thrived in this environment; others I think would've done better working randomly each week.´´

Q3: Student Engagement with Career and Life Advice: Finally, how did students engage with the alumni, especially during the professions talks? The alumni enjoyed giving the professions talks each week, but it was difficult to get student participation in some sections:

- I enjoyed it a lot, but it was hard to get the students to engage. I rarely got any questions from the students. [CS1L1]
- I enjoyed giving the talks, however most students didn't ask questions during the talk period. If someone had a question about the talk topic, they asked it either to me after the lab period or over chat after. [CS1L2]
- I felt like the student would just tune me out. [CS1L3]
- The topics are things I would've wanted to hear about when I was a freshman, so I thought the topics were good. I'm not sure how valuable my students found it though. I hope it was valuable for them, but I'm not sure because they were all very quiet and no one really asked questions afterwards. [CS1L4]
- This was one of my favorite parts of the class. When I was a student, I

definitely did not know what it was like to work as a software engineer, so I hope I was able to give students some insight into what it is like. [CS1L5]

- Its something that I do for my interns all the time. I am big on setting people up for future success. I tell my interns we are first going to build the toolbox, then start filling it up so they are prepared for their future career. [CS1L6]
- I really enjoyed giving the talks to the students each week – although I did enjoy some topics more than others. With being online, it was difficult to tell how much the students appreciated my insights and advice. I'm very interested to see the course survey responses so I can get a better idea of how useful they were. I think because this class is mostly first years and a couple sophomores, they aren't actively thinking about internships and networking yet. For example, I offered to give notes on resumes, and no one sent me their resume. I enjoyed them a lot more in the Fall when most the students were CS majors. I found myself not wanting to go into as much detail as I was worried they would find some of the topics irrelevant. The first few weeks I got a lot of blank stares during this portion of my lecture, so these talks got shorter as the semester went on. [CS1L7]
- Engagement was low, and so I'm not sure how valuable students found the time. The topics were topics I enjoy talking about, so I didn't mind speaking at all, but it's hard to say how useful it was to the students. [CS1L8]
- I felt comfortable sharing these topics with students. I tried to give a unique perspective given the challenges I faced in my 20s. [CS1W1]

The students asked about the following topics: questions about their jobs, internships, preparing for interviews, resumes, development environments at work, programming languages at work, why they chose CS, choosing CS courses, research opportunities, and preparing for the next CS course. Alumni felt that different talk topics were most meaningful, with some providing more than one topic to question 4 on the survey: Resumes and internships (4), Interviewing (3), Debugging (2), and the following each had one response: Typical work day, Life/career integration, College advice: keeping healthy mental state, College advice: studying abroad and clubs, Gratitude, First job.

5 Discussion, Limitations, and Conclusions

As an under-staffed CS program, having alumni who can serve as adjunct faculty is a great and necessary resource. While the program has regularly hired adjunct faculty for the past seven years, Fall 2021 was the first semester

that included the professions talks in CS 1 lab and workshop. The alumni enjoyed giving these talks, but student engagement with career and life advice varied across sections. The online evening class format, combined with small class sizes of five to nine students, may have contributed to the lack of student participation. It may be more intimidating to ask questions over the screen than in person. Student maturity may impact participation – the CS majors were mostly first-year students. The electrical engineering, math, and physics majors were sophomores to seniors and may not be interested in CS careers. Perhaps they just wanted to use the lab time to complete the exercises.

Because labs were taught online in Fall 2021, the CS program could recruit adjuncts in Oregon or Washington (tax laws prohibits recruiting elsewhere). This provided a much larger pool of adjuncts, especially including a large alumni presence in Seattle. The adjunct faculty held software engineering positions for their regular jobs, so the semester stipend of $3333.33 was not the incentive. The adjuncts took this opportunity because they wanted to work with students. Also, all labs and the workshop were scheduled to start at 4:10pm or 7:10pm to accommodate full-time workers' schedules. While the online evening format worked well for adjuncts, it seems that both the alumni and students prefer in person teaching and students probably prefer day-time lab sections. The students choose the University to have residential community, small classes, interactions with peers and faculty, so online courses do not seem to be a great match to the population.

There are limitations to this study and results. The study took place at one institution and contextual factors may not translate to other environments. While many of the alumni were new to the adjunct role, they had high levels of prior engagement with the CS program. The instructional delivery was mostly online and likely contributed to the effectiveness of the professions talks. In the future, the CS program may adapt the professions talks topics and continue to assess the in person effectiveness. Due to staffing shortages, the CS program could not staff some labs with full-time faculty to run a comparative study, so there is no control or comparison group in this study.

Overall, inviting alumni to be adjunct faculty has been a positive experience for the program and the alumni. All agreed they would be an adjunct faculty member again. This seems to be an effective solution for the staffing shortage while providing students access to mentors in the profession. As graduates, the alumni knew the CS curriculum and labs well, and they could answer questions about the current state of the profession that full-time faculty may not be able to answer. Forward-thinking students even asked their adjunct faculty to connect professionally via LinkedIn. Maybe, one day, the adjunct and student will be working together again.

Acknowledgements

This work was supported by a Sweo Faculty Fellowship at the University of Portland. We thank the donor for the opportunity to conduct this research study and the alumni who served as adjuncts.

Appendix A: Survey

Thank you for serving as an adjunct for the computer science program. Your skills, experience, advice, and help are certainly appreciated. Please reflect upon and answer the questions below.

1. For the following statements, put an "x" in one of the boxes to indicate your opinion: strongly agree, agree, disagree, strongly disagree, not applicable.

	Strongly agree	Agree	Disagree	Strongly disagree	Not applicable
a. I would be an adjunct again for CS 1 lab.					
b. I felt prepared to be an adjunct for CS 1 lab.					
c. I enjoyed being an adjunct for CS 1 lab.					
d. Teaching the course online worked well for me.					
e. Teaching the course online worked well for the students.					
f. I prefer teaching in person versus online.					
g. I enjoyed working with the students.					
h. Students utilized lab time productively.					
j. Students utilized office hour time.					
j. Students worked well together in pairs/groups.					
k. Students communicated well with me.					
l. Students communicated well with each other.					

Feel free to add any comments about your responses to question 1:

2. Describe how you felt about giving the college/career/life advice talks each week.
3. What questions did students ask you that were outside the CS 1 Java material?
4. What college/career/life advice did you provide that you think was most meaningful for the students?
5. What surprised you about working with introductory computer science students?
6. How often did you engage with students outside of the scheduled lab? What tools did you use (email, Zoom, Teams, Slack, etc.)?
7. What was most meaningful for you about the adjunct experience?
8. Do you plan to keep in touch with your CS 1 students? If so, how so?
9. Thinking about the overall course, what worked well in CS 1 lab?
10. Thinking about the overall course, what do you suggest for improvements for CS 1 lab?
11. Do you have any comments about specific labs and/or pacing in the labs?
12. Is there anything else you want to share about CS 1 experience?

Figure 3: Survey Questions

References

[1] Association for Computing Machinery and IEEE Computer Society Task Force. *Computing Curricula 2020: Paradigms for Global Computing Education*. https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2020.pdf. 2021.

[2] C.E. Bakal et al. "Industry Adjuncts: Lessons Learned". In: *Proceedings of the ASEE Annual Conference and Exposition* (2011). https://peer.asee.org/industry-adjuncts-lessons-learned.

[3] E. Bettinger and B.T. Long. "Do College Instructors Matter? The Effects of Adjuncts and Graduate Assistants on Students' Interests and Success, NBER Working Paper No. 20370". In: *National Bureau of Economic Research* (Mar. 2004). https://www.nber.org/system/files/working_papers/w10370/w10370.pdf.

[4] Code.org. *2021 State of Computer Science Education*. https://advocacy.code.org/. 2021.

[5] Computing Research Association. *Generation CS: Computer Science Undergraduate Enrollments Surge Since 2006*. https://cra.org/data/Generation-CS/. 2017.

[6] B. Fung. "Tech companies are hiring more liberal arts majors than you think". In: *The Washington Post* (2015). https://www.washingtonpost.com/news/the-switch/wp/2015/08/26/tech-companies-are-hiring-more-liberal-arts-majors-than-you-think/.

[7] T. Gibbons. "Course Guides: A Model for Bringing Professionals into the Classroom". In: *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education* (2012). DOI: https://dl.acm.org/doi/10.1145/2157136.2157172.

[8] B. Hanks et al. "Pair programming in education: a literature review". In: *Computer Science Education* 21.2 (2011). DOI: https://www.tandfonline.com/doi/full/10.1080/08993408.2011.579808.

[9] C.K. Hobbs and H.H. Tsang. "Industry in the Classroom: Equipping Students with Real-World Experience A reflection on the effects on industry partnered projects on computing education". In: *Proceedings of the Western Canadian Conference on Computing Education* (May 2014). DOI: https://doi.org/10.1145/2597959.2597967.

[10] S.P. Imberman et al. “Strategies for maximizing the value of industry adjuncts: the Tech-in-Residence Corps model”. In: *Journal of Computing Sciences in Colleges* 35.8 (2020). DOI: https://dl.acm.org/doi/abs/10.5555/3417639.3417668.

[11] A.J. Ko and K. Davis. “Computing Mentorship in a Software Boomtown: Relationships to Adolescent Interest and Beliefs”. In: *Proceedings of the ACM International Computing Education Research Workshop* (2017). DOI: http://dx.doi.org/10.1145/3105726.3106177.

[12] I. Kowarski. “What Can You Do with a Computer Science Degree?” In: *US News and World REport* (2019). https://www.usnews.com/education/best-graduate-schools/articles/2019-05-02/what-can-you-do-with-a-computer-science-degree.

[13] R.W. Lent and S.D. Brown. “Social cognitive model of career self-management: Toward a unifying view of adaptive career behavior across the life span”. In: *Journal of Counseling Psychology* 60.4 (2013), pp. 557–568. DOI: https://doi.org/10.1037/a0033446.

[14] R.W. Lent, S.D. Brown, and G. Hacket. “Social Cognitive Career Theory”. In: *Career Choice and Development* 4.1 (2002), pp. 255–311.

[15] A. Loten. “America’s Got Talent, Just Not Enough in IT”. In: *The Wall Street Journal* (2019). https://www.wsj.com/articles/americas-got-talent-just-not-enough-in-it-11571168626.

[16] A. Settle, J. Lalor, and T. Steinbach. “Reconsidering the Impact of CS1 on Novice Attitudes”. In: *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education* (2015). DOI: http://dx.doi.org/10.1145/2676723.2677235.

[17] S. Stemler. “An overview of content analysis”. In: *Practical Assessment, Research and Evaluation* 7.17 (2000). DOI: https://doi.org/10.7275/z6fm-2e34.

[18] J. Tenenberg. “Industry fellows: bringing professional practice into the classroom”. In: *Proceedings of the ACM SIGCSE Technical Symposium on Computer Science Education* (2010). DOI: https://doi.org/10.1145/1734263.1734290.

[19] H.M. Walker. “Ways to Help New, Visiting, and Adjunct Faculty”. In: *Journal of Computing Sciences in Colleges* 35.5 (2019). DOI: https://dl.acm.org/doi/10.5555/3381613.3381624.

Automatic Assessment of the Design Quality of Student Python and Java Programs*

J. Walker Orr
Computer and Information Sciences
George Fox University
Newberg, OR 97132
jorr@georgefox.edu

Abstract

Programs are a kind of communication to both computers and people, hence as students are trained to write programs they need to learn to write well-designed, readable code rather than code that simply functions correctly. The difficulty in teaching good design practices that promote readability is the labor intensiveness of assessing student programs. Typically assessing design quality involves a careful reading of student programs in order to give personalized feedback which naturally is time consuming for instructors. We propose a rule-based system that assesses student programs for quality of design of and provides personalized, precise feedback on how to improve their work. To study its effectiveness, we made the system available to students by deploying it online, allowing students to receive feedback and make corrections before turning in their assignments. The students benefited from the system and the rate of design quality flaws dropped 47.84% on average over 4 different assignments, 2 in Python and 2 in Java, in comparison to the previous 2 to 3 years of student submissions.

1 Introduction

Recently there has been increasing interest in intelligent tutoring systems for programming instruction. Primarily the focus has been on assisting students by

giving them helpful hints on how to complete their programming assignments. There has been a lot of work surrounding the *Hour of Code* [2], a massively open online course intended to teach children how to program with a visual programming language. The visual language contain constructs analogous to conditional statements but lacks *loops*. Nevertheless, the project is popular and there has been significant research into producing a intelligent tutors for it. Typically the goal of these systems [7, 3] is to suggest edits to a student's program to get it closer to correct functionality. Another approach create an embedding for student programs that could be used to suggest hints on how to correct their programs [8].

However, becoming a capable and competent programmer involves more than simply writing functional code, solutions must also be readable and understandable, following commonly accepted conventions. Even if a program works correctly when first created, the reality of most software systems is that of continuous maintenance and development. Unnecessary complexity, convoluted logic, and unconventional style will result is mistakes in the ongoing maintenance and development of software. Naturally, a major difficulty of teaching the practice of writing readable, well-design programs is the time-intensiveness of giving quality feedback. Determining if a student's program works correctly can usually be assessed with unit or system integration tests. However, providing feedback on the quality of program design generally speaking requires as close reading of the code which of course is time consuming.

In the professional setting, "linters" are commonly used to check for design errors and violations of programming conventions. For example, Pylint [9] is a commonly used "linter" for the Python programming language which does catch design flaws and bad practices such as the use of global variables. Additionally, there is the *PEP8* standard and eponymous program that checks Python programs for violations of the standards. For Java, tools such as CheckStyle [1] applies configurable rules to ensure adherence to convention and code formatting.

However useful, the needs of an educational setting differ from the professional in a few key ways. First of all, they are intended to be used by professional programmers who are experienced and (hopefully) have a strong sense of good design. This means that these "linters" are primarily checking for a short list of less egregious violations. Typically "linters" check code format and use heuristics to judge code complexity. The notion of "the right tool for the job" is left up to the discretion of the professional. Some design choices such as the use of closures may be appropriate in professional setting but are generally not a good choice for an assignment in an introductory computer science class. Hence a "linter" may rightfully permit some practices and design choices that are simply not appropriate for students. Another example is the

use of recursion. Recursion is elegant tool in some settings but novice students can abuse it, for example, by recursively calling a "main" function to repeat the logic of a program rather than using a *while* loop. A "linter" typically would not even check for the use of recursion, but this bad practice can be common in introductory computer science courses. Students, especially early on, require tighter constraints to guide them to use simple and direct approaches to relatively simple problems. These types of constraints are not generally considered in a professional setting. These kinds of very specific design requirements generally do not occur in a professional setting hence tools such as Pylint, PEP8, or CheckStyle do not have the ability to check for them.

A recent work targeting the professional setting, *DeepCodeReviewer* [4] utilized a database of code reviews to suggest design quality improvements for C# code segments. It is a deep learning model trained on a propriety data taken from Microsoft's software version control system. Likewise, the feedback it generates is also not particularly relevant to education since it is trained on professional's code who likely do not make the same mistakes as first-year computer science students.

Recently there has been some work using neural networks to assess the design quality of student programs [6]. Though that system accurately predicted design scores and gave high-level feedback, there are some limitations both to that system and to any system based on machine learning. Feedback on the program level, though shown to be useful, is not as helpful as feedback targeted at specific segments of the program such as functions or individual lines. More detailed and specified the feedback will presumably be easier for a student to understand and remedy.

1.1 Our Approach and Contribution

Our approach is to take design principles, model them by formally representing the principles as logical rules applicable to an abstract syntax tree. The design principles were in general each expanded into multiple rules prescribing syntactic structures that are either best practices or are patterns that should be avoided. The models were implemented in Haskell [5] since each rule could be easily be stated in a declarative fashion. These model are the first for the design quality of Python & Java programs for an educational environment. The implemented models was then hosted as a web service, allowing students to evaluate their programs at any time and as many times as they wished. Finally, data was collected on the number of design quality errors made by students who had access to the models versus work done by students in previous years. The data indicates a large drop in the rate of design errors made by students.

2 Method

The development of our intelligent tutoring system was a process that started with the articulation of our the principles we use to assess good design. Consolidating both written policy and practices as well as our general intuition about what makes the design of a program good was perhaps the most difficult aspect of creating the system. After these principles were realized and identified, they were contextualized for both Python and Java programs and more formally represented as logical rules for the respective syntactic structures. Next a model was implemented for each language as a program written the functional programming language Haskell. Though there are rules in common, Java and Python are very different syntactically so implementing a model was a more direct and effective approach. Haskell's declarative nature and history of use for analysing programming languages made it a natural choice. Finally the tutoring system was deployed as a web service, allowing students enrolled in our introductory computer science course to use it at anytime to validate their programs before the assignments were due.

2.1 Models

After articulating our design principles of our institution, the principles were contextualized and specified for both the Python and Java programming languages. The model for these principles are rules represented in first order logic. Each rule makes use of predicates that express proprieties of abstract syntax trees (AST). The rules are in effect constraints on the space of ASTs, eliminating those with or without some features.

The purpose of the models is to detect design errors in a student's program. It could be useful to detect segments of a student's program that are well designed in order to encourage students by applauding their success. However, the purpose of this model is to point out mistakes so that they can be remedied by the student. Also our institution's existing principles and written policy on program design focused heavily on avoiding mistakes. Further, the existence of a design mistake is much more objective and readily identifiable than a good design quality.

With this in mind, our models are a collection of rules, applied to an AST, to identify individual mistakes exactly where they occur in a student's program. Each rule in model will generate a mistake to be reported to the student if the condition of the rule is true. The condition i.e. body of each rule consists of a logical combination of predicates. Each predicate is simply a observation of the existence or nonexistence of property of the AST. A predicate is simply a boolean function over "objects" which nodes in the AST which correspond to either statements or expressions in the source code. A statement is an operation

```
def record_score(h_won):
    global human_score
    global comp_score

    if h_won:
        human_score += 1
    else:
        comp_score += 1
```

(a) Global variables example.

```
def find(my_list, value):
    i = 0

    for other in my_list:
        if other == value:
            return i

        i += 1
```

(b) Nested return statement example.

Figure 1: Python code segments inspired by student programs.

of a program, for example the assignment of a variable, the definition of a function, or a "return" from a function. An expression is a computable value, such as an arithmetic operation, the comparison of two values, or the evaluation of a function. Generally speaking, every modern programming language can be broken down into a combination of statements and expressions including Python. Each expression or statement may be composed of more statements or expressions hence forming a sub-tree in the AST. Altogether, in our model a predicate is a boolean function that operates over a sub-tree in the AST. This allows for great expressive power and the ability to detect any characteristic that is grammatically represented in the AST.

Consider the example in Figure 1a which contains code making use of a global variable. Besides the fact that the use of a global variable is widely considered a bad practice, it is also in violation the principle of using explicit logic over implicit logic. A rule to capture this is straight-forward,

$$\forall f, s \; Fun(f) \wedge Desc(f, s) \wedge Global(s) \implies M(f, s) \tag{1}$$

The predicate $Fun(f)$ determines if f is a function, $Desc(f, s)$ ensures that s is a descendent of f in the AST, and $Global(s)$ is true if s is a "global" statement. If all three of these predicates are true, there is a mistake $M(f, s)$ in function f in statement s which is sufficient information to generate a helpful message. The head, that is, the consequent of each rule in the model is simply a mistake predicate, hence no complex inference is necessary. In general, the models' rules follow this pattern, the body of the rule consists of AST predicates and the head of the rule is a type of mistake.

In the previous example the type of mistake could easily be detected with a simple string search, however identifying the function they are found in is more difficult. In general, the mistakes our model identifies requires knowledge

of the AST. Of particular importance is the ancestor-descendent relationship captured by the predicate *Desc*. Consider the example found in Figure 1b which contains a segment of Python code with a nested "return" statement. In this example, context is critical since "return" is legitimate and necessary in general. However in this example, the code violates our principle of one-way-in-one-way-out as well as subtly contains a bug. If the "value" is not contained in the list, the value "None" will implicitly be returned. If the code calling the function always expects an integer return value the an exception will be raised. Likewise this type of mistake is detected in a straight-forward fashion,

$$\forall f, s\ Fun(f) \wedge Desc(f, s) \wedge \neg Child(f, s) \wedge Return(s) \implies M(f, s) \tag{2}$$

The predicate $Child(f, s)$ determines if s is a direct child of f and $Return(s)$ is true if s is a return statement. Here the subtly of the nestled "return" statement is readily captured by the use of the *Child* and *Desc* predicates. A legitimate "return" will be directly under the function declaration in the AST, while a nested "return" will be deeper in the AST. The contrast between $\neg Child$ and *Desc*, that is a "return" statement being a syntactic descendent of a function declaration but not a direct child of it, is exactly the definition of a nested "return".

These two examples point out a key aspect of the model, the primary complex predicates needed are *Desc* and *Child*, the rest of the predicates simply identify the type of expression or statement which is immediately apparent from the AST information.

2.2 Implementation

The models were implemented in the Haskell programming language. While many languages could be a suitable choice, and some such as Prolog are even designed for logical deduction, Haskell has several natural advantages. First, Haskell has support for parsing many common programming languages such as Python, Java, and JavaScript. One benefit of Haskell is that the syntactic elements of Python and Java are expressed directly as Haskell data structures. Functions in Haskell pattern-match on directly on data structures which means functions can easily be written to check specific parts of an AST. For example, this means a function can be written to specifically check for errors in Java method declarations and associated statements. Second, functional programming's declarative style and high-level functions mean the models' rules can be almost directly expressed in the language. The logical rules of the model examine the AST for the existence or nonexistence of certain syntactic elements or identify particular relationships between them. Most of these operations have a corresponding higher-order functions is used to implement them.

Because of differences in the Python and Java languages as well as subsequent differences in parsers, we decided to implement the Java and Python models as separate programs. There are enough similarities that it is possible unify both programs, however it made more sense practically speaking to keep them separate. Haskell is so effective at the task of representing our models, the implementation for Python is only 216 lines long while the Java implementation is 495 lines excluding comments and empty lines.

2.3 Personalized Feedback

The primary purpose of the system is to provide personalized feedback to students rapidly. Our rule based model is fast, evaluating a student program in less than a second on a commodity computer. The model was deployed to a web server and was made available to students with a simple HTML form. There was nothing for the student to setup or install on their own computer, since the tutoring system was available as a web service. Student could very easily use the system by simply submitting their code via the form. The model was run immediately when a student submitted their program, producing feedback on design mistakes. Since the model is fast, there was no need to queue up submissions or rate-limit the service at all.

Students were able to check their assignments for mistakes as many times as they wanted. Further, the system was available 24/7, allowing the students to check their work whenever was convenient before the due date. This meant the students had rapid feedback on their assignments and a chance to improve their work before being evaluated by the instructor. Further, this increased the transparency of the grading process, since the rules are explicitly stated and the mistakes generated by them are easily understood.

3 Experiments

The effectiveness of the system was evaluated on student programs from multiple years of introduction to computer science courses. The system was made available to students during both the first and second introduction to computer science courses. Students were encouraged to use the system to evaluated their programs multiple times before submitting their program for grading. This gave the students an opportunity to correct their mistakes before being graded. The students were also informed that the output of the system was going to be used to a guide to instructors to grade their assignments, which provided incentive to use the system and transparency into the grading processing.

The experimental setup is simple, the Python programs from 2021 and the Java programs from 2022 are from students who had access to the tutoring system. Submissions from previous years were used as a basis for comparison.

The actual assignments, standards, and requirements remained consistent over the years, the only difference between experimental year and previous years is student access to and feedback from our system. The system was used to quantify errors in the experimental and previous years' assignments. This data was used to estimate the rate of mistakes made by students each year. Hence the experiment compares the error rate of students who had access to the system versus students from previous years that did not.

3.1 Dataset

The dataset consists of programs from 4 student assignments collected over 4 years, 2 Python assignments taken from a introduction to computer science course and 2 Java assignments taken from the second semester introduction to computer science course. All the programs included in the dataset were syntactically correct, all student submissions that could not be parsed into an AST simply were not included. The Python assignments were the last 2 assignments of the course and were chosen because they were the most complex and lengthy assignments in the class. The "Craps" assignment involves implementing the classic casino game as a command-line program. Likewise the "RPS" assignment is the game of rock-paper-scissors. The number of student programs totals to 506, with 109 of them being submitted in 2021. Those submissions had our system available for feedback. A summary of the Python data can be found in table 1.

The Java assignments are from the middle of the second semester introduction to computer science course. The "Car" assignment involves creating a class to represent a Car, its fuel, MPG, and odometer as well as methods to make it "drive" etc. Similarly, the "Balloon" assignment involves creating a class to represent a spherical balloon with its radius and has methods to inflate and deflate it with cubic units of air. These methods require a mathematical conversion between radius and cubic units. The total number of student submissions was 298 with 59 of them submitted in 2022. A summary of the Java data can be found in table 1. We did not record the number of times, if any, a student used our system to gain feedback.

3.2 Results

The rate of mistakes made by students significantly dropped with the introduction of our system. For all the assignments, the drop in mistake rate was both statistically and substantial. For the Python assignments, the comparison in mistake rates is between the 2021 programs and all previous years. Similarly, for the Java assignments the comparison in mistake rates was between 2022 and the years 2020 and 2021. The difference between the mistake rates was

Program	Year	Programs	Mistakes	Rate
RPS (Python)	2018	68	311	4.57
	2019	74	285	3.85
	2020	70	231	3.30
	2021	55	145	2.64
Craps (Python)	2018	65	388	6.0
	2019	78	463	5.94
	2020	42	210	5.00
	2021	54	181	3.35
Car (Java)	2020 - 2021	104	138	1.33
	2022	30	8	0.27
Balloon (Java)	2020 - 2021	101	416	4.12
	2022	29	74	2.55

Table 1: The statistics of student Python & Java programs. Included are the number of student program submissions, the total number of design quality mistakes, and a rate of mistakes. Students only had access to the intelligent tutoring system during 2021 for Python and 2022 for Java.

compared with a two-sample Poisson test. For all assignments, the difference was found to be significant with p-values was less than .00002. This is unsurprising considering the drop in the mistake rates was 32.31%, 41.23%, 79.70%, & 38.11% for "RPS", "Craps", "Car", and "Balloon" respectively. The magnitude of the improvements clearly demonstrate the effectiveness of the system. We believe that both the availability of the system along with the fact that it was used to guide assessment and grading provided the motivation to use the system and make corrections. Further, the drop in design quality errors was noticeable for the instructors anecdotally. Also, student appreciated the system as well, since it was fast, convenient, and provided additional transparency into the assessment process.

In the improvements in Java assignments were noticeable for two reasons. First the dramatic nature of the improvement, as much as a 79.70% drop in the rate of mistakes. Second, was the fact that all the style guidelines had been written down specifically for Java and were available to students for years. Unsurprisingly, students apparently did not read or apply the guidelines despite being told that their work would be assessed according to them. However our tutoring system which directly encoded the guidelines, actually got the students to comply. The difference is likely the dynamic nature of the system, giving precise, rapid feedback which is easier to utilize than reading an 8 page PDF of general rules. It is likely far easier for students, especially relatively inexperienced ones, to respond to specific feedback over general statements.

Overall, the system was highly effective and that was only possible because of voluntary student participation.

4 Conclusions & Future Work

Overall, an intelligent tutoring system was designed based to give rapid feedback on program design quality. The model it implemented was derived from a set of design principles which were encoded as rules over abstract syntax trees for both Python and Java. The system was fast, accurate, and was highly effective at delivering feedback to students. As a result, the quality of student code improved substantially, also indicating their voluntary use of the system.

Though successful there are some types of design quality mistakes that the tutoring system does not catch. Mistakes such as bad variable names, large blocks of repeated code, or uninformative comments cannot be identified by the system but would be a useful extension. These kinds of mistakes are hard to typify directly with rules, so a machine learning model or other methods might be needed.

References

[1] CheckStyle. *CheckStyle*. https://checkstyle.sourceforge.io/.

[2] Code.org. *Code.org: Learn computer science*. https://code.org/research.

[3] Aleksandr Efremov, Ahana Ghosh, and Adish Singla. "Zero-shot learning of hint policy via reinforcement learning and program synthesis". In: *International Educational Data Mining Society* (2020).

[4] Anshul Gupta and Neel Sundaresan. "Intelligent code reviews using deep learning". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day*. 2018.

[5] Simon Peyton Jones. *Haskell 98 language and libraries: the revised report*. Cambridge University Press, 2003.

[6] J. Walker Orr and Nathaniel Russell. "Automatic Assessment of the Design Quality of Python Programs with Personalized Feedback". In: *International Educational Data Mining Society* (2021).

[7] Benjamin Paassen et al. "The continuous hint factory-providing hints in vast and sparsely populated edit distance spaces". In: *Journal of Educational Data Mining* (2018).

[8] Chris Piech et al. "Learning program embeddings to propagate feedback on student code". In: *International conference on machine Learning*. PMLR. 2015, pp. 1093–1102.

[9] Pylint.org. *Pylint - code analysis for Python*. https://pylint.org.

Github Copilot in the Classroom: Learning to Code with AI Assistance*

Ben Puryear and Gina Sprint
Department of Computer Science
Gonzaga University
Spokane, WA 99258
bpuryear@zagmail.gonzaga.edu, sprint@gonzaga.edu

Abstract

Recent advances in deep machine learning have enabled artificial intelligence-driven development environments (AIDEs). AIDEs are programming tools that, given comments or starter code, can generate code solution suggestions. As the accuracy of these tools continues to increase, one particular AIDE from Github, Copilot, has been gaining significant attention for its performance and ease of use. The rise of Copilot suggests that code solution generation tools will soon be commonplace in both the industry and in computer science courses, with expert and novice programmers alike benefiting from using these tools. More specifically for novices, the effects of Copilot on the process of learning to code are mostly unknown. In this paper, we perform initial explorations into these effects. Using introductory computer science and data science courses, we evaluate Copilot-generated programming assignment solutions for correctness, style, skill level appropriateness, grade scores, and potential plagiarism. Our findings indicate Copilot generates mostly unique code that can solve introductory assignments with human-graded scores ranging from 68% to 95%. Based on these results, we provide recommendations for educators to help adapt their courses to incorporate new AIDE-based programming workflows.

1 Introduction

In 1967, British scholar Christopher Longuet-Higgins theorized about a "Computer-Assisted Typewriter" that would improve a programmer's efficiency by automatically completing words after only a few characters had been typed. Over the years, this idea has evolved into the most frequently used programming assistance tool, "code completion" [1]. In addition to completing single words, intelligent code completion tools, such as Microsoft's IntelliSense, can suggest large code blocks to help speed up common coding tasks like writing constructors or getters/setters. Recently, advancements in deep machine learning have enabled the next generation of code completion tools, namely artificial intelligence-driven development environments (AIDEs). AIDEs are programming tools, like code completion, that generate suggested code snippets based on a description of what code should do, such as a comment or the start of a function definition. Announced in July of 2021, Github Copilot is one such AIDE that is already gaining widespread popularity, with some researchers predicting it will become mainstream in as few as two years [6]. Copilot is based on OpenAI's Codex language model and was trained on 159 Gb of data from public repositories on Github, allowing it to generate code in dozens of different languages, with particular success demonstrated in Python, JavaScript, TypeScript, Ruby, Java, and Go. [3]. In the one year that Copilot has been available as a "technical preview" extension for Visual Studio Code, Neovim, and JetBrains, at least 1.2 million programmers have played with the tool. In that short time, over twenty research papers have been published investigating its performance and copyright concerns related to code authorship.

Because Copilot is easy to use and can generate solutions for simple programming problems, it will redefine how people learn to code and how educators teach introductory computer science courses. In fact, Ernst and Bavota stated "The nature of learning programming will change dramatically with AIDEs. Whether these assistants will speed up or slow down the learning process is currently an open question" [5]. The authors further speculated that Copilot might already be able to solve CS1 assignments better than the students enrolled in the course. To solve assignments, students can provide Copilot snippets from the assignment specifications and accept Copilot's suggested solutions. In this case, students may receive passing grades on programming assignments without learning how to solve the assignment on their own. Though, what happens when a Copilot-dependent programmer eventually runs into a slightly more complicated problem that AIDEs cannot (yet) solve? To help mitigate this scenario, educators should become familiar with AIDE workflows and how introductory students use them while learning to program. This paper provides an overview of this process and how educators can begin to adjust courses to teach skills that include using AIDEs.

2 Related work

AIDEs, like Github Copilot (copilot.github.com), TabNine (tabnine.com), and Kite (kite.com), are products of recent advances in deep machine learning. As the number of parameters in these deep learning models increases, so too does the accuracy of these tools. For example, one set of experiments showed Codex's solve rate on a benchmark dataset jumped from 59.5% to 72.3% when the number of model parameters increased from 2.5 billion to 12 billion [3]. Recent research has shown that we can expect AIDEs to permeate the traditional programming workflow because they offer many benefits for users, such as automating mundane tasks [5], increasing perceived productivity [6, 9, 10, 11], and helping with learning to program or learning a new language [6, 5]. Despite these advantages, AIDEs are not a replacement for a human programmer with relevant knowledge and skills. This is because there are several known issues associated with using AIDEs, including the potential for low-quality code, errors, vulnerabilities, or bias; a compromise in efficiency, documentation, and adherence to style guidelines; and a weakening of problem solving skills due to over-reliance [6, 5, 2, 8, 3, 7]. Additionally, speculative drawbacks include legal issues related to software authorship, copyright, and licensing [4, 8]; how the programmer-computer interaction and workflow will change [5]; and variability in performance for different underrepresented user groups [11].

The aforementioned benefits and shortcomings of AIDEs apply to novice programmers and computer science students in introductory courses who use the tool as well [5, 6]. AIDEs can assist students with completing code for programming assignments, side projects, and technical coding challenges for job interviews. For example, typically students (and experts) leave their editor to search Google or StackOverflow for how to solve a problem. With AIDEs, these results can be curated and presented within the editor and in the context of the specific problem. Such a use case will likely help students learn to program because the AIDE is like a partner in pair programming or a chatbot providing code templates for students to tinker with [11]. As AIDEs continue to improve, these code templates will likely solve entire assignments. Preliminary experimentation suggests that Codex-based AIDEs can already generate passing solutions to CS1 programming assignments [5]. If a student can paste the requirements of an assignment in an AIDE and produce a solution with little to no problem solving, code writing, or debugging, what are students learning? From an educational perspective, perhaps the most important question is: in an AIDE-prevalent future, what *should* computer science students be learning? In this paper, we present the first research in this area that aims to investigate the effects of AIDEs on learning to program in introductory courses.

3 Methods

Our investigation was motivated by our desire to become familiar with Github Copilot, evaluate its code suggestions, and understand how students can use it when learning to code. More specifically, we were interested in answering the following three research questions:

1. Could Copilot be a viable tool for introductory programming students?
2. Could Copilot solutions earn a passing grade score?
3. Could Copilot solutions be flagged as plagiarism?

To answer these questions, we used Copilot to help solve coding assignments from two in-person courses. The first course, CS0, was a Python course designed to help introduce algorithmic problem solving to students with little to no programming experience. This course included eight programming assignments (PAs) covering common introductory topics, which are listed in Table 1. The second course was an introductory course in data science, henceforth referred to as DS1. This course also used Python; however, the students in this course had either already taken at least CS1 (in C++) or were concurrently enrolled in CS1. There were seven data-oriented programming assignments (DAs) in this course. The main topic(s) of each DA are also provided in Table 1. Different from the CS0 PAs, DS1's DAs used non-standard Python libraries that are commonly used in data science, including NumPy, SciPy, Pandas, Requests, Matplotlib, and Sci-kit Learn. DAs also included non-programming components, such as ethics write-ups and prep work for the final project, which were omitted from our analyses.

To answer research question #1, a sophomore computer science major solved the assignments from both courses with and without Copilot enabled. Copilot solutions were generated by copying and pasting text from assignment descriptions into code comments. The first suggested code snippet was accepted as the Copilot-generated solution (see the Appendix for an example using DA1). During this process, code, notes, and screen recordings were saved to document the workflow. After we acquired preliminary results with both assignment sets, we further investigated how Copilot's DA solutions compared to student-authored solutions from an offering of the DS1 course in the Fall of 2021 (N = 32 students). We obtained approval from our university institutional review board to utilize DS1 student code for research purposes. For research question #2, we utilized the hypothetical grades Copilot DA solutions would have received if they were turned in as student work. To do this, we asked the Fall 2021 student grader, a junior computer science major, to score the Copilot submissions as if they were student-authored.

We were interested in researching question #3 for a few reasons. First, Copilot is trained on public Github repositories and students (and experts

Table 1: CS0 programming assignment (PA) and DS1 data assignment (DA) descriptions, student perceived difficulty, and Copilot evaluation metrics.

Main Topic(s)	**Student Difficulty**	**Correct -ness**	**Appropri -ateness**	**Style**
PA1 Arithmetic	Low	4.2	5.0	5.0
PA2 Functions	Medium	4.6	5.0	4.9
PA3 Conditionals	Low	3.3	5.0	5.0
PA4 Loops	Medium	3.7	4.3	4.0
PA5 Files	High	3.3	5.0	4.3
PA6 Lists	High	3.6	3.9	4.6
PA7 Strings	Medium	4.7	5.0	4.5
PA8 Classes	High	3.1	5.0	5.0
DA1 Lists/simple stats	Low	5.0	5.0	5.0
DA2 2D lists/CSV files	Medium	3.9	4.9	5.0
DA3 Data wrangling	High	3.6	4.3	3.9
DA4 APIs/JSON	Medium	4.8	5.0	4.5
DA5 Jupyter/data viz	High	4.0	4.1	4.2
DA6 Hypothesis testing	Medium	4.5	3.9	5.0
DA7 Machine learning	High	2.8	3.8	4.1

alike) often search the web for help. Second, because the DS1 course was at the introductory level, we anticipated Copilot may produce a common solution that a student would likely come up with on their own. Finally, because Copilot was available in technical preview during the DS1 course, some students were already using Copilot. Of the 32 students, 3 voluntarily stated they were playing with the Copilot extension in Visual Studio Code. To answer this research question, we uploaded all the student and Copilot DA solutions to three software similarity/plagiarism detection systems: MOSS (Measure of Software Similarity), Codequiry, and Copyleaks. These tools compute the percentage of overlapping code between all possible pairs of solutions. Codequiry and Copyleaks also compare solutions with code found on the web.

4 Results and Discussion

We assessed CS0 and DS1 Copilot solutions using several evaluation metrics. Regarding research question #1, Table 1 shows the topics and student perceived difficulty of the assignment, then provides scores for Copilot's generated code solutions in terms of correctness, style, and skill level appropriateness. The range for each of these metrics was 0 (low) to 5 (high). In general, we

were impressed with how well Copilot solved most tasks in both courses, including the later ones in DS1 that used non-standard libraries and/or web APIs. Except for documentation, Copilot perfectly solved one assignment, the first and simplest one in DS1. Copilot was unable to solve all the tasks in the other assignments, even the first few in CS0 on topics such as arithmetic and functions. Here are examples of incorrect solutions Copilot generated:

1. Using a non-current year when calculating a user's age
2. Using ceiling when rounding should be used
3. Incorrectly implementing game logic, even when provided the game rules
4. Using a library without importing it
5. Not validating user input in a certain range of correct values

Such difficulties often stemmed from details of the assignments' contexts, such as parsing a text file containing stops made on a road trip (PA5). The format of the file was "custom" to the assignment: road trip stop information was separated by newlines, with slightly different information available for the starting location and the destination location.

In general, Copilot used coding syntax and techniques appropriate for the student's skill level. Occasionally it would use a technique not covered in the CS0 course, such as command line arguments or list comprehensions. Furthermore, some of the solutions were not written clearly, such as unnecessarily opening and closing a file repeatedly or including unreachable code (such as after a `return` statement). For style, Copilot generally used consistent naming conventions and formatting. Regarding documentation, Copilot sometimes generated inline comments at the top of files, but rarely within code to describe it. For other aspects of style and "good" coding practices, here are a few examples of Copilot's shortcomings:

1. Writing long functions that do not follow a good bottom-up design
2. Missing output formatting details
3. Including grammar mistakes in user-facing strings
4. Failing to add a unique "personality" to interactions with the user
5. Modifying original memory when should be working with a copy

Figure 1 shows results from investigating research question #2 about the grades Copilot-generated code would have received for DS1 assignments. Note that some grades in Figure 1 were higher than 100% because each DA included one or more bonus tasks. On average, Copilot scored a 84% on the programming portions of the DAs, with a score on DA3 higher than DA3's average student score. Mostly, Copilot did not receive full credit because of runtime errors (e.g., attempting to compute numeric statistics using string inputs), logic errors (e.g., producing incorrect results), incomplete tasks (e.g.,

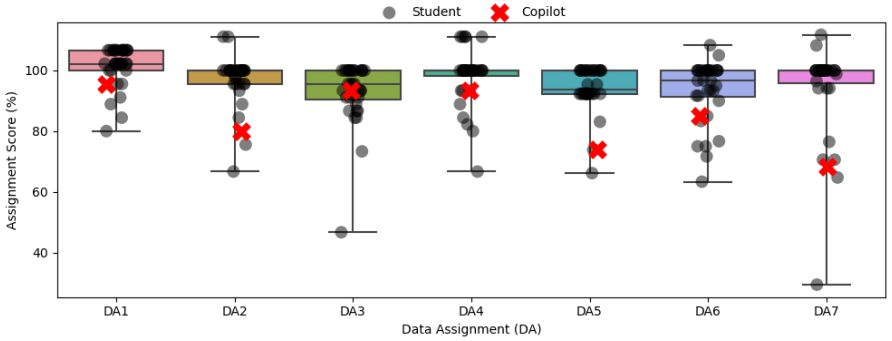


Figure 1: Student (N = 32) and Copilot scores for each data assignment.

Table 2: Copilot-generated data assignment (DA) solutions' software similarity and detected plagiarism scores.

DA#	MOSS (student/student max)	MOSS (Copilot/student max)	Codequiry	Copyleaks
1	75%	31%	0%	0%
2	55%	8%	0%	0%
3	91%	36%	0%	0%
4	66%	15%	0%	0%
5	80%	28%	0%	5%
6	74%	12%	0%	0%
7	57%	12%	0%	0%

failing to read data from a JSON file), and low adherence to a coding standard (e.g., not documenting code). The prevalence of Copilot-generated logic errors (about one per assignment), provides evidence that students should begin learning formal testing techniques in introductory courses. Copilot received a particularly low percentage score (68%) for DA7 because it struggled to use linear regression with a common machine learning dataset. This low score was still considered a passing grade ($\geq$ 60%), indicating Copilot can significantly help students obtain decent grades on introductory Python assignments with minimal input from the student.

The software similarity and plagiarism detection results did not suggest plagiarism on behalf of Copilot (see Table 2). The highest MOSS Copilot/student match over all DAs was 36%, which was lower than each DA's highest student/student match. Investigating the Copilot/student matches on a case-by-case

basis revealed that the matching lines of code were often scenarios where code was expected to be common among student submissions because there was a typical solution or there was a need to declare similar variables. For example, in DA1 students were expected to write their own code to implement simple summary statistics like mean, median, mode, standard deviation, etc (see Appendix). In Copilot's DA1 solution, the following median code was flagged as a match to student-authored code, though this is a generic solution:

```
if count % 2 == 0:
  median = (numbers[count // 2] + numbers[count // 2 - 1]) / 2
else:
  median = numbers[count // 2]
```

Codequiry and Copyleaks detected even lower matches with student code and with code found on the web. Only Copyleaks detected a 5% match on DA5, which was due to a common Python dictionary declared to help with a data cleaning/preparation task. To summarize our conclusions for research question #3, Copilot-generated code is generally not similar enough to student-authored code to suggest plagiarism. This suggests that students are less likely to violate academic integrity policies by using Copilot for help than looking at a classmate's code for help; however, as more students start using Copilot, issues may arise. Even though Copilot suggestions can be different for the same input, there is likely going to be scenarios where students may turn in similar solutions using Copilot independently.

Because Copilot is easy to use and can accurately solve a variety of problems, AIDE functionality will soon become standard in many coding workflows, including those used by professionals and students of all levels. While novice programmers should learn industry standard tools, they will still need to understand the problem and the code generated by the AIDE. Furthermore, all programmers need to know how to test code for functional correctness and efficiency, and how to adhere to a coding standard, regardless of who/what wrote the code. Therefore, based on our experience, literature review, and results presented in this paper, we recommend the following suggestions for educators to prepare for teaching programming in an AIDE-prevalent era:

1. Describe AIDEs to students, ensuring that all students have equal access to the tools, not just the ones "in the know"
2. Explain the advantages and disadvantages of using AIDEs
3. Design assignments to have unique contexts and personalization opportunities that differentiate solutions from commonly written code
4. Teach debugging and testing techniques in introductory courses, such as unit testing and test-driven development
5. Include assessment methods, like written/oral exams or reflections, that ensure students can solve problems without relying on AIDE assistance

5 Summary and Future Work

In the near future, AIDE-like functionality will likely become the new code completion standard. In this paper, we explored the possible effects of AIDEs, namely Github Copilot, in introductory programming courses. Our preliminary results suggested that Copilot is an easy to use and accurate enough tool that novice programmers can use it to write the majority of the code required to solve fundamental programming tasks. For an introductory data science course with Python, we found that Copilot-generated solutions earned a human-graded score between 68% and 95%. When compared to actual student-authored code, Copilot had very low code similarity scores. This suggests AI-generated solutions are unique and difficult to distinguish from human-authored solutions. For these reasons, we encourage computer science educators to become familiar with how AIDEs work and begin to design their coursework and development workflow to incorporate them. Future work includes observing how novice programmers utilize Copilot and analyzing similarities among code produced by different students using Copilot for the same assignment.

References

[1] Sven Amann, Sebastian Proksch, Sarah Nadi, and Mira Mezini. A Study of Visual Studio Usage in Practice. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 124–134, March 2016.

[2] Owura Asare, Meiyappan Nagappan, and N. Asokan. Is GitHub's Copilot as Bad As Humans at Introducing Vulnerabilities in Code? Technical Report arXiv:2204.04741, arXiv, April 2022.

[3] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew,

Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code. Technical Report arXiv:2107.03374, arXiv, July 2021.

[4] Matteo Ciniselli, Luca Pascarella, and Gabriele Bavota. To What Extent do Deep Learning-based Code Recommenders Generate Predictions by Cloning Code from the Training Set? Technical Report arXiv:2204.06894, arXiv, April 2022.

[5] Neil A. Ernst and Gabriele Bavota. AI-Driven Development Is Here: Should You Worry? *IEEE Software*, 39(2):106–110, March 2022. Conference Name: IEEE Software.

[6] Ekaterina A. Moroz, Vladimir O. Grizkevich, and Igor M. Novozhilov. The Potential of Artificial Intelligence as a Method of Software Developer's Productivity Improvement. In *2022 Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pages 386–390, January 2022.

[7] Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. Technical report, August 2021.

[8] Zhensu Sun, Xiaoning Du, Fu Song, Mingze Ni, and Li Li. CoProtector: Protect Open-Source Code against Unauthorized Training Usage with Data Poisoning. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pages 652–660, New York, NY, USA, April 2022. Association for Computing Machinery.

[9] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*, pages 1–7, New Orleans LA USA, April 2022. ACM.

[10] Frank F. Xu, Bogdan Vasilescu, and Graham Neubig. In-IDE Code Generation from Natural Language: Promise and Challenges. *ACM Trans. Softw. Eng. Methodol.*, 31(2):29:1–29:47, March 2022.

[11] Albert Ziegler, Eirini Kalliamvakou, Shawn Simister, Ganesh Sittampalam, Alice Li, Andrew Rice, Devon Rifkin, and Edward Aftandilian. Productivity Assessment of Neural Code Completion. Technical Report arXiv:2205.06537, arXiv, May 2022.

Appendix

The following code excerpts are from Github Copilot's DA1 solution. Comments were student-generated with text copied/pasted from the assignment specification. Python code snippets were generated by Copilot using the preceding comments. Note the code has been abbreviated with ... for brevity.

```
# Create a list with the following numbers: 84.4,...
numbers = [84.4, 75.8, 42.1, 25.9, 51.1, 40.5, 78.4,
  30.3, 47.7, 58.3, 90.8, 50.5, 28.2, 61.8, 25.1, 91.0]
# Add 75.6 to the list
numbers.append(75.6)
# Compute the average of the numbers
average = sum(numbers) / len(numbers)
# Compute the standard deviation of the numbers
sum_of_squares = 0
for number in numbers:
  sum_of_squares += (number - average) ** 2
variance = sum_of_squares / count
standard_deviation = variance ** 0.5
...
# Round the results to 2 decimal places
average = round(average, 2)
standard_deviation = round(standard_deviation, 2)
...
# Print the results with informative messages
print("The average is", average)
print("The standard deviation is", standard_deviation)
...
# Prompt the user for a starting value and an ending
# value replace all values in the list between the
# starting and ending values inclusive with 0 and print
start = float(input("Enter a starting value: "))
end = float(input("Enter an ending value: "))
for i in range(len(numbers)):
  if numbers[i] >= start and numbers[i] <= end:
    numbers[i] = 0
print(numbers)
...
# Print the list in reverse without using reverse()
for i in range(len(numbers) - 1, -1, -1):
  print(numbers[i], end=" ")
```

Teaching Software Project Management through Roleplay via Game Development*

Nathaniel Kremer-Herman
Department of Computer Science
Seattle University
Seattle, WA 98122
nkremer-herman@seattleu.edu

Abstract

Providing opportunities to introduce proper software project management methodologies in undergraduate computer science programs can be difficult. This is especially so for programs experiencing struggles including it in their curriculum due to either the scale of the program (whether the number of faculty or students) or breadth of topics already taught. However, it is *invaluable* for students to be introduced to these concepts in a low stakes environment so they may be better prepared for their careers after graduating. We provide an experience report of a month-long undergraduate course which introduces key topics in project management as a side effect of hands-on experience developing video games. In groups, students delivered an educational video game from start to finish for a client (a faculty or staff member outside of the computing fields). Their client had a real need for their proposed video game, and students successfully met these needs. We demonstrate an effective environment to reinforce key project management topics and discuss lessons learned providing a free range environment for students to learn about project management.

1 Introduction

Software project management skills are useful no matter what area of computing one enters. Many skills make up project management including but not limited to: identifying client needs, creating prospective development timelines, outlining key software features, equitably dividing labor, and providing actionable metrics for measuring success during project development. Many methodologies currently exist to help manage software projects such as Agile [4], Scrum [8], and Kanban [2].

Teaching even one of these methodologies to sufficient depth in an undergraduate computer science program can be difficult. There is a risk of shoehorning project management principles into curricula where it does not truly fit, minimizing the impact of both the original course content and project management content. On the other hand, it can be difficult to teach project management concepts in isolation, especially if there are not sufficient stakes to make using a project management methodology necessary (such as a real client, a well-defined project, and/or consequences for missed deadlines). Finally, focusing an entire course on project management risks consuming available teaching slots for other courses in a department's curriculum. This final point highlights an issue for computer science programs experiencing scaling pains such as having too few faculty to cover a sufficient breadth of topics or too few students to offer courses which deviate beyond traditional curriculum.

We present an experience teaching an undergraduate, elective course in educational game development which, as an intended side effect, also taught the core concepts of the Kanban development methodology and required students to use the methodology in active development of a video game. This elective course served as an introduction to video game development as it applies to educational settings (K-12, undergraduate, professional development, and social outreach). The course introduced students to clients with predefined game ideas, had them team up to develop a game for their chosen client, and present their experience designing and implementing the game in a final showcase.

We demonstrate that the game development and pedagogical portions of the course were an effective veneer for teaching software project management without compromising the integrity of a course in the core curriculum nor teaching it in isolation without proper stakes. The class took on the role of a game development studio to reify the concepts of project management through hands-on experience. Through this persistent roleplaying, we maintained a level of adherence to some best practices in the Kanban methodology. The course consisted of self-assessment activities to track student growth and led to the successful creation of four educational video games for on-campus clients. We demonstrate that this experience resulted in significant increases across the board in students' self-confidence in key software project management skills.

2 Motivation and Related Work

Applying concepts of roleplaying and gamification to project-based courses have been successful at a variety of institutions. Multiple recent surveys [3, 5] and literature reviews [7] have noted an upward trend in the adoption of gamification as applied to software engineering education, particularly as it applies to software projects and software project management. In particular, these works note a wide diversity of methodologies for implementing gamification to blend theoretical concepts and hands-on experiential learning. We provide an experience report of applying roleplay to gamify learning software project management and provide an evaluation of the course offering's impact on student self-confidence in applying key project management skills.

One particularly insightful work applied roleplay throughout the structure of a video game development course [6]. Students were empowered to make group decisions like a game development studio: shelving game ideas that were developmental dead ends, refocusing work toward more successful projects, and enforcing deadlines. The course was structured around building the, "soft skills," typical of professional software development similar to the course offering presented in this paper. We used this work as inspiration for what real world concepts and structures could be feasibly represented in the classroom.

Another course used roleplaying elements to provide students structure for measuring their own growth toward skill mastery [1]. Students were provided character sheets similar to tabletop roleplaying games which contained the key course skills. As students went through the course, there would be opportunities to improve their skills on their sheets. This was accomplished through teamwork. We also sought inspiration from this work to encourage students to put together project groups composed of people with a variety of skills such as programming, digital arts, and website styling.

3 Course Design and Structure

This educational game design course was offered as an elective course with no prerequisites in computer science coursework (though an introductory computer science course was preferred). It was offered during a month-long intensive study term at the host institution, and as such this was the only class students had in their schedules. Its focus was to introduce principles of video game design and pedagogy as they apply to educational video games. Students self-selected groups of three to four and chose a project based on a list of game pitches (a paragraph of descriptive text) which corresponded with a client from faculty and staff outside of computing disciplines. Chosen games included one teaching institutional fire safety policies, a physics game about vector fields,

a game teaching basic color theory, and a philosophy game demonstrating the Prisoner's Dilemma.

During the first week, each group met with their client to better understand what game they would be creating. They then spent the month developing an educational video game from start to finish. This involved planning and designing features, weekly client meetings, equitably implementing their selected features, evaluating the effectiveness of their game as educational content and as entertainment, and communicating their progress to their peers in class.

Each class session was split into lecture (1 hour) and free work time (2 hours). The lecture portion introduced topics relevant to game design, education, and project management. The work time began with a daily standup meeting within each group followed by a daily report to the class from each group. During the standup, students consulted their shared Kanban board and updated it accordingly. Students were then allowed to work for as much or as little as they felt necessary each day. At the end of the month, every group successfully completed an educational video game for their chosen client.

Table 1: Educational Game Design Course Information

Course Name	Educational Game Design
Prerequisites	None
Enrollment	13 of max 16
First Offering	2022
Rotation	Every other year
Languages	JavaScript, C#, HTML, CSS

3.1 Learning Objectives

This educational game design course introduced education topics such as the principles of popular educational models, using storytelling/roleplaying as vehicles for learning, and gamification of education. Elements of game design such as storyboarding, implementing game mechanics, measuring user interactivity, and aesthetics were taught to have students create an *interactive*, *entertaining*, and *effective* educational game. The students were required to assess their game through playtesting along those three axes. As key course learning objectives, students were tasked to:

- Communicate and divide work among a team toward a common goal.
- Effectively use project management tools to prioritize work on a deadline.
- Identify the facets of entertainment, educational, and edutainment media.
- Identify best practices for understanding client and user needs.
- Translate a client's vision into user interfaces and activities.

3.2 Key Topics

The learning objectives were reinforced across various topics. Table 2 lists the key topics introduced during the month-long course. Included are the topic name, the week it was introduced, and the focus (education, game design, or project management). The topics are also listed in the order they were introduced to students. Class sessions were held daily Monday through Friday.

Table 2: Key Course Topics

Topic Name	Week Introduced	Focus
Principles of Proj. Management	1	Proj. Management
Version Control and GitHub	1	Proj. Management
Identifying Client Needs	1	Proj. Management
Video Game Design Primer	1	Game Design
Educational Games	2	Education
Web Browser Games	2	Game Design
User Interfaces	2	Game Design
User Assessment	2	Education
User Testing and Feedback	3	Game Design
Release/Go-to-Market Strategy	3	Proj. Management
Patches/Continued Support	3	Proj. Management
Case Studies	4	All

Three days in the month were dedicated as solely work time with no lecture content. Similarly, the final week of the course focused more on work time than lecture. Instead of lecture, case studies of extant educational game companies and educational games were discussed to demonstrate the field being a viable career to students. Additionally, discussions about the line between entertainment and education and the ethics of logging user behavior in-game were facilitated during the final class week. As such no new topics were introduced.

3.3 Assessments

Assessment of student learning was divided into four categories: client updates, in-class participation, a final project, and a final presentation. Student groups were required to meet with their client once a week. Each week, every group filled out a form summarizing their client meeting. This included summarizing their demonstrated progress, the division of labor, noting client feedback, stating upcoming targets for the following week, and listing any roadblocks the group encountered during that week's development. This assessment was meant to act as a guide for questions they should ask their client, a layer of accountability to ensure progress was being made, and a tool for self-reflection

of progress made each week.

In-class participation was not graded solely on attendance or discussion. Whenever a student led their group's daily standup or provided the class their group's daily progress update, they received participation points. Students also provided other groups peer feedback on the progress of their game as well as their source code three times throughout the month. Participants in the peer feedback sessions also earned participation points.

The final project and presentation, though separate assessments, were tightly coupled. The project was required to be open source, hosted on GitHub, and be playable with few to no errors. Students also wrote a game manual which instructed players how to play the game. Also included in the manual was commentary on the key features of the game as told by the student who took the lead developing each feature. This added additional self-reflection as well as insight into the students' development processes throughout the month.

The presentation was the culmination of the term. Each group was given twenty minutes to present a slideshow going over their client's initial vision, the group's identified client needs, the key features and mechanics of their game, conclusions they have drawn about their final product as it related to their client's needs, and a live demo of their game in action. After all the presentations, the class (and visitors) were invited to play the completed games (and read through printed manuals) for the remainder of the class session. This open house showcase allowed the students to provide further insight into their games with their peers and the visiting faculty and students.

4 Evaluation

Evaluation of this course occurred in multiple forms. Students were given a course pre-test and post-test asking them to rate their confidence in key skills as well as demonstrate knowledge. Additionally, course evaluations were gathered on behalf of the institution.

4.1 Course Pre-Test and Post-Test

The pre-test and post-test consisted of rating scale questions and free response questions. The scale ranged from 0 to 5 (a score of 0 indicated no experience while a 5 indicated most confidence). Students rated their self-confidence in tasks related to project management: programming, debugging, group communication, use of project management tools, prioritization of work, client communication, and translating client vision into features. They were additionally asked to rate their confidence in their ability to differentiate between educational, entertainment, and edutainment media. The free response questions

included space for students to identify what can make a video game an effective educational tool, best practices for identifying and understanding client needs, and methods for ensuring a group completes assigned work on time. All 13 students took both the pre-test and post-test.

We performed a paired Student's t-test on the rating scale questions to gauge growth in student self-confidence between the pre-test and post-test. Table 3 demonstrates the results of the t-test and the mean scores for the measured activities. The mean score is shown to have increased significantly. Since each group successfully completed their games, we may infer their successes demonstrated an overall increased confidence gained during the month in the tasks referred in the tests.

Table 3: Summary of t-test on Pre-Test and Post-Test

p-value	0.0001	
Confidence	95%	
t	6.7499	
Test	Pre	Post
Mean Score	3.279	4.332
Std. Deviation	0.764	0.493
Std. Error of Mean	0.212	0.137
Min. Score	2.250	3.375
Max. Score	4.375	4.875
Activity	Pre	Post
Programming	2.846	3.923
Debugging	3.000	4.000
Group Communication	3.692	4.462
Project Management Tools	2.461	4.077
Work Prioritization	4.000	4.462
Client Communication	3.923	4.462
Understanding Client Vision	2.769	4.154
Entertainment vs. Educational	3.538	4.615

4.2 Course Evaluations

The course evaluations were collected during the final week of classes. Each response was anonymous and collected by the institution through a standardized online form. When asked what was most effective about the course, one student mentioned, "I think the freedom for groups to just start working independently on games worked really well for a course meant to be about experiential learning. It made every little mistake and breakthrough a learning opportunity." This was rewarding feedback to receive since this student seemed to truly grasp

the purpose of the course. When comparing the course to others, another student noted, "Most other computer science classes will create multiple labs or assignments when this is not necessary. Being able to focus into a project and have passion about it has been invaluable to my learning experience." Another student remarked that using Kanban was especially effective for their learning.

Of the 13 students in the course, 9 filled out course evaluations. All responded that the course and assignments positively contributed to their learning. Every respondent also indicated that, after taking this course, their interest in the subject area increased. For the students, this was an irregular course in the curriculum, and it would seem the course structure was particularly effective and novel for all who provided evaluations.

5 Discussion

Given the feedback provided by students, it became clear this offering of an educational game development course was effective in increasing students' self-confidence in key skills related to software project management. We consider this course offering successful in accomplishing its goals. We can attribute this accomplishment to a few key factors: the majority of learning was experiential, students were frequently tasked to reflect on their learning, and there were multiple layers of accountability to keep students progressing.

5.1 Free Range, Experiential Learning

Being a project-based course, the majority of in-class time was devoted to group projects. This free range approach was effective in shepherding students toward hands-on learning of course topics. Structure was provided to encourage students to discuss design decisions with their peers, to share technical struggles and triumphs, and to work together to successfully apply course concepts. This took the form of mandatory daily standup meetings, daily progress reports to the class, and instructor encouragement to share student insights for the betterment of the class.

5.2 Frequent Opportunities for Self-Reflection

Select in-class activities concluded with a brief self-reflection survey where each student would have to summarize the purpose of the activity in their own words, to rate the effectiveness of the activity, and to reflect on what they liked and disliked about it. This provided in-class time to reflect on learning at least once per week. Additionally, students were asked to look back on their progress toward their project each week by filling out their weekly client update form. Each student had to summarize their own contributions, and the group

as a whole needed to determine how far along they believed they were relative to both their own plan and their client's expectations.

5.3 Layers of Accountability

Throughout the month students always had multiple layers of accountability to ensure forward progress was being made on their projects. They were graded for submitting a weekly client update which summarized the group's contributions for the given week. This was submitted to the instructor *and* the client for their review, ensuring that the instructor and client were being told the same thing. Groups were also required to meet with their client once a week at a time of their choosing. This ensured face-to-face time to clear up miscommunications, to demonstrate progress to clients, and to encourage honest dialogue for roadmapping the month.

Additionally, there were also three separate in-class peer feedback sessions. In each session one student in each group would demo their game progress and walk through their source code. Students from other groups would visit these presenters and give feedback by filling out brief feedback forms. This feedback was collected by the presenters and shared with their fellow group members at the end of the session. Once the feedback was understood by all in each group, the instructor collected the forms as part of the course participation grade.

Finally, students were given the option to submit assessments of their fellow group members at the end of the month. This was a last chance at accountability in case group communication had not corrected any inequitable divisions of labor (taking into account relative student skill level). Only three students filled out this optional form.

6 Conclusions

An important consideration for the success of this course was its scale. A class size of 13 is quite small, and there was ample time for student-instructor interaction in every class session. The enrollment also limited the number of groups (thus projects to shepherd) to four. It is easy to imagine scaling issues this course would experience if offered as-is with higher enrollment.

However, the benefit of providing students with real world experience while still within the safety of the classroom cannot be overstated. In this course offering, students on the whole saw a marked improvement in their self-confidence across fundamental skills in software development and software project management. Through the lens of video games (an incredibly popular medium for typical undergraduates), the students were able to engage with topics that will benefit them throughout their computing careers: communicating to both technical and nontechnical audiences, rallying a team around a shared goal,

independent programming and debugging, and realizing an idea into software from inception to release. From their provided self-reflections and course evaluations, it would seem they are also aware how profoundly the free range course structure allowed them to make the most of this educational experience.

References

[1] Azim Abdool, Daniel Ringis, Aniel Maharajh, Lynda Sirju, and Hannah Abdool. Datarpg: Improving student motivation in data science through gaming elements. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, 2017.

[2] Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. Kanban in software development: A systematic literature review. In *2013 39th Euromicro conference on software engineering and advanced applications*, pages 9–16. IEEE, 2013.

[3] Rodrigo Henrique Barbosa Monteiro, Maurício Ronny de Almeida Souza, Sandro Ronaldo Bezerra Oliveira, Carlos dos Santos Portela, and Cesar Elias de Cristo Lobato. The diversity of gamification evaluation in the software engineering education and industry: Trends, comparisons and gaps. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 154–164, 2021.

[4] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001.

[5] Muhammad Shoaib Farooq, Ayza Hamid, Atif Alvi, and Uzma Omer. Blended learning models, curricula, and gamification in project management education. *IEEE Access*, 10:60341–60361, 2022.

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

[7] Marek Milosz and Elzbieta Milosz. Gamification in engineering education – a preliminary literature review. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 1975–1979, 2020.

[8] Ken Schwaber and Mike Beedle. *Agile software development with scrum. Series in agile software development*, volume 1. Prentice Hall Upper Saddle River, 2002.

Successfully Incorporating a Cyber Security Competition into an Intro to Computer Security Course*

David Zeichick
Computer Science Department
California State University, Chico
Chico, CA 95926
dzeichick@csuchico.edu

Abstract

Competitions incorporated into a computer security class can complement the theoretical content and enhance the practical material. When the material is taught in a fun and scaffolded way, computer security competitions offer an invaluable hands-on learning experience for students. However, experience shows that adding a competition as a supplemental activity does not necessarily lead to success. The course must prepare students for the competition, providing them with sufficient background knowledge to successfully approach the challenges. Additionally, student success and enjoyment depend upon group collaboration.

This research studied the evolution of an introduction to computer security class over the period of fifteen semesters, as purposeful enhancements based on student feedback were made to best incorporate the National Cyber League competition into the course.

1 Introduction and Related work

The goal of selecting a method of instruction is to create a learning environment in which students efficiently and effectively acquire skills and knowledge in a

way that is appealing to the student[6]. Over the last few decades, there has been a paradigm shift in which teachers focus more on student learning than teaching, giving more control to the student[1]. The idea is that students independently form knowledge as the instructor acts as a guide through the process[10].

Many terms have been used to describe this approach, such as flexible learning[19], experiential learning[2], self-directed learning, and the more apparent student-centered learning[16]. Student-centered learning has been summarized as focusing on 1) deep learning and understanding, 2) active rather than passive learning, 3) student accountability, and 4) student autonomy[11].

Deep learning involves applying skills and knowledge to a real-world problem[13]. The effectiveness of this approach was demonstrated in a study by Schwartz et al. that included 8,000 students at 45 universities and found that the students whose high school science classes focused on depth outperformed their counterparts whose high school science classes were breadth-focused[18]. For this to work, the student must be able to choose their learning path and pace while receiving continuous, rapid feedback on their progress[13].

Active learning should focus on what the student is doing rather than what the teacher does[5]. In the computer science field, cyber competitions such as Capture the Flags (CTFs) have been used for active learning experiences for over 20 years[17]. Adding a cyber competition as an activity for students provides autonomy since students typically work on it at their own pace. Additionally, the achievable goals and progress inherently designed into a cyber competition meet the requirement for accountability in a student-centered learning curriculum[7].

Competitions are instrumental in cyber security due to the complex nature of the knowledge and skills included in the domain[9]. Adding a competition to a class is a method of gamifying the curriculum, which has been shown to make learning more fun, increasing the motivation to progress, and thereby easing the burden of the complex material[14]. Competitions' open-ended assessment measures students' progress and encourages them to freely expand their skillset, pushing their limits[13]. Students have reported spending many hours attempting to solve a challenge through their research on the Internet[3]. In fact, students' motivation to solve challenges was significantly greater than their effort toward traditional class assignments[3].

A student's score in a competition is an effective way to measure learning. Chothia and Novakovic studied the correlation between students' understanding of cyber security topics and their scores in a cyber security competition and found that they correlate highly with[4]. Another benefit is that significantly less cheating was discovered in a class involving a cyber security competition than in a course involving traditional assignments[4].

Cyber security competitions have been in existence for many years; iCTF, hosted by UC Santa Barbara, is one of the most significant and longest-running annual events dating back to 2002[4]. The goal of a typical CTF is for a team to attack an unpatched competitor's host while at the same time defending their vulnerable host[9]. An offshoot of this type of competition is CyberPatriot which involves high school students attempting to harden Windows and Linux systems in the quickest[13]. Another type of cyber competition involves students solving challenges in a jeopardy style where points are awarded for each challenge solved with more points for more difficult challenges. One of the largest competitions is the National Cyber League (NCL), in which over ten thousand high school and college students participate every year.

The NCL is a biannual competition designed for high school through college students to solve real-world cybersecurity problems[12]. There are four phases to the NCL: 1) the Gym, which includes step-by-step instructions designed to assist students in solving the fundamental challenges, 2) the Practice Game, in which students are allowed to assist each other in solving challenges, 3) the Individual Game where students work independently, and 4) the Team Game where up to seven students work as a team. All four phases of the competition include easy, medium, and hard questions from nine cyber security categories: cryptography, password cracking, log analysis, network traffic analysis, forensics, web application exploitation, scanning, enumeration and exploitation, and open-source intelligence. These categories align with the National Institute of Standards and Technology's (NIST) National Initiative for Cybersecurity Education's (NICE) work roles[15].

There are several published pitfalls related to incorporating cyber security events into a course's curriculum. First, not everyone likes to compete; trait competitiveness refers to the extent to which people enjoy interpersonal competition and the desire to win[8]. This problem can be exacerbated through the use of leaderboards. Studies of leaderboards have been mixed, ranging from showing favorable to showing adverse effects[8]. It is possible that this discrepancy has to do with how leaderboards are related to the student's grades for the competition (this will be demonstrated in the findings of this research).

Next, not all CTFs are designed to encourage novice players[9]. This leaves new players in a state of frustration; therefore, very little learning takes place. This has compounding effects on students that are used to a traditional teaching style in which they expect to be taught all that they need to know to solve a particular challenge and get frustrated when presented with an unfamiliar challenge[3]. Another related issue is that most competitions don't include partial credit; therefore when a student feels like they can't solve a challenge, they don't even attempt it[9]. Lastly, a frustration among competitors is time constraints[17]. Students who are required to compete as part of a class need

to balance the time they invest in a competition with other classwork.

2 Methods

This research studied the evolution of an introduction to computer security class over fifteen semesters. The metrics used were anonymous surveys called the student evaluation of teaching (SETs). This was given to all students near the end of each semester in a required introduction to computer security class for computer science students at a four-year university. In total, 547 out of 765 enrolled students completed the SET. This is an impressive 72% participation rate. The same professor taught the class throughout the period studied. It is important to note that this survey is given to all students at the university and is not specific to the introduction to computer security class; it is a generic survey created to apply to all courses at the university and therefore does not mention anything about security competitions.

The first survey included in this research was from Spring 2013, the first year the introduction to computer security class was taught. At the time of inception, a computer security competition was not part of the curriculum. It wasn't until five semesters later, in the Fall of 2016, that the NCL was incorporated into the class.

The class continued with minor changes in grading students on the competition for seven more semesters. In the Spring of 2020, a post-competition debrief was added, in which students shared their answers and struggles with the competition with each other. Then, two semesters later, in the Spring of 2021, major changes to the structure and grading of the class were implemented. This involved rearranging the course material to cover all categories of the NCL before the start of the competition[20]. Another change was that students were given a set score to aim for, for a grade (1000 points would give them an 85% for the lab) when in previous semesters, the students' grade in the competition was based on the final average grade across all students in the class. Finally, the students were not required to participate in all phases of the competition, leaving the Team Game as optional extra credit.

3 Results and Discussion

The impact of incorporating the NCL into the Introduction to Security class was measured by analyzing the results of the SETs, which were completed anonymously by the students each semester. The SETs include both quantitative and qualitative information. The quantitative score is based upon a five-point scale, with one being the lowest and five the highest. The students

are also allowed to provide comments about their overall experience of the class. These comments have been categorized based on any mention of the NCL.

The SETs include various questions about the quality of the teaching, the course material, and the assignments. All of the SET questions were reviewed to identify the questions that would be relevant to measuring the impact of incorporating the NCL into the class. Two questions were found to be relevant: 1) the course assignments contribute to learning, and 2) the course increased my knowledge of the subject.

3.1 Quantitative Results - SET Scores

The class was first taught in the Spring of 2013, and five semesters later, in the Fall of 2016, the NCL was adopted. The introduction of the NCL appears to have made a big impact, reflected in significant jumps in SET scores for both questions being evaluated; the average score related to assignments contributing to learning jumped from 4.45 to 4.77, which is a 7% increase, and the score related to knowledge improved from 3.97 to 4.92, an impressive jump of 24% (see figures 1 and 2 below). This increase was not maintained over the subsequent nine semesters. However, the average pre-NCL introduction to post-NCL introduction across all semesters showed a rise of 5% for the assignment SET question and 11% for the knowledge-related SET question. The Spring of 2017 knowledge-related SET question score was an outlier, with a score of 4.21. After careful analysis of this semester, no reason was discovered for this lower-than-average score.

Figure 1: SET results on a scale from 1 to 5 for the question about the course assignments contributing to learning.

Another improvement in the SET score for the question on course assignments contributing to learning occurred in the Spring of 2020 with the addition of a group assignment designed to disseminate knowledge gained from the NCL (see figure 1 above). The group assignment involved self-forming groups of three and describing how to solve one of the NCL challenges. They were

Figure 2: SET results, on a scale from 1 to 5, for the question about the course increasing the students' knowledge of the subject matter.

to post their solution on a Google Document shared with the entire class so that the course could all learn from each other. The SET score increased on average across all semesters by 5% from pre-assignment to post-assignment.

The next significant change to the curriculum occurred in the Spring of 2021. Based on comments from students (see the comments section later in the paper), the class schedule was revamped to include instruction and labs on eight of the nine cyber security categories. The only category not covered is open-source intelligence since it involves the ability for students to Google for information, a skill they can build on their own. This change involved front-loading the class in which most of the labs were assigned in the first half of the semester instead of the second half.

Another change introduced in the Spring of 2021 was how the NCL was graded. In previous semesters students were graded on doing all three games: the practice, individual, and team games. However, students commented that this was too much of a time requirement. To address this issue, the team game was no longer required and offered as an extra credit lab. The other grading change was to their grade based on the points they scored. Before this change, students were graded based on the average score of the class. The average was constantly changing as students solved challenges. The class average from past semesters was relatively consistent at 1000 points. Therefore, it was announced before the start of the competition that for the students to earn a B (or 85%) in the competition, they would need to score 1000 points. If they scored above or below that amount, a sliding scale would be used to determine their grade.

The SET scores for both of the SET questions for Spring of 2021 were the highest since the class's inception and close to perfect. The score for the question about course assignments contributing to learning was 4.94 out of 5, and the question about the course increasing their knowledge of the subject matter was 4.97 out of 5 (refer to figures 1 and 2 above).

3.2 Qualitative Results - SET Comments

The SETs include a section where students can include comments about the class. In 2013, the prompt for the comments was generic. In the Fall of 2017, this changed to have two separate comment sections with the prompts "what did your instructor do to make this class a good learning experience for you" and "what could your instructor do in the future to make this a better class." Therefore, there were two semesters, Fall 2016 and Spring 2017, in which the comments section differed since the class had adopted the NCL. This change greatly affected the number of comments that students made, increasing from 66% before Fall 2016 to 89% after. However, this change does not appear to have affected the percentage of comments related to the NCL.

On average, 54% of the comments made about the class are NCL related. It needs to be reemphasized that the SET does not mention the NCL by name and includes the same questions in all courses across the university. So, without any prompting about the NCL, on average, half of the students' comments were related to the NCL. Additionally, 48% of the comments pertaining to the NCL were purely positive. For example, one student stated, "The NCL competitions are a huge plus in being able to apply some of the knowledge learned throughout the class." Another student went as far as saying that the "NCL really is the best part of this class. It really helps you get a feel for security and what it entails."

As stated above, 48% of the NCL-related comments are purely positive, so what about the other 52%? The remaining NCL comments have been organized into the following four categories: 1) the students didn't feel prepared, 2) the students didn't like how the NCL was included in grading, 3) the NCL took up too much of the students' time, and 4) they just did not like the NCL. This last category comprised just 4% of the overall NCL-related comments (5 out of 133 NCL-related comments over ten semesters). Almost a third, 29%, of the comments related to students not feeling like they were prepared for the NCL.

The percentage of comments about not being prepared drops to 13% and 19% for Fall 2020 and Spring 2021, respectively (see figure 3 below). During these two semesters, the purely positive comments comprised an average of 66% of the NCL-related comments. This period coincides with COVID and the move of the class to a completely online format. Since the NCL is already online, the move for this class and the competition was extremely smooth. The spike in positive comments could correlate to their appreciation of the NCL being available to them during the pandemic, while so many aspects of their classroom and life were forced to change.

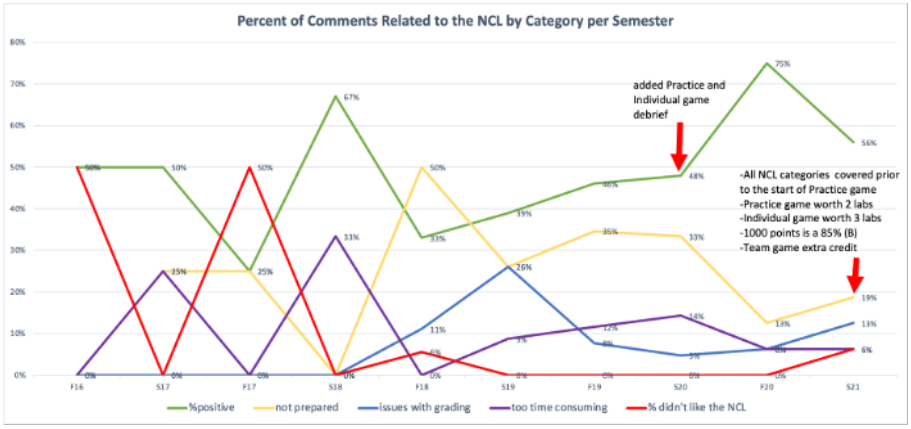


Figure 3: Percent of comments related to the NCL by category per semester.

4 Conclusions

Including the NCL as a series of labs in the class improved the SET scores for the question regarding the course assignments contributing to learning and the question about the course, increasing the student's knowledge of the subject matter by 5% and 11% on average, respectively. Unprompted, the students mention the NCL in just over half of their comments. Almost half of the NCL-related comments were completely positive, about a third commented about how they didn't feel prepared. The remaining either didn't like how the NCL was incorporated into their grade, thought the NCL was too time-consuming. In response to feedback from the students, adjustments to the class were made, further improving the SET scores (see figure 4 below).

These statistics point to a very favorable attitude towards the NCL. The students appear to enjoy the NCL and appreciate how it furthers their learning with applicable computer security-related skills. This is summed up with the following student's comment: "The addition of a competition to this class ... really allows you to learn a lot. It is clear that the topics in the NCL are a much more in-depth and complex look at what we learn in regular class time, so if you want to spend the extra time and learn all about security, this provides a great opportunity to dive deeper."

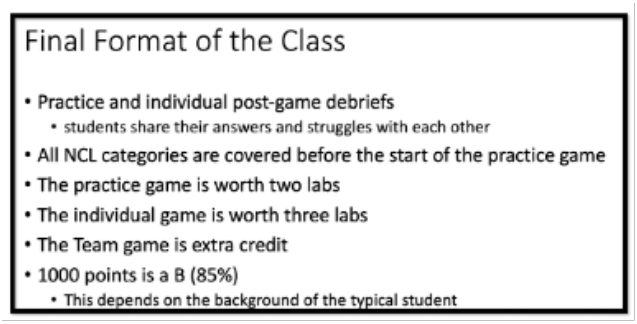


Figure 4: Summary of adjustments made to the class based on student comments.

References

[1] Robert B Barr and John Tagg. "From teaching to learning—A new paradigm for undergraduate education". In: *Change: The magazine of higher learning* 27.6 (1995), pp. 12–26.

[2] Philip Burnard. "Carl Rogers and postmodernism: Challenges in nursing and health sciences". In: *Nursing & Health Sciences* 1.4 (1999), pp. 241–247.

[3] Martin Carlisle, Michael Chiaramonte, and David Caswell. "Using {CTFs} for an Undergraduate Cyber Education". In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.

[4] Tom Chothia and Chris Novakovic. "An Offline Capture The {Flag-Style} Virtual Machine and an Assessment of Its Value for Cybersecurity Education". In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.

[5] Joy Crosby RM Harden. "AMEE Guide No 20: The good teacher is more than a lecturer-the twelve roles of the teacher". In: *Medical teacher* 22.4 (2000), pp. 334–347.

[6] Melissa Dark. "Advancing cybersecurity education". In: *IEEE Security & Privacy* 12.6 (2014), pp. 79–83.

[7] Juho Hamari, Jonna Koivisto, and Harri Sarsa. "Does gamification work?–a literature review of empirical studies on gamification". In: *2014 47th Hawaii international conference on system sciences*. Ieee. 2014, pp. 3025–3034.

[8] Christoph E Höllig, Andranik Tumasjan, and Isabell M Welpe. "Individualizing gamified systems: The role of trait competitiveness and leaderboard design". In: *Journal of Business Research* 106 (2020), pp. 288–303.

[9] Menelaos Katsantonis, Panayotis Fouliras, and Ioannis Mavridis. "Conceptual analysis of cyber security education based on live competitions". In: *2017 IEEE Global Engineering Education Conference (EDUCON)*. IEEE. 2017, pp. 771–779.

[10] David Kember. "A reconceptualisation of the research into university academics' conceptions of teaching". In: *Learning and instruction* 7.3 (1997), pp. 255–275.

[11] Susan J Lea, David Stephenson, and Juliette Troy. "Higher education students' attitudes to student-centred learning: beyond'educational bulimia'?" In: *Studies in higher education* 28.3 (2003), pp. 321–334.

[12] National Cyber League. *National Cyber League*. `https://nationalcyberleague.org`.

[13] Daniel Manson and Ronald Pike. "The case for depth in cybersecurity education". In: *Acm Inroads* 5.1 (2014), pp. 47–52.

[14] Cristina Ioana Muntean. "Raising engagement in e-learning through gamification". In: *Proc. 6th international conference on virtual learning ICVL*. Vol. 1. 2011, pp. 323–329.

[15] NICCS. *The Workforce Framework for Cybersecurity (NICE Framework) Work Roles | NICCS*. `https://niccs.cisa.gov/about-niccs/workforce-framework-cybersecurity-nice-framework-work-roles`.

[16] Geraldine O'Neill and Tim McMahon. "Student-centred learning: What does it mean for students and lecturers". In: (2005).

[17] Aunshul Rege. "Multidisciplinary experiential learning for holistic cybersecurity education, research and evaluation". In: *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. 2015.

[18] Marc S Schwartz et al. "Depth versus breadth: How content coverage in high school science courses relates to later success in college science coursework". In: *Science education* 93.5 (2009), pp. 798–826.

[19] Peter G Taylor. "Changing expectations: Preparing students for flexible learning". In: *International Journal for Academic Development* 5.2 (2000), pp. 107–115.

[20] David Zeichick. *Incorporating the NCL into an intro to computer security class*. `https://express.adobe.com/page/C9rJsqx6cVKaL/`.

A Survey of Cloud-hosted, Publicly-available, Cyber-ranges for Educational Institutions*

Stu Steiner[1], Ananth Jillepalli[2], Daniel Conte de Leon[2]
[1]Eastern Washington University, Spokane, WA 99202
[2]University of Idaho, Moscow, ID 83844
ssteiner@ewu.edu, ajillepalli@ieee.org, dcontedeleon@ieee.org

Abstract

Creating, deploying, maintaining, and updating cybersecurity hands-on labs is difficult at best, if not impossible for small to mid-size colleges and universities with limited faculty and staff. For each faculty member there is a trade off between creating hands-on labs versus actual instruction time. Both are important and the individual faculty member has to make a choice of what is best for their individual students.

In this research we conducted a survey of cloud-hosted, publicly-available cyber-ranges. This work doesn't explain how to create a cyber-range, instead it classifies each cyber-range based on five different categories including: classroom management, type of labs, number of labs, supplementary reading materials, and license management. Furthermore, we investigate the cost per student based on a variety of factors.

1 Introduction

There is an increasing global need for workforce ready trained cybersecurity professionals [3]. Being workforce ready means students are trained with hands-on exercises [1]. There are several methods for developing and deploying hands-on exercises for a cohort of students. Select methods include: (i) develop from scratch, deploy locally [1, 3]; (ii) develop from scratch, deploy on cloud [2]; (iii) curate existing labs, deploy locally [1, 3]; (iv) curate existing labs, deploy on cloud [2]; or (v) use a cloud-hosted, publicly-available service that has all

the needed labs for a specific training program. Methods (i) through (iv) are not the focus of this manuscript; we focus on facilitating use of a cloud-hosted, publicly-available cyber-range for educational institutions (v). A Cyber-range is defined as "A computing environment that enables the simulation of digital devices and their associated networks, including desktops, workstations, servers and services, mobile and IoT and/or ICS devices, switches, routers, firewalls, and other digital infrastructure items, with varied levels of complexity and fidelity, and for the purposes of supporting the safe and secure execution of cybersecurity exercises and experiments for instructional, research, and/or evaluation purposes."

Cyber-ranges are useful for instructors who may not have extensive computing infrastructure available, and want to minimize the initial required setup time. Furthermore, they enable students to access hands-on exercises in a plug-and-play environment.

1.1 Current Problem

Today, it is a challenge to identify cloud-hosted and publicly-available cyber-ranges that fit the requirements for a specific cybersecurity course [4]. Several characteristics of the platform and content need to be considered by the instructor to understand which cyber-range service can best achieve their course objectives.

Our investigatory question was: "Which turn-key and ready-to-use cloud-hosted cyber-ranges are available for enabling students to perform hands-on activities for a given cybersecurity-focused course?"

We searched Google Scholar for published works that attempted to address the question posted above. In our search, we found many articles reporting on cyber-ranges, but none that addressed the question. Most currently available articles are descriptions of the design and/or implementation of private research or educational cyber-ranges. There is a lack of literature describing cloud-hosted, publicly-available, education-focused, cyber-range services.

1.2 Contribution

In this article our goal is to help the cybersecurity educational community answer the question above. We achieve this goal by providing enough information and resources including a list of available cyber-range offerings. We introduce and classify their characteristics, present the pricing modes and known pricing estimates, and attempt to estimate the cost per-student, per-semester for two example courses. The focus of this manuscript is turn-key, ready-to-use, cloud-hosted cyber-range platforms that require no infrastructure setup or maintenance, that can be accessed with a web browser.

2 Available Cyber-Ranges

We conducted a search for cloud-hosted, publicly-available, education-oriented cyber-ranges. Our findings are presented in Table 1. This list of cyber-ranges is valid as of May 19 2022, and may be referred to as service or service providers.

List of Cloud-hosted, Publicly-available Cyber-Ranges for Educational Institutions					
Name	**CM**	**TL**	**SRM**	**LM**	**NL**
CompTIA [comptia.com]	N	SS	BI	ISM	NA
CyberBit [cyberbit.com]	F	H	NB	IM	1300+
EC-Council [eccouncil.org/]	N	SS	BI	ISM	NA
EDURange [edurange.org]	P	H	NB	ISM	8+
Hack The Box [hackthebox.org]	F	SG	NB	ISM	90+
Immersive Labs [immersivelabs.com]	F	H	NB	IM	1500+
Infosec Learning [infoseclearning.com]	F	H	BI	ISM	300+
Jones & Bartlett [jblearning.com]	F	H	BS	ISM	120+
NICE Challenge [nice-challenge.com]	F	H	NB	IM	70+
Project Ares [projectares.academy]	F	H	NB	ISM	400+
RangeForce [rangeforce.com]	F	H	NB	ISM	400+
Testout [testout.com]	P	SS	BI	ISM	120+
ThriveDX [thrivedx.com]	P	SG	NB	ISM	150+
Try Hack Me [tryhackme.com]	F	H	NB	ISM	500+
uCertify [ucertify.com]	F	SS	BI	ISM	NA

Table 1: Displays evaluated cyber-ranges and their characteristics. **KEY**: **Classroom Management (CM)**: classifiers *N*: None, *F*: Full, *P*: Partial. **Type of Labs (TL)**: classifiers *SS*: Step-by-Step, *H*: Hybrid, *SG*: Self-Guided. **Supplementary Reading Materials (SRM)**: classifiers *BI*: Book Included, *BS*: Book Separate, *NB*: No Book. **License Management (LM)**: classifiers *ISM*: Individual Student or Institution Managed, *IM*: Institution-Managed. **Number of Labs (NL)**: classifiers *NA*: Not Applicable, Number of Labs Available.

2.1 Characteristics of Available Cyber-Ranges

We characterize service providers using the following five criteria: (i) *Classroom Management*, (ii) *Type of Labs*, (iii) *Number of Labs*, (iv) *Supplementary Reading Materials*, and (v) *License Management*.

2.2 Classroom Management

Classroom Management criteria defines the service provider's available instructor ready student management tools. We evaluated the following classroom management tools including: (i) student progress tracking, (ii) teams creation, (iii) team progress tracking, and (iv) grade management. Student progress tracking refers to the ability to provide the instructor with information regarding a student's status in completing lab tasks. Teams creation refers to a group creation feature that the instructor can use to form teams and assign the teams a set of lab tasks. Team progress tracking refers to the tools provided for the instructor regarding a team's status in completing lab tasks. Grade Management refers to a module which can be used by the instructor to grade submitted lab task activities.

The classroom management criteria is classified as: (i) Full, (ii) Partial, and (iii) None. Full indicates the service provider supports all the listed criteria. Partial classifier indicates the service provider supports a portion of the listed criteria. None indicates the service provider does not support the listed criteria.

2.3 Type of Labs

Type of Labs criteria that define the nature of provided lab activities provided for both the students and the instructor include: (i) Self-Guided, (ii) Step-by-Step, and (iii) Hybrid. Self-Guided indicates the service provider has activities that are predominantly student driven, meaning the student is expected to perform the activities without being given an outright solution. Step-by-step indicates the service provider has activities that are predominantly driven by provided instructions, meaning the students are provided a significant amount of help in performing the activities. Hybrid indicates the service provider activities are a mix between the self-guided and step-by-step.

2.4 Supplementary Reading Materials

Supplementary Reading Materials criteria defines if additional reading resources are provided including: (i) Book Included, (ii) Book Separate, and (iii) No Book. Book Included indicates the service provider issues a print or eBook textbook to complement the hands-on labs. Book Separate indicates the supplementary reading material can be purchased for an additional fee. No Book indicates there is no supplementary reading material.

2.5 License Management

License Management criteria indicates the type of license management provided by the service provider, including: (i) Individual Student and (ii) Institution

Managed. Individual Student indicates each student purchases the service provider software license. Individual licenses are purchased via directly at the service provider's website or at an institution's bookstore.

Institution Managed refers to the department or the instructor purchasing all the licenses for a class. Once the licenses are purchased, the instructor will assign each individual a license. Advantages of institution managed licensing include, some services provide volume purchase discounts, and class roster changes are easily managed without a cost penalty to a student that drops. Institution managed licensing can be passed onto the students as course fees.

2.6 Number of Labs

Number of Labs criteria defines the number of lab activities provided by the service including: (i) Cloud, (ii) Local Installation Service Provider Software, and (iii) Local Installation No Service Provider Software. Cloud activities are performed on remotely hosted system environments, and are potentially accessed via a web browser. Local Installation Service Provider Software activities require the user to install the activities environment on a non-service provider local system. Local Installation No Service Provider Software are activities the user conducts on a local machine without installation of any service provider software.

3 Pricing Modes and Costs of Available Cyber-Ranges

We discuss pricing mode criteria and costs for the service providers discussed in Section 2. The information provided in the section lists the pricing modes and costs for the cyber-ranges listed in Table 1.

3.1 Pricing Modes

In our survey we encountered three types of pricing modes, (i) Ranged, (ii) Fixed, and (iii) No Cost. Service providers final costs for Ranged pricing mode are dependent on several factors including number of modules, number of labs, number of students, type of licensing, and type of classroom management. Ranged classifiers include: Ranged, based on modules selected; Ranged, based on the number of students; Ranged, based on the number of modules (labs) and/or number of students selected; and Ranged, based on the tier selected. Institutions Service providers final costs for Fixed pricing mode represents a singular cost. A few services are available at no cost for US-based non-profit institutions.

Approximate Costs of Cyber-Ranges listed in Table 1		
Name	**PM**	**Costs (40 students)**
CompTIA	RM	$180 to $240 /student/year
CyberBit	RMS	$1200 to $1500 /student/year
EC-Council	RM	$50 to $1200 /student/year
EDURange	NC	$0
Hack The Box	RT	$117 /student/3 months
Immersive Labs	RMS	$40k to $150k /class/year
Infosec Learning	RMS	$800 /student/year
Jones & Bartlett Learning	RM	$100 to $350 /student/year
NICE Challenge Project	NC	$0
Project Ares	RM	$15 to $225 /student/month
RangeForce	RS	$500 /class/year
Testout	FC	$120 /student/18 months
ThriveDX	FC	$100 /student/5 months
Try Hack Me	RMS	$10 to $25 /student/month
uCertify	FC	$140 /student/18 months

Table 2: Displays the pricing mode and costs for the cyber-ranges listed in Table 1. **KEY: *P*ricing Mode (PM)**: Classifiers *RM*: Ranged, Modules Selected; *RS*: Ranged, Number of Students; *RMS*: Ranged, Number of Modules and/or Number of Students; *RT*: Ranged, Tier Selected; *FC*: Fixed Costs; *NC*: No Cost.

3.2 Costs

Costs listed include customized quotes from some of the service providers. Such customized cost quotes vary from one institution to another due to factors, such as number of students, number of labs, number of courses, and length of the contract. We encourage interested instructors to contact the respective service providers directly without harboring any cost expectations.

4 Two Course Case Studies

In this subsection we introduce two case studies and an estimate of the hands-on activities for each course. The purpose of these case studies is to allow us to compare offerings and corresponding per-student, per-course, cost estimates of using the described cyber-ranges.

As part of the pricing model, most commercial cyber-ranges will provide customized content for each course based on the cybersecurity objectives of the course. Furthermore, most of the listed cyber-ranges in Table 1 have some

free content that is limited. For example, RangeForce has a free community edition with a total of 30 modules. These modules provide limited hands-om labs for Orange Team, Yellow Team, Blue Team, and Purple Team.

The first case study is for a freshman/sophomore course where the students have no/limited cybersecurity knowledge. The second case study is for a senior level network security course, that has prerequisites of the course from the first case study, and networking background.

In order to normalize the student cost/labs across both quarter and semester schools, the case studies are built on a 12-week course. Furthermore, each course has objectives that are mapped to the NIST/NICE framework. The details for each course are discussed below. NOTE: the course numbers and titles are fictional. The NICE/NIST knowledge units can be found at: `https://dl.dod.cyber.mil/wp-content/uploads/cae/pdf/unclass-cae-cd_ku.pdf`

4.1 Case Study 1: Cybersecurity Fundamentals

This course presumes the students have no cybersecurity experience. The course objectives are aligned with the Center of Academic Excellence in Cybersecurity outcomes of: Cybersecurity Foundations, Cybersecurity Principles, IT System Components, and Basic Cryptography.

4.1.1 Case Study 1: Cyber Ranges Utilized

The content for this course is built around CompTIA Security+. The students in this course require significant instruction and significant direction when it comes to labs. The hands-on labs are structured based on the free version of Immersive Labs and the paid content of uCertify. The students purchased uCertify out of pocket as the required textbook.

The Immersive Labs' free content requires the students to complete a number of interactive hands-on labs outside of class. Some of the labs are step-by-step, and a majority of the labs are hybrid. The students create a free account, they enroll and complete the assigned labs, and they submit a PDF that is a screen capture of the completed lab in the appropriate Learning Management System (LMS). The grading is usually conducted on a scale where students receive a 0 for no submission, a 1 for an incomplete lab, and a 2 for a fully completed lab. Immersive Labs and uCertify were chosen based on cost and the combination of step-by-step and hybrid labs.

The uCertify content is all inclusive, including an online textbook, labs that correspond to the chapters in the textbook, quizzes, and flashcards. uCertify seamlessly integrates with the popular LMS. uCertify also has multiple pretests, and post-tests to assess the students' knowledge and to prepare them for the actual exam if the students so desire.

4.1.2 Case Study 1: Cyber Range Costs

For this course the six month version of uCertify was chosen for approximately $84 per student. The course was structured around the students reading the chapter content before class, and then the instructor supplementing the material with lecture. The uCertify labs were assigned as both in-class labs, where the instructor was present to answer questions, and take home labs, where the students logged in from home on their own time and completed the lab. In either scenario, once the student completed the lab it automatically updated in the LMS. Based on the readings, the lectures, and the labs, both uCertify quizzes and instructor made quizzes were administered.

4.1.3 Case Study 1: Cyber Range Evaluation

The use of Immersive Labs helped introduce the basic concepts. The use of uCertify then solidified those basic concepts through the uCertify labs. The uCertify labs were basic and step-by-step; however, since this was the students' first exposure, the labs satisfied the learning outcome expectations of an introductory course. One problem was the uCertify access through the web browser was slow. Another problem was integration of the LMS and uCertify. The uCertify LMS interface was difficult to use.

4.2 Case Study 2: Network Security

This course presumes the students have extensive cybersecurity experience, including all the prerequisites. The course objectives are aligned with the CAE outcomes of: Network Defense (NDF), Network Technology and Protocols (NTP), and Penetration Testing (PTT).

4.2.1 Case Study 2: Cyber Ranges Utilized

The content for this course is built around traditional network security topics like the SEED labs [2] and basic penetration testing concepts. The course usually starts with a review of networking concepts, specifically networking technologies and protocols, including the protocol weaknesses. The students in this course don't require significant instruction or significant direction related to the labs. HTB was chosen for the cost and the learning academy. The cost per license was $90 and we paid from course fees. There was no out of pocket student expense.

The students completed a set number of learning modules from the Academy subscription. Like the Immersive Labs from Case Study 1, there was no LMS integration; therefore, students were required to submit a PDF. For the Enterprise HTB, the students were assigned dedicated boxes that required the

students conduct penetration testing based on the Academy learning modules. The instructor occasionally lectured, usually offering a walk through of a dedicated box based on the Academy learning concepts. The students were required to submit a PDF screen capture illustrating their progress on the dedicated box.

4.2.2 Case Study 2: Cyber Range Costs

The three month subscription to Hack The Box (HTB) was chosen. HTB requires two different subscriptions. The students are required to purchase the monthly $18 subscription from Academy HTB. For the dedicated boxes for the penetration testing portion, the department purchased the Enterprise HTB licenses for $90 per student.

4.2.3 Case Study 2: Cyber Range Evaluation

One regret we have from this case study is we did not require formalized penetration testing write-ups explaining the processes the students used to penetration test the box.

Responses to the end-of-quarter course evaluation survey showed the students found the HTB content to be beneficial in achieving course learning objectives. The students enjoyed trying to penetration test the box to discover the hidden flag. The students were split on the Academy learning modules, based on the required time to read and complete the hands-on labs. For the instructor there was no LMS integration. The HTB management interface was easy to use; however, if a student encountered a problem with their license, trouble shooting was slow and cumbersome due to the time differences between the US and the United Kingdom. A difficult challenge was finding the correct balance between lecture and self-study based on the assigned HTB Academy modules. HTB would be beneficial to students that work well independently over a lecture based course such as this case study.

5 Conclusion

It is important to disclose that what was presented above is a potential example of usage. Other courses designed in a different way and/or with different assignment of activities would result in a different pricing outcome. We strongly suggest not to interpret the data shown above as the authors' recommendation for any one particular cyber-range. Each of the presented cyber-ranges provide different offerings with different cost and functionalities and very disparate pricing modes. The answer to the question of which offering is the optimal of-

fering for a given course will depend heavily on the length of the course and the particular selection of activities.

In this manuscript we investigated, classified and analyzed turn-key, ready-to-use, cloud-hosted cyber-range platforms that require no infrastructure setup or maintenance. Our conclusion is, this classification and analysis should be beneficial to small to medium sized colleges and universities that have limited faculty, and resources needed to provided sufficient hands-on labs to its cybersecurity students. Ultimately it is the decision of each university on how the cost per student will be amortized across the student population. Future work includes a comparative analysis of the listed services as observed during deployment in additional courses.

It is our goal that this foundation survey benefits every school looking to incorporate hands-on labs in their courses without having to create, deploy, maintain and update the labs.

6 Disclosures and Acknowledgements

The authors certify that they are not affiliated with any of the cyber-ranges described in this article or their sponsoring organizations or companies, other than having evaluated or used some of these offerings in some of our courses.

We would like to thank staff representatives from services listed in Table 1 for providing us with demonstrations and other relevant information regarding their respective services.

References

[1] Daniel Conte de Leon, Ananth A. Jillepalli, Victor J. House, Jim Alves-Foss, and Frederick T. Sheldon. Tutorials and Laboratory for Hands-On OS Cybersecurity Instruction. *Journal of Computing Sciences in Colleges*, 34(1), October 2018.

[2] Wenliang Du. Seed: Hands-on lab exercises for computer security education. *IEEE Security and Privacy Magazine*, 9(5):70–73, 2011.

[3] Ananth A. Jillepalli, Daniel Conte de Leon, and Frederick T. Sheldon. CERES NetSec: Hands-on Network Security Tutorials. *Journal of Computing Sciences in Colleges*, 33(5):88–96, May 2018.

[4] Vasileios Mavroeidis and Audun Jøsang. Data-driven threat hunting using sysmon. In *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*. ACM, 2018.

Hands-On SQL Injection in the Classroom: Lessons Learned*

Jens Mache[1], Carlos García Morán[1], Nic Richardson[1], Wyeth Greenlaw Rollins[1], Richard Weiss[2]
[1]Lewis & Clark College, Portland, OR 97219
{jmache, carlos, nrichardson, wyethg}@lclark.edu
[2]The Evergreen State College, Olympia, WA 98505
weissr@evergreen.edu

Abstract

SQL injections remain a serious security threat to applications using databases. In this experience paper, we report on teaching SQL injection hands-on using the EDURange platform in two different undergraduate courses, Web Development and Databases. We analyze the results from a voluntary survey with answers from 17 students who took the Web Development course and from 8 students who took the Database course. We focus our discussion around several lessons we learned, including the importance of guiding questions, covering unions and padding, and how to deal with the possibility of students adversely modifying the learning environment.

1 Introduction

Web-facing applications are common targets for attackers who seek to expose sensitive information such as passwords or credit card information. The list of the top ten security risks in 2021 at the Open Web Application Security Project (OWASP) [6] shows injection vulnerabilities as number three. This category includes SQL injection. Injection attacks can occur when unsanitized user

input is included in executed statements. These types of attacks are often used to gain access to sensitive information in SQL databases. Taylor et al. [8] have surveyed seven popular database textbooks and found that most of them do not cover this topic adequately. To address this problem, we have developed an exercise to teach students how SQL injections work and how to prevent them, which were the learning goals for the exercise we created.

One could discuss SQL injection attacks in class, but hands-on exercises are critical for learning and engagement. In this experience paper, we explore hands-on teaching platforms and discuss our experience teaching SQL injection using the EDURange platform [10, 2]. Our main goal was to explore how to teach SQL injection and what the obstacles might be.

2 Related Work

Numerous hands-on exercises have been created to aid the teaching of SQL injection. Yuan [11] describes eight web security labs, three of which deal with SQL injection. However, they are primarily focused on vulnerability detection and testing. They found that the "real-world" aspects of the exercises made the material more interesting to students. Du [3] outlines the labs of the SEED project, one of which focuses on SQL injections.The EDURange version goes beyond these attacks and can also run in the cloud or locally. Most important is that EDURange exercises are extensible by the instructor.

Li et al [4] outline eight security labs that can be played in two modes: war mode and peace mode. War mode has students work as ethical attackers while peace mode has students attempt to prevent attacks. However, the war mode has not been developed for the SQL injection lab.

The Web Application Hacker's Handbook and the accompanying Portswigger website [7] have an extensive set of exercises. They provide an overview of SQL injection that includes an injection cheat sheet and a video explanation of the topic. They also provide injection samples to accomplish different goals such as retrieving hidden data, subverting application logic, UNION attacks, examining the database, and blind SQL injection. Alongside the explanatory information, they provide 16 labs. Each lab contains the author's solution as well as community solutions in the form of videos. A few of the labs require the user to accomplish the same goal on different SQL implementations (MySQL, Oracle, etc). While focusing on multiple implementations might be appropriate for deeper dives into SQL, this can cause confusion in introductory SQL exercises. The Portswigger labs [7] require Burp Suite or other tools to complete the later labs. Burp Suite is a web security testing tool with the ability to intercept and modify HTTP requests made by the client. This capability is required in order to complete Portswigger's blind SQL injection labs. However,

this could cause unnecessary friction during in-class activities. Portswigger contains a large body of work and might be too much to be included in Web or database classes. The EDURange containers provide all of the necessary software tools and tries to pare down the scope of the exercise while including some complex but important examples.

Basit [1] created a platform with 12 challenges to introduce students to SQL injection. Unfortunately, the levels hosted at `www.databases.cs.virginia.edu/sqlinject/` are no longer available. They attempt to structure the levels in a way that each level builds on the previous. Level one asks students to create a SQL injection that will expose all usernames contained within a "users" table. The hint gives students the answer. Level two has the student create a SQL injection that exposes the names of other tables in the database. This is a big leap. It requires the student to be familiar with the implementation-specific SQL table called information_schema.tables. EDURange addresses some of the problems of bridging the gaps between levels by using guiding questions. In addition, EDURange is flexible. Instructors can easily modify exercises by adding their own questions or by inserting/deleting levels based on classroom data that is collected and student feedback.

3 Levels and courses

The goal of the EDURange SQL CTF (capture the flag) exercise is to teach students why SQL injections are important, how they work, and how to prevent them. It applies to any web application with a database back end.

Our exercise has three levels. Students are given the following information: the queries for level 1 (SELECT * FROM countries WHERE name='<ARG>';) and level 2 (SELECT * FROM books WHERE author LIKE '%<ARG>%';), and a hint for level 3 ("count the number of columns in the table").

Fig. 1 shows level 3 after input without SQL injection. Fig. 2 shows level 3 after a successful SQL injection.

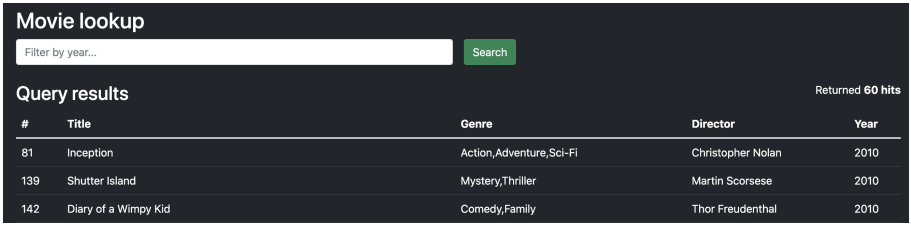


Figure 1: Level 3 after '2010' has been entered.

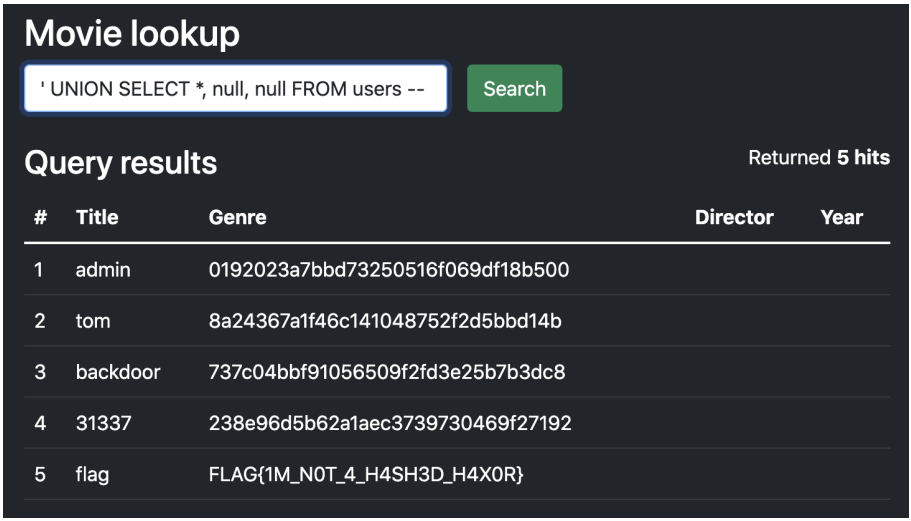


Figure 2: Level 3 after a successful SQL injection.

Students were also given the following guiding questions:

1. What is a "comment" symbol?
2. What boolean operator did you use to dump the table from level 1?
3. What is the flag for level 1?
4. What special character is the LIKE query using in level 2?
5. What is the flag for level 2?
6. What keyword could you use to query two combined tables?
7. What is the flag for level 3?
8. What is the password for user "backdoor"?

In Spring of 2022, we taught SQL injection in two undergraduate courses, web development and databases, using the EDURange platform. Students in the database class had spent several weeks writing SQL queries, while those in web development only had about one week. The same instructor introduced the same exercise in both classes.

In the web development course, students had an hour long class period on one day to mostly independently try to answer the questions. They were asked

to spend an additional hour after class to continue exploration. The next two class days provided more time and review, but also introduced another topic.

In the database course, one of the co-authors of this paper (who had been sitting in the back for most of the term) was invited to give a guest appearance. The same SQL injection exercise was used. Since the time available was limited to about 40 minutes of one class day, the instructor showed a guided walk-through in which the students could follow along.

4 Student survey

In both courses, in an anonymous survey, students were asked six questions on a 5-point Likert scale. They were also asked the two following open-ended questions: "What problems did you encounter in completing the lab?" and "What changes could be made to the lab to enhance your learning?"

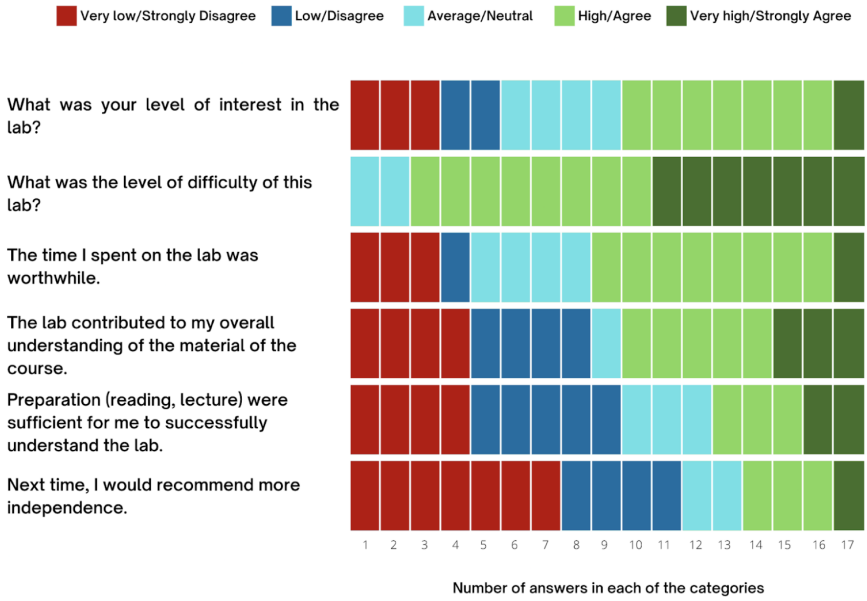


Figure 3: Survey results from the web development course.

We received responses from 17 students in the Web Development course and 8 students in the Database course. Most students in the Web Development course found the activity to be interesting and challenging. Looking at the graph, a theme emerges suggesting students in this course wanted more

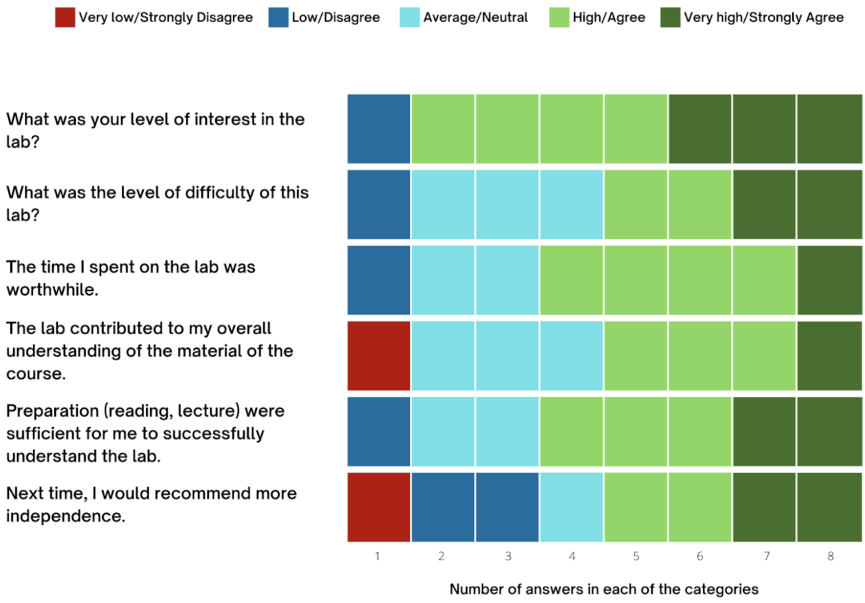


Figure 4: Survey results from the databases course.

guidance before and during the activity. In the two open-ended questions, Web Development students reported that they felt confused or "unsure what to do a lot of the time."

In contrast, more students in the databases course agreed that the activity was interesting and helpful to build their understanding of the course material. The database students' answers to the open-ended questions suggest they were less confused about how to complete the exercises. There was a higher demand for independence rather than guidance during the exercises. One student responded saying they would have liked "[m]ore time to sit and think about how to solve the problems, [and] less time spent being told how to do them."

5 Lessons Learned

5.1 Guiding questions

This feedback from the students helped to inform our lessons learned. Specifically, it helped to demonstrate the importance of guiding questions in leading students through the exercises. Where the web development class could have used better hints and guiding questions, the database students could have ben-

efited from more variety in the types of SQL injections used and more time to solve the problems on their own. It is clear that platforms used to teach SQL injection must be flexible. Instructors should be able to adjust questions to match the level of difficulty and guidance which will suit the students best. The EDURange platform [2] now includes an editable YAML file which instructors can use to change and rearrange the guiding questions associated with the exercise. Guiding questions are especially useful for advancing to levels where it is difficult to get the solution by trial and error, as in the case of padding (to force matching numbers of columns in an UNION statement).

5.2 UNION attacks and padding

The UNION operator is used to combine the result-set of two or more SELECT statements. It can be used to spill secrets from tables other than the one in the original query. However, to use UNION, each SELECT statement must produce the same number of columns with the same data types. The former can require padding, a topic that some students may be unfamiliar with and find difficult to grasp. We have seen multiple approaches to padding. Some exercises avoid padding altogether by only using a single column while others try to demonstrate the best methods for finding a table's dimensions. We believe that it is important to address padding as it is often needed to display data on the site that is being attacked. We feel that it is most beneficial to start with a simplified example using only one or two columns (to avoid the need for padding), before demonstrating methods to count columns for padding.

5.3 Progression of levels

Level 1 of our exercise introduces comments, quotes and logic. These rudimentary parts of SQL injections are easy to utilize. With this knowledge students can start to retrieve unintended information and gain unwanted access. After students have used these ideas, they are ready to bypass common prevention measures such as blocking all comment symbols. This leads to an opportunity for students to learn that different representations (like ASCII) for these characters can be used in their place. Next, students can be introduced to queries that use the LIKE clause and how wildcards can be used to match common names of tables, columns, etc. After these concepts, students can begin forming injections that query information about the database itself. This may include the version, table names, and the number of rows or columns. At this point, padding should be discussed as it is common for the number or type of columns returned by the injection to not match those of the original query. It is our opinion these introductory exercises should only use one SQL flavor as the use of multiple may cause unnecessary confusion. Guiding questions should be provided for each level.

5.4 How to handle "DROP TABLE"

After students learn these new and intriguing techniques, our hope is that they will get satisfaction out of putting them into practice and experimenting with new ideas. Unfortunately, the place students may attempt this experimentation is on our site (that hosts the hands-on exercise). This can cause problems if the appropriate precautions are not taken. One of the more harsh injections a testbed needs to be ready for is students attempting to drop tables. This is something that we experienced during the first iteration of our exercise. Fig. 5 shows the apologetic email we got from a student.

I dropped the main data table from my webfu page. Whoops. I couldn't figure out what I was supposed to do, so I thought I should try breaking it! I guess I tried too hard.

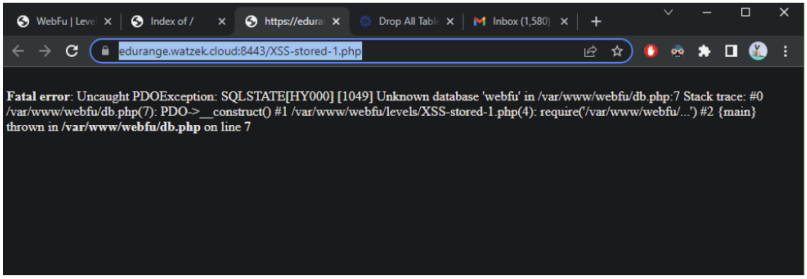


Figure 5: Email from a student.

How to combat this while still giving students the opportunity to experiment with SQL injections? One resource-intensive option would be to have individual containers or databases for every student. This would allow students to experience destruction without damaging the learning environment for other students. In our current setup, we try to detect and prevent the DROP command and similar statement that alter the state of the database. We do notify the student when we detect a potentially destructive command. At any rate, it is recommended to be ready (with scripts) to rebuild the learning environment to its original state.

5.5 Learning Environment

We believe an important part of a learning environment includes hands-on activities. Hands-on activities give students many opportunities to actively participate in the classroom and aid their learning. With interactive platforms,

we can allow students to test their knowledge immediately and help them retain the knowledge and get assistance if needed. Cloud-based learning platforms eliminate barriers to entry for students as well as many compatibility issues which could limit students. Often with local environments, a lot of the student's time could easily be taken up by attempting to set up the environment instead of being able to practice the material at hand. Additionally, we believe that gamification is a very strong way to maintain student interest and help them enjoy the material. Some common ways we implemented gamification is by giving the students tasks to find specific information within the site such as flags and passwords. Future work would be to turn this into a Red Team/Blue Team exercise by allowing students to modify code.

5.6 Logging student activity

We wanted to log user input within the scenario to see how students were approaching problems and to identify common misunderstandings or struggles for students within the testbed. EDURange has the capability to capture student actions [5]. Theoretically students could be individually identified by their session. However, this can be blocked by using incognito browser tabs. Indeed, a browser assigns a new session to each new incognito tab that the student may open, causing the logger into treating each session as a distinct user. Once we are able to consistently distinguish between users, we will be able to assign logged activities to participants accurately. This can help us improve feedback to students that are struggling with any part of the activity. In the future, we plan to extend our previous work [9] and use machine learning to identify these patterns and determine the best time to help students through the activity with a helpful hint.

336	eyJfZnJlc2giC	SQL-1	Cuba	[{\id\":\"57\"	\"name\":\"Cuba\"	\"code\":\"CU\"}]"
337	eyJfZnJlc2giC	SQL-1	Cuba\'--	[]	PDOException: SQLSTATE[42C	2022-04-22T13:01:01-07:00
338	eyJfZnJlc2giC	SQL-1	Cuba\' OR 1-1	[]	PDOException: SQLSTATE[42C	2022-04-22T13:01:05-07:00
339	.eJy1kc1KA0E	SQL-1		[]	NULL	2022-04-22T13:06:33-07:00
340	.eJy1kc1KA0E	SQL-1	\' OR 1=1;--	[{\id\":\"1\"	\"name\":\"Afghanistan\"	\"code\":\"AF\"}
341	.eJzF0sFKAzE	SQL-2		[]	NULL	2022-04-22T13:06:59-07:00
342	.eJzF0sFKAzE	SQL-2	%	[{\id\":\"1\"	\"title\":\"Brave New World\	\"author\":\"Aldous Huxley\'
343	.eJyt081KAzE	SQL-1		[]	NULL	2022-04-22T13:07:17-07:00
344	.eJyt081KAzE	SQL-1	Cuba	[{\id\":\"57\"	\"name\":\"Cuba\"	\"code\":\"CU\"}]"
345	.eJzF0s1KAzE	SQL-3		[]	NULL	2022-04-22T13:07:23-07:00
346	.eJzF0s1KAzE	SQL-3	\' UNION SELECT table_name FROM information_schema.tables;--	[]	PDOException: SQLSTATE[21C	2022-04-22T13:07:29-07:00

Figure 6: Excerpt from the log file showing line number, session id (truncated), exercise level, user input, output (if any), error (if any), timestamp.

6 Conclusion and Future Work

Using hands-on exercises in the EDURange platform, we integrated the security topic of SQL injection into two different non-security undergraduate courses, Web Development and Databases. We collected feedback from student activity logs, a voluntary survey, and from direct communication with the students.

Many students had questions about UNION attacks. These attacks are often used to retrieve data from other tables, e.g. passwords of users. In general, one needs to use padding for this. Based on student feedback, we plan to improve the exercise in two ways. We would add a preparatory level that uses UNION without padding, and we would add more guiding questions on this topic.

Even if the exercise doesn't call for it, curious students may experiment with the "DROP TABLE" statement and thus destroy the learning environment. We discussed several ways in which this could be handled.

Future work includes using machine learning to identify students having trouble, especially with padding and to evaluate which hints to give and when.

7 Additional Information

We contacted our IRB and got exempt status. We plan to make some of the artifacts available to the community. We would like to especially thank Alain Kaegi. This work was supported by the National Science Foundation under grants 2216485 and 2216492.

References

[1] Nada Basit et al. "A Learning Platform for SQL Injection". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE '19. Minneapolis, MN, USA: Association for Computing Machinery, 2019, pp. 184–190. ISBN: 9781450358903. DOI: 10.1145/3287324.3287490. URL: https://doi.org/10.1145/3287324.3287490.

[2] Jack Cook et al. "An authoring process to construct docker containers to help instructors develop cybersecurity exercises". In: *Journal of Computing Sciences in Colleges* 37.10 (2022), pp. 37–47.

[3] Wenliang Du. "SEED: Hands-On Lab Exercises for Computer Security Education". In: *IEEE Security & Privacy* 9.5 (2011), pp. 70–73. DOI: 10.1109/MSP.2011.139.

[4] Lei Li et al. “Developing Hands-on Labware for Emerging Database Security”. In: *Proceedings of the 17th Annual Conference on Information Technology Education*. SIGITE ’16. Boston, Massachusetts, USA: Association for Computing Machinery, 2016, pp. 60–64. ISBN: 9781450344524. DOI: 10.1145/2978192.2978225. URL: https://doi.org/10.1145/2978192.2978225.

[5] Jelena Mirkovic et al. “Using terminal histories to monitor student progress on hands-on exercises”. In: *Proceedings of the 51st ACM technical symposium on computer science education*. 2020, pp. 866–872.

[6] *OWASP*. 2022. URL: https://owasp.org/www-project-top-ten/.

[7] *Portswigger*. 2022. URL: https://portswigger.net/web-security/sql-injection.

[8] Cynthia Taylor and Saheel Sakharkar. “’);DROP TABLE Textbooks;–: An Argument for SQL Injection Coverage in Database Textbooks”. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE ’19. Minneapolis, MN, USA: Association for Computing Machinery, 2019, pp. 191–197. ISBN: 9781450358903. DOI: 10.1145/3287324.3287429. URL: https://doi.org/10.1145/3287324.3287429.

[9] Quinn Vinlove, Jens Mache, and Richard Weiss. “Predicting student success in cybersecurity exercises with a support vector classifier”. In: *Journal of computing sciences in colleges* 36.1 (2020).

[10] Richard Weiss et al. “Finding the balance between guidance and independence in cybersecurity exercises”. In: *2016 USENIX Workshop on Advances in Security Education (ASE 16)*. 2016.

[11] Xiaohong Yuan et al. “Hands-on Laboratory Exercises for Teaching Software Security”. In: *Proceedings of the 16th Colloquium for Information System Security Education* (2012).

A Systematic Review on the Effectiveness of Programming Camps on Middle School Students' Programming Knowledge and Attitudes of Computing*

Carla De Lira[1], Rachel Wong[2], and Olusola Adesope[3]
[1]Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99163
carla.delira@wsu.edu
[2]Theory and Practice in Teacher Education
University of Tennessee, Knoxville, TN 37996
rwong2@utk.edu
[3]Educational Psychology
Washington State University, Pullman, WA 99163
olusola.adesope@wsu.edu

Abstract

Computer science (CS) outreach during K-12 grades plays a huge role in students' interest to pursue computing as a career. With the current 13 percent projected growth of computing occupations from 2020-2030, the demand for CS graduates has remained steady. Programming camps provide the ability to engage K-12 students in computing activities with the hopes of making a short-term program have a lasting impact on their student attitudes and understanding of the tech field. The main purpose of this systematic review is to evaluate the current state of programming camps for middle school students and their effectiveness in supporting positive student attitudes and increase in programming knowledge. This review provides connections to current CS education research and makes recommendations on future reporting of programming camp-related studies.

1 Introduction and Related Work

Programming camps are a type of outreach opportunity for K-12 students to engage with programming and computing concepts via project-based and experiential-based learning outside of regular classroom hours [**3**, **15**, **21**, **23**, **25**]. Short-term CS opportunities could provide a long-lasting impact on students' perceptions of CS [**15**, **17**]. Although programming camps seem to have the components necessary to foster interest in CS in a shorter period, much work is needed to systematically evaluate curriculum and camp organizing decisions on middle school student attitudes towards CS and learning how to code.

To date, there has been one systematic review of CS outreach programs [**7**] that centers primarily on what data has been collected and the general goal of the outreach program, such as increasing engagement in programming students belonging to underrepresented groups. There are no systematic reviews that outline what programming camp organizing decisions, such as curriculum and logistics, are most effective in improving student attitudes towards programming and programming knowledge. Thus, this systematic review seeks to evaluate the current state of programming camps for middle school students and their effectiveness in supporting positive student attitudes and increased programming knowledge. This systematic review explores the current state of the literature on programming camps for middle school students through the following research questions:

1. How effective are programming camps in improving programming knowledge and/or computational thinking skills?
2. How effective are programming camps in improving student attitudes towards computing?

2 Method

This systematic review utilizes the Preferred Reporting Items for Systematic Reviews and Meta-analyses (PRISMA) guidelines [**8**]. PRISMA guidelines were chosen to promote transparency and reproducibility of article retrieval and coding process.

2.1 Search Terms Structure

The keywords were structured to ensure sensitivity in retrieving as many programming camp-related studies about middle school students as possible:

- Included Independent variable (Programming Camp) terms: computer science camp OR computer science camps OR IT camp OR IT camps OR programming camps OR programming camp OR code camps OR code camp OR coding camp OR coding camps OR computing camp OR computing camps OR tech camp OR tech camps OR technology camp OR technology camps.
- Included Population terms: middle school* OR junior high school* OR eighth grade* OR sixth grade* OR seventh grade* OR 6-8 grade* OR grade 6-8 OR grade 6 OR grade 7 OR grade 8 OR 8th grade OR 7th grade OR 6th grade
- Excluded Population terms: elementary school OR grammar school OR primary school OR grade 1 OR grade 2 OR grade 3 OR grade 4 OR grade 5 OR grade 9 OR grade 10 OR grade 11 OR grade 12 OR kindergarten* OR adult school OR first grade* OR second grade* OR third grade* OR fourth grade* OR fifth grade* OR ninth grade* OR tenth grade* OR eleventh grade* OR twelfth grade* OR high school* OR secondary school* AND preschool* OR early childhood OR 1st grade OR 2nd grade OR 3rd grade OR 4th grade OR 5th grade OR 9th grade OR 10th grade OR 11th grade OR 12th grade

2.2 Electronic Databases Search

Articles were retrieved from five databases: ERIC ProQuest, Web of Science, PsycInfo, ACM Digital Library, and IEEE Xplore Search. Since there was a previously published systematic literature review on computer science outreach this search was filtered from 2016 to 2021. All initial articles were retrieved on September 25, 2021.

2.3 Study Selection

Sixty-five duplicate papers were first removed. Then, studies were screened in two phases. In phase one, the titles and abstracts were screened by a single reviewer and 2509 studies were excluded. In phase two, the full-texts of 112 studies were retrieved and evaluated. A total of 14 studies were retained for inclusion in this systematic review [**1**, **6**, **8**, **11**, **14**, **17**, **21**, **22**, **24**, **25**].

2.4 Data Items and Collection Process

Eligible papers were coded for programming knowledge/CT skill development or student attitude change pre- and post-programming camp. The data extraction focused primarily on programming camp logistics, paper characteristics,

and data to answer the research questions. Further coding efforts were made to create broader categories for each variable for the purpose of analysis.

2.5 Data Analysis

For each coded category, frequencies and percentages were calculated. To uncover patterns on several data extracted and coded, fine grade data extraction columns and filtering were used to identify opportunities for cross variable analysis and sorting based on subcategories in each variable. For cross variable analysis, frequencies and percentages were calculated.

3 Results

To answer our research questions, we focused our analysis of 16 within-group studies on the organizing decision characteristics of programming camps and the effectiveness of the programming camps on programming knowledge.

3.1 Organizing Decision Characteristics of Programming Camp

Regarding programming camp teaching decisions, such as teaching approach, project-based was the most common approach in programming camps (n = 8). This is followed by the hands-on scaffolded approach (n= 4). Three studies utilized a mixed approach. Two of three studies which utilized a mixed approach implemented project-based learning with unplugged activities. Only one study did not report their teaching approach.

3.2 Programming Camp Effectiveness on Programming Knowledge

Six studies addressed programming knowledge or computational thinking skills, while ten studies did not. For the studies that examined these outcomes, five saw an improvement in programming or computational thinking skills. One study had inconclusive findings.

Although most programming camps did not focus explicitly on assessing knowledge or computational thinking skills, there was effective reporting on the languages and concepts covered. Block programming (i.e., MIT App Inventor and Scratch) was the most common programming language format (n = 7), followed by a combination of block and text-based programming languages (n = 5). Only one study used an offline method through unplugged programming and one study did not report the programming language or platform used.

Further analysis was done to examine potential patterns between programming languages and any other variables relevant to programming camp logis-

tics. Figure 1 provides an overview of programming camp logistics as it relates to the choice of programming language.

Category	Subcategories	Block Programming	Text-based Programming	Both	Other	Not Reported
Length of Camp	1-5 Days	5	N/A	N/A	N/A	N/A
	6-10 Days	2	1	4	1	1
	1+ Days	N/A	1	1	N/A	N/A
Length Per Day	1-3 Hours	3	1	N/A	N/A	N/A
	4+ Hours	3	N/A	1	N/A	N/A
	None Reported	1	1	4	1	1
Teaching Approach	Project-based	2	1	3	1	1
	Mixed Approach	1	0	2	0	0
	Hands-on scaffolded approach	3	1	0	0	0
	Not reported	1	0	0	0	0

Figure 1: Overview of Programming Language Usage Frequencies

Seven studies did not explicitly report specific programming or computational thinking concepts. However, nine studies did report concept coverage for programming and computational thinking, such as variables, loops, control structures, data structures, debugging, and computational thinking.

3.3 Programming Camp Effectiveness on Student Attitudes

Fourteen studies observed improvements in student attitudes towards computing. Eight studies saw an increase in positive CS perceptions. All 14 studies observed an increase in positive student attitudes such as increased interest in CS, self-efficacy in CS, interest in STEM, and engagement in learning CS.

Figure 2 provides an overview of what programming camp logistics may have played a factor in the two commonly investigated student outcomes, CS perceptions and interest.

3.4 Effectiveness on Programming Knowledge

The results on programming knowledge and computational thinking skills provide directions on how to explore these student outcomes in programming camps. In general, the 5/6 studies which investigated programming knowledge and computational thinking saw an improvement. The other eligible studies only collected pre- and post-camp data to assess programming knowledge or computational thinking skills. Fields et al.'s study was the only study among 6 which investigated the learning progress of middle school students from their first day to last day. The main reason why their results were inconclusive

Category	Subcategories	Increased positive CS perceptions	Increased interest in CS
Length of Camp	1-5 Days	1	2
	6-10 Days	5	1
	11+ Days	2	0
Length Per Day	1-3 Hours	1	2
	4+ Hours	2	0
	Not Reported	5	0
Teaching Approach	Project-based	3	11
	Mixed	2	0
	Hands-on scaffolded	2	2
	Not Reported	1	0

Figure 2: Overview of Logistics and Student Attitudes

was their observation data showed peaks of collective understanding of easier material on day 1 to gradually harder concepts leading up to day 5 of their programming camp [**10**]. This suggests that organizers and scholars should measure programming knowledge throughout the camp to better assess and improve on curriculum designs for future programming camps.

Many papers provided robust details on programming language and concept coverage. Block programming is the most used programming camp language, since it is geared towards novice programmers due to its visual, easy-to-use interfaces [**9**, **11**, **25**]. Surprisingly, the next most common programming language approach is a combination of block programming like Scratch [**11**, **25**] and text-based programming like Python [**1**]. According to literature, jumping from block to text-based programming may require time to do [**16**]. However, results show that studies that introduced both block and text-based programming took 6-10 days; none introduced them within 1-5 days. Indicative that programming camps that have middle schools' students code in both likely run on the longer side to ensure that students do not become overwhelmed by a quick transition into text-based programming.

In terms of concept coverage, core concepts like variables and control structures were covered and reported explicitly in the eligible studies. These are concepts that have been commonly covered in current introductory courses or learning opportunities for middle school students [**12**]. However, computational thinking skills are not widely introduced in recent programming camps, and coverage of sub-concepts varied from study to study [**10**, **24**]. Debugging was another concept that was not widely introduced in recent programming camps.

3.5 Effectiveness on Student Attitudes

Perception of CS was the most assessed student attitude sub-construct among the eligible studies. All studies that investigated CS perception observed an increase of positive perceptions. Especially for middle school students [**5**, **13**, **20**], this is a promising potential long-term impact on their decision to pursue CS [**15**, **17**].

The decision for a vast majority of programming camps to utilize a project-based approach is not surprising. Project-based learning has been shown to be effective for motivating students to solve real-world problems by creating apps that impact a community [**15**] or designing robots to perform useful tasks [**21**]. Further, project-based learning thrives in a heavily social environment [**19**], which can support students' perceptions of CS away from the stereotypes commonly associated with a computer, such as impersonal, intimidating, and isolating [**19**].

3.6 On Reporting Studies

Many decisions about organizing the programming camp logistics remain largely inconclusive due to the lack of information available about basic study design and context details in the eligible papers. According to a previous systematic review on CS outreach programs from 2009 to 2015 [**7**], the authors made several recommendations on reporting outreach details for future CS outreach studies. These recommendations continue to remain unclear or unreported for eligible studies since 2016.

From a research standpoint, more readily available descriptive statistics would provide the ability for researchers to calculate the effect size, which could then more tangibly and quantitatively measure impact on specific programming knowledge and student outcomes through a meta-analysis. From an organizer level, this information would be particularly helpful in managing resources if programming camp research of middle school students can explicitly map student outcomes to more specific programming camp logistical decisions such as length of camp or parent involvement.

4 Limitations

First, there is currently no programming camp study that has stated to be a randomized controlled trial study. Secondly, both paper selection and coding papers were done by a single reviewer, and there is likely some bias based reviewer's prior experience and expertise level in the field. Thirdly, there were only 14 eligible papers which met most inclusion criteria with 16 studies, which can also affect the generalizability of the systematic review results. Fourth,

several studies which met nearly all eligible criteria were excluded due to the participation of mixed middle school and non-middle school students in the programming camp. Lastly, risk of bias, publication bias, and study quality of each paper was not assessed.

5 Conclusion

The findings from this study provide a general overview of how recent programming camps have assessed the effectiveness in organizing and designing curriculum to support middle school students' programming knowledge and student attitudes. The programming camps' organizing decisions from eligible studies have so far aligned with current literature and research on teaching novice programmers. Of the several programming camp organization decisions investigated in this review, the high usage of block programming is supported by research on how it can provide a less intimidating introduction to programming and, therefore, support programming knowledge. Another significant finding is the high implementation of a project-based approach towards teaching middle school students, which may support the dismantlement of negative CS perceptions as isolating and impersonal. This review also identified gaps in reporting programming camp logistical, study design, and descriptive statistics data, and the importance of reporting clear and specific details for assisting both future programming camps and research.

References

[1] C. Bryant. "A Middle-School Camp Emphasizing Data Science and Computing for Social Good". en. In: *Proceedings of the 50th ACM SIGCSE*. New York, NY, USA, Feb. 2019, pp. 358–364.

[2] R. Cabrera, M. Ángeles Carrión, and A. Carrión. "Camps IEEE Ecuador: A proposal to increase children's interest in STEM areas". en. In: *2021 IEEE XXVIII INTERCON*. Aug. 2021, pp. 1–4.

[3] L. Carter. "Why students with an apparent aptitude for computer science don't choose to major in computer science". en. In: *SIGCSE Bull* 38.1 (Mar. 2006), pp. 27–31.

[4] S. Cheryan, B.J. Drury, and M. Vichayapai. "Enduring Influence of Stereotypical Computer Science Role Models on Women's Academic Aspirations". en. In: *Psychology of Women Quarterly* 37.1 (Mar. 2013), pp. 72–79.

[5] R. Christensen et al. “Longitudinal analysis of cognitive constructs fostered by STEM activities for middle school students”. en. In: *Knowledge Management and E-Learning* 6 (June 2014), pp. 103–122.

[6] P.J. Clarke et al. “Impact of Using Tools in an Undergraduate Software Testing Course Supported by WReSTT”. en. In: *ACM Trans. Comput. Educ* 17.4 (Aug. 2017).

[7] A. Decker and M.M. McGill. “Pre-College Computing Outreach Research: Towards Improving the Practice”. en. In: *Proceedings of the 2017 ACM SIGCSE*. New York, NY, USA, 2017, pp. 153–158.

[8] A. DeWitt. “What We Say vs. What They Do: A Comparison of Middle-School Coding Camps in the CS Education Literature and Mainstream Coding Camps (Abstract Only”. en. In: *Proceedings of the 2017 ACM SIGCSE*. New York, NY, USA, Mar. 2017, p. 707.

[9] A. Emerson. “Cluster-Based Analysis of Novice Coding Misconceptions in Block-Based Programming”. en. In: *Proceedings of the 51st ACM SIGCSE*. New York, NY, USA, 2020, pp. 825–831.

[10] R. Feldhausen, J.L. Weese, and N.H. Bean. “Increasing Student Self-Efficacy in Computational Thinking via STEM Outreach Programs”. en. In: *Proceedings of the 49th ACM SIGCSE*. New York, NY, USA, 2018, pp. 302–307.

[11] D.A. Fields, Y.B. Kafai, and M.T. Giang. “Youth Computational Participation in the Wild: Understanding Experience and Equity in Participating and Programming in the Online Scratch Community”. en. In: *ACM Trans. Comput. Educ* 17.3 (Aug. 2017).

[12] S. Grover, P. Lundh, and N. Jackiw. “Non-Programming Activities for Engagement with Foundational Concepts in Introductory Programming”. en. In: *Proceedings of the 50th ACM SIGCSE*. New York, NY, USA, 2019, pp. 1136–1142.

[13] L.S. Hirsch, S. Berliner-Heyman, and J.L. Cusack. “Introducing Middle School Students to Engineering Principles and the Engineering Design Process Through an Academic Summer Program”. en. In: *INTERNATIONAL JOURNAL OF ENGINEERING EDUCATION* 33.1, B (2017), pp. 398–407.

[14] D.Lusa Krug et al. “Code Beats: A Virtual Camp for Middle Schoolers Coding Hip Hop”. en. In: *Proceedings of the 52nd ACM SIGCSE*. New York, NY, USA, 2021, pp. 397–403.

[15] A.-J. Lakanen and T. Kärkkäinen. "Identifying Pathways to Computer Science: The Long-Term Impact of Short-Term Game Programming Outreach Interventions". en. In: *ACM Trans. Comput. Educ* 19.3 (Jan. 2019). DOI: 10.1145/3283070..

[16] L. Moors, A. Luxton-Reilly, and P. Denny. "Transitioning from Block-Based to Text-Based Programming Languages". en. In: *2018 International Conference on Learning and Teaching in Computing and Engineering*. Apr, 2018, pp. 57–64.

[17] C.N. Outlay, A.J. Platt, and K. Conroy. "Getting IT Together: A Longitudinal Look at Linking Girls' Interest in IT Careers to Lessons Taught in Middle School Camps". en. In: *ACM Trans. Comput. Educ* 17.4 (Aug. 2017), pp. 1–20 17.

[18] B.R. Page et al. "Robotics Education To and Through College". en. In: *ROBOTICS IN EDUCATION: CURRENT RESEARCH AND INNOVATIONS*. Vol. 1023. 2020, pp. 101–113.

[19] B. Pérez and Á.L. Rubio. "A Project-Based Learning Approach for Enhancing Learning Skills and Motivation in Software Engineering". en. In: *Proceedings of the 51st SIGCSE*. New York, NY, USA, 2020, pp. 309–315.

[20] R. Taub, M. Armoni, and M. Ben-Ari. "CS Unplugged and Middle-School Students' Views, Attitudes, and Intentions Regarding CS". en. In: *ACM Trans. Comput. Educ* 12.2 ().

[21] C. Wang and M. Frye. "miniGEMS 2018 Summer Camp Evaluation: Empowering Middle School Girls in STEAM". en. In: *2019 IEEE Integrated STEM Education Conference*. 2019, pp. 149–155.

[22] S. Wang et al. "Introducing STEM to 7th Grade Girls using SeaPerch and Scratch". en. In: *2020 IEEE Frontiers in Education Conference (FIE*. Oct. 2020, pp. 1–8.

[23] C. Yang and D.R.R. Smith. "Research Support-Oriented MATLAB Learning: Tackling Difficult Concepts and Promoting Personalised Learning". en. In: *New Directions in the Teaching of Physical Sciences* 12.1 (2017), pp. 1–6.

[24] D. Yang, Y. Baek, and S. Swanson. "Developing Computational Thinking through Project-Based Airplane Design Activities". en. In: *2020 IEEE Frontiers in Education Conference*. Oct. 2020, pp. 1–4.

[25] N. Zamin et al. "Learning Block Programming using Scratch among School Children in Malaysia and Australia: An Exploratory Study". en. In: *2018 4th International Conference on Computer and Information Sciences (ICCOINS*. Aug. 2018, pp. 1–6.

A Course Model for Enhancing Applied Non-Technical Skills of Computer Science Students*

Ben Tribelhorn[1] and Radana Dvorak[2]
[1]The Donald P. Shiley School of Engineering
University of Portland
Portland, OR
tribelhb@up.edu
[2]The Hal and Inge Marcus School of Engineering
St. Martin's University
Lacey, WA
rdvorak@stmartin.edu

Abstract

This paper presents a course model for changing the mindset of students as future leaders in technology-based fields. Research shows that entrepreneurial education within engineering is an important facet of degree programs; however there are not as many models incorporating this mindset focus into computer science education. The authors describe a "Business and Technology Ventures" course that is required for computer science majors. The topics, outline, and projects are described with potential adaptions so that others can adopt or adapt components of this course model. One of the specific benefits of this approach is that it increases job readiness by enhancing non-technical skills. This course has been offered three times, and the instructors are satisfied with students' achievement of the learning objectives and students' achievements.

1 Introduction

According to a survey by the Association of American Colleges and Universities (AAC&U), employers highly value skills outside of technical competency, including teamwork, creative thinking, and communication.[3] Additionally, employers rate self-confidence third (after work ethic and initiative) for mindsets. Of these, self-confidence is probably the easiest to improve. Changing demographics of Computer Science (CS) students means that integrating job skills of communication, teamwork, professionalism, and leadership are increasingly crucial for successful teams.

Various models of entrepreneurial education within CS have been investigated in recent years. Often these focus on mindset shifts and appear in Software Engineering or Capstone courses.[8] In the same report by the AAC&U, employers place great value on internships, so students need to be better prepared to get these positions prior to a senior-level experience. This model aims to help improve students' preparedness for internships and careers. There is an opportunity to further focus on developing the mindset skill of self-confidence and essential non-technical skills as a focal point of a course for CS students.

2 Course Model

2.1 Background

Within engineering programs, Engineering Economics might be a comparable course to the model proposed. However, these courses tend to focus on learning objectives mostly relevant to professional certification within engineering disciplines. Additionally, these courses look at project comparisons but not particularly concerned with ideation or implementation. Entrepreneurship-focused courses are a better comparison to address the skills highlighted. To date, there is no single model for embedding entrepreneurial education within CS, so the authors propose this model given the considerations at our local institution.

2.2 University Context

The department's yearly program review identified the need for creating this course. We identified multiple issues from lowering rates of career outcomes at graduation, senior exit survey data expressing a feeling of lack of direct preparedness for job searching, and dissatisfaction with a requirement of the major to take an Innovation course in the Business department. University of Portland CS students are required a course focused on ideation and innovation,

which was not perceived as practical enough from a software engineering perspective. We decided to adjust the course to meet the needs of the CS program. Making the course an elective gave students the option to take the required course from the Business School focused on ideation, or our new course, which focused more holistically on career readiness.

Secondly, this new focused course filled a gap in our advising work where we focus on professional development each semester. This course is designed to be taken in the Spring of the Junior year after taking the CS Seminar course in the Fall semester which delves into ethical issues within technology.

2.3 Development

The authors both had a variety of past experiences incorporated into the development of this course, having founded businesses, liaised with industry, and worked with external foundations. One goal in developing this course was to include a comprehensive survey of useful topics and contexts for our students to be better prepared for any future industry experience.

Course materials were integrated from personal knowledge and sources including: cases from Harvard Business School Publishing (HBSP)[7], The Entrepreneur's Guide to Business[2], Exponential Organizations[4], Cracking the Coding Interview[5]. The only textbook requirement is that the students must purchase the negotiation simulations from HBSP.

2.4 Course Assessments

As our program is ABET (Accreditation Board for Engineering and Technology) accredited, we find it helpful to link the course learning objectives to Program Outcomes[1]. This course aligns strongly with the following three:

3. Communicate effectively in a variety of professional contexts.

4. Recognize professional responsibilities and make informed judgments in computing practice based on legal and ethical principles.

5. Function effectively as a member or leader of a team engaged in activities appropriate to the program's discipline.

The course is organized around the following learning objectives with notable mappings:

- Evaluate job offers and rank them based on total value (Economic equivalence)
- Communicate a business goal (ABET 3, 5)

- Convey technical knowledge (ABET 3)
- Define and discuss policy (ABET 4, 5)
- Identify and evaluate risk as it relates to law and policy (ABET 4)
- Demonstrate the ability to prepare, describe, and effectively defend a business plan. (ABET 3, 5)
- Analyze and create value in negotiation scenarios (ABET 3)
- Analyze companies and business ideas to project value and growth potential (Apply engineering economics principles)

Although this course is not a benchmark course, since it functions as an elective, the mapping to ABET outcomes helps support the inclusion of this course in our curriculum. In terms of formal assessment, individual assignments could be selected to track student performance. In the results, the authors will comment on the qualitative performance of students and their feedback.

2.5 Design

The course design consists of two focus areas: individual professional development and a team-based semester-long project. The course is taught three times a week for 55 minutes. The beginning of the course covers professional development topics in order to ensure students prepare for the career fair that occurs before spring break, which includes a graded activity. Then we introduce the group project and cover topics in the context of businesses. The course schedule is listed in Appendix A.

Individual professional development: The students participate in several activities, including resumé reviews, creating a job and skills matrix, job window shopping, writing cover letters, giving elevator talks, conducting peer technical interviews, and completing negotiation simulations. These activities and assignments are supplemented with lectures on personal finance, tactics for interviews, negotiation, etc. In our capstone course, the following Fall, we have students complete the 10-year design your life activity (from the Stanford Design School), which would be appropriate in this class if we didn't have it embedded elsewhere.

The students find the negotiation-focused work some of the most useful. In addition to simulations from HBSP cases, we also give a Multiple Equivalent Simultaneous Offer (MESO) assignment, which requires the students to prepare three job offers from the same company with different terms that are equally appealing. Students find this assignment very challenging as it is difficult to quantify various job benefits in pure dollar terms (e.g., an extra week of paid time off vs. a private office).

Team-based project: Students spend the semester working in a group of approximately four ideating and iterating on a business idea in an area of technology. The project scope is broad in that students are encouraged to create product ideas that would require additional advances to be commercialized. This allows partially unrealistic business ideas to excite and inspire students through the business planning work. Students must brainstorm business ideas and then complete a lean business plan, a one-page document with notes on each business plan component. Over the course, each additional topic is covered in class time to lecture and/or research to complete sections of a complete business plan. These topics include: product description as a problem and solution, market analysis (target and competition), revenue sources, development and operating expenses, business funding, and business operations (human resources, policy, intellectual property, export law). Ultimately, students will produce a business plan with several appendices based on these topics. With additional elevator pitches of the product to the class, they get feedback to refine their product, culminating in a final venture capital (VC) pitch to a panel of judges (faculty).

One of the most challenging topics is business funding and how the various percentages of ownership change. So after showcasing various technology firms' interesting successes in terms of founder control (Snap, Meta, Alphabet), we show a Shark Tank episode pausing after each investor makes an offer to see how each compares. These real-life examples build much deeper engagement and understanding. Other areas of difficulty include market analysis and ideation. For the market analysis, students find that Statista[1], with access through the school library, can be useful for finding relevant demographics and numbers. For ideation, we use a number of brainstorming techniques, some of which can be found in Thinkertoys[6].

2.6 Adoption

The authors recognize that not every institution could implement an entire course. We suggest that components of this course could be integrated with existing courses. The authors believe that a Software Engineering course is a great place for elevator pitches, presentations, even negotiation (e.g., of software requirements). An ethics course could address issues of wages, benefits, and maternity/paternity leave. Lastly, it would be easy to conduct whiteboard coding interviews in an Analysis of Algorithms course. A Software Engineering course is also an excellent place for interview preparation, and a variant of technical interviewing could include interviewing for team lead which requires leadership and management skills.

[1] `www.statista.com` provides over a million statistics across 170 industries integrating over 22,000 sources as of July 2022.

Many institutions use their Senior Capstone to measure their ABET outcomes, which is another great place to embed some of these activities. The MESO assignment and negotiation simulations would be a great fit as these students should hopefully be about to compare job offers, and this would prepare them for that process. The authors will provide materials from the course to anyone who makes an email request.

3 Results

This course was piloted in the classroom, with the second offering completely online due to COVID-19, and the latest iteration back in the classroom. In the classroom, especially, the course is engaging, and students preferred it to the previous innovation-focused requirement. The instructors are satisfied with the achievement of the learning objectives. Students gain considerable confidence in public speaking through the elevator pitches, interviews, presentations, and negotiations. The students feel better prepared for industry because they become more familiar with terminology outside of CS, especially when it comes to business-focused language.

At the end of the semester, students have a complete product idea which could be developed. Students often learn that some creative ideas might not be actionable without a large R&D investment. Examples of previous projects include: ski-goggles that use UV/radar to see through fog and snow, mask compliance detection camera software for retail, traffic and power grid smart management software, and quantum high-frequency trading algorithms/software. With the open-ended nature of this product ideation, students find that their collaboration skills have increased. On most teams at least one student develops leadership skills they didn't expect as they take the lead on selling and pitching the product to the rest of the class.

4 Discussion

The instructors find that students leave this class with a more upbeat attitude towards their careers regarding their readiness to apply and integrate into a team. The focus on negotiation and individual oral communication skills helps develop a noticeable mindset shift in the students' self-confidence.

Combining particular knowledge and skills with a team project balances the class and helps the students engage. With the large set of topics, not all student work is of the highest quality, especially in some areas of the business plans, since they are not experts in finance, accounting, or other business areas. Despite this, the VC pitches are a real highlight and showcase effective communication. Ultimately, the course develops three of the five ABET Pro-

gram Outcomes for CS students with this course model, and its components are readily adoptable at other institutions.

Acknowledgements

Thanks to the Donald P. Shiley School of Engineering for supporting this work and course development.

References

[1] ABET. Criteria for accrediting computing programs, 2021-2022. https://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2021-2022/.

[2] Constance E. Bagley. *The Entrepreneur's Guide to Law and Strategy.* Cengage Learning, 5th edition, 2017.

[3] Ashley Finley. How college contributes to workforce success: Employer views on what matters most. *Association of American Colleges and Universities*, 2021.

[4] Salim Ismail. *Exponential Organizations.* Diversion Books, 2014.

[5] Gayle Laakmann McDowell. *Cracking the Coding Interview.* CareerCup, 6th edition, 2015.

[6] Michael Michalko. *Thinkertoys: A Handbook of Creative-Thinking Techniques.* Ten Speed Press, 2nd edition, 2010.

[7] Harvard Business School Publishing. https://hbsp.harvard.edu.

[8] Ben Tribelhorn, Heather Dillon, Andrew M Nuxoll, and Nicole C Ralston. Connecting entrepreneurial mindset to software development. In *2021 ASEE Virtual Annual Conference Content Access*, 2021.

Appendix A

See Table 1.

Table 1: Course Schedule. ICA=In class activity

M	W	F	Deliverable for next M
Professional skills	Job window shopping	Elevator pitch	Job skills matrix
Holiday	Cover letter, ICA: Resumes	Personal finance	Resume
ICA: interviews	ICA: Brainstorming	ICA: Brainstorming	Cover letter
ICA: Market analysis	Incorporating a business	Patents & IP	Team business ideas
Real estate	Taxes	Negotiation	Team elevator pitch
Benefits	Benefits, MESO	ICA: Elevator pitches	Applied for jobs proof
ICA: Negotiation 1	ICA: Project time	Contracts	
Contracts	ICA: Pitch & project time	Global laws, IP	Lean business plan
ICA: export law	Financing businesses	ICA: Financing	
ICA: Shark Tank	Human resources	Corporate policy	
ICA: Negotiation 2	ICA: Policy	ICA: Data & ethics	Business plan draft
ICA: Project time	ICA: Project time	Guest lecture: Security policy	Watch & reflect CEO testimony
ICA: VC Pitch	ICA: Project time	Holiday	Corporate security policy
VC Presentations	VC Presentations	VC Presentations	Final draft all materials

Tutorial on Automating Configuring Parallel Compute Environments*

Conference Tutorial

Bryan Dixon
Computer Science Department
California State University - Chico
Chico CA, 95929
bcdixon@csuchico.edu

We presented work at a recent CCSC on how we automated the creation of small compute clusters for students [5]. This work discussed how we provided a guide to students to purchase two Jetson Nano boards and a suite of Ansible playbooks for the students to automate setting their boards up as a compute cluster [4][1]. We mentioned in this work an unintended consequence: students were using the playbooks to create multi-node clusters on local and cloud VMs instead of buying Jetson Nano boards.

These playbooks can work on more than just Jetson Nanos, which led to our idea for this tutorial. Since we have these ansible playbooks that could work to set up two or more nodes as a compute cluster on pretty much any Debian-based nodes. We thought it would be worthwhile walking interested faculty through step by step how to get a two-node cloud-hosted compute cluster setup using these playbooks.

This is a hands on tutorial will have the following:

1. Introduce the basic concept, end goal, and applications
2. Get everyone Google Cloud coupons and help them apply them
3. Walk through setting up two GCP virtual machine instances to be used for our example
4. Get tutorial repository onto a GCP VM instance.

5. Get dependencies for Ansible setup
6. Configure Ansible machine inventory for the VM instances created
7. Run playbooks
8. Use provided test code to validate our cluster works

Computers and the internet will be necessary to engage in hands-on activities. Google Cloud's EDU grant folks will be helping to facilitate coupons for attendees to use so that you can do the tutorial at no direct costs[3]. Google for Education is a platinum national partner for CCSC [2].

Biography

Dr. Bryan Dixon is an Associate Professor of Computer Science and has served students of diverse backgrounds for over ten years through various courses, from sophomore to graduate level. He spent a large part of his recent professional growth and teaching building systems and tools to help other faculty teach complex systems or DevOps concepts. He helped found the ACM-W student chapter at California State University Chico in 2020 and has continued to promote computing education and career paths for young women.

References

[1] Buy the latest jetson products. `https://developer.nvidia.com/buy-jetson/`, Oct 2020.

[2] Google cloud ccsc national partner. `https://www.ccsc.org/partners/google/`, Apr 2022.

[3] Google cloud higher education programs. `https://cloud.google.com/edu`, Apr 2022.

[4] Red Hat Ansible. `https://www.ansible.com`.

[5] Bryan Dixon. Automating configuring parallel compute environments for students. *Journal of Computing Sciences in Colleges*, 37(4):25–29, 2021.

An Introduction to MPI Parallel Programming with MPJ Express Library*

Conference Tutorial

Xuguang Chen
Department of Computer Science
Saint Martin's University
Lacey, WA 98503
xchen@stmartin.edu

Introduction

Parallel computing has been applied in many areas, for example databases, data mining, real time simulation of systems, financial risk management, climate modeling, and advanced graphics. As a result, it is becoming more and more popular in recent years. At present, parallel programming models can primarily be classified into two categories, which are message passing model and shared memory model. If the message-passing model is applied, it means that each task is allocated its private memories, and correspondingly different tasks can communicate each other via message exchange. Message Passing Interface (MPI) is a specification primarily focusing on the message-passing model. It was designed by the researchers from academia and industry. MPI tells the syntax and semantics of a core of library routines that are used for message-passing programs and usually are implemented in C or FORTRAN.

Java is an object-oriented and general-purpose programming language that can be used in various areas, including parallel computing. The compiled Java code will be able to run on each platform supporting Java. MPJ Express is an open source Java message passing library, which helps application developers write and execute parallel applications for multi-core processors and compute clusters or clouds. This tutorial introduced the basic skills conducting MPI parallel programming in Java, using MPJ Express library as an example.

At first, the tutorial shows where to acquire MPJ Express Software and how to install the software on Windows machines and Linux machines. After that, it explains how to edit, compile, and run an MPI program in Java on a Windows machine and a Linux machine. Then, the applications of several basic MPJ Express routines are covered, followed by the examples. These routine are vital for point-to-point parallel programming and collective communications, such as send/receive, broadcast, reduce, scatter, and gather operations. Following this, the tutorial will introduce how to apply MPJ Express library when writing a Java application communicating with a MySQL database. Finally, the sample code used in the tutorial will be provided and other materials suitable for self-study will be described.

Description

The tutorial is focusing on the audience who is a beginner to parallel programming and/or is interested in MPI programming, having basic knowledge of the programming languages, such as Python, Java, C#, Fortran, and/or C++. The expected learning outcomes are the followings. After attending the tutorial, the audience should know where to get a copy of MPJ Express and how to install the software on a windows machine or Linux machine. In addition, the audience should learn the concepts of MPI, point-to-point communications, and collective communications. Other than that, how to edit, compile, and run an MPI programs in Java will be learned. Moreover, the audience should learn how to implement various basic MPI operations in MPJ Express, especially basic point-to-point communications and collective communications. At the end of the tutorial, the audience can be provided the e-version of the lecture notes, code as examples, and other materials for self-study, if needed.

Reflective Curriculum Review for Liberal Arts Computing Programs*

Conference Tutorial

Jakob Barnard[1], Grant Braught[2], Janet Davis[3],
Amanda Holland-Minkley[4], David Reed[5], Karl Schmitt[6],
Andrea Tartaro[7], James Teresco[8]
[1]University of Jamestown, Jamestown, ND 58405
Jakob.Barnard@uj.edu
[2]Dickinson College, Carlisle, PA 17013
braught@dickinson.edu
[3]Whitman College, Walla Walla, WA 99362
davisj@whitman.edu
[4]Washington & Jefferson College, Washington, PA 15317
ahollandminkley@washjeff.edu
[5]Creighton University, Omaha, NE 68178
DaveReed@creighton.edu
[6]Trinity Christian College, Palos Heights, IL 60463
Karl.Schmitt@trnty.edu
[7]Furman University, Greenville, SC 29690
andrea.tartaro@furman.edu
[8]Siena College, Loudonville, NY 12211
jteresco@siena.edu

The ACM/IEEE-CS/AAAI curricula task force is currently developing an updated set of Computer Science Curricula guidelines, referred to as CS202X (since the release date is not yet determined). Information about the task force and preliminary drafts of the Knowledge Areas that will be included in the guidelines can be found online at `http://csed.acm.org`. To assist institutions in applying the new guidelines, CS202X will also publish a Cur-

ricular Practices Volume. This volume will include an article by the SIGCSE Committee on Computing Education in Liberal Arts Colleges (SIGCSE-LAC Committee) that will focus on designing or revising CS curricula in liberal arts contexts. Liberal arts colleges, and smaller colleges in general, face unique challenges when designing curricula. Small faculty sizes, limits on the number of courses that can be required for a major and the need for flexibility in student programs of study constrain designs. However, these environments also provide the opportunity to craft distinctive curricula fitted to institutional mission, departmental strengths, locale, student populations and unique academic experiences. These challenges and opportunities, combined with the size of prior curricular recommendations, have often forced smaller programs to assess trade-offs between achieving full coverage of curricular recommendations and their other priorities.

The SIGCSE-LAC Committee has heard from many faculty that their institutional and departmental contexts have indeed complicated the adoption of prior curricular guidelines. While the CS2013 and upcoming CS202X recommendations provide some flexibility for curriculum designers by dividing content into core and supplemental categories, smaller colleges still face challenges selecting content and packaging it into coherent curricula. To assist in this process, the committee is developing guidance for effectively integrating CS202X as a part of the design, evaluation and revision of computer science and related programs in the liberal arts. This guidance will encourage faculty to reflect on their programs and the role of CS202X, beginning with their institutional and departmental priorities, opportunities and constraints. Ultimately, this guidance will be presented in the committee's article in the CS202X Curricular Practices volume.

This session will open with an overview and brief discussion of the current CS202X draft. Participants will then begin working through a preliminary version of the committees' reflective assessment process. This process is framed by a series of scaffolding questions that begin from institutional and departmental missions, identities, contexts, priorities, initiatives, opportunities, and constraints. From there, participants will be led to identify design principles for guiding their curricular choices including the CS202X recommendations. Participants will leave the session with a better understanding of how CS202X can impact their programs and a jumpstart on the reflective assessment process. Feedback on the process and this session are welcome and will be used to refine the committee's guidance prior to its publication in the CS202X Curricular Practices volume.

Presenter Biography

Two of the eight co-authors of this session plan to serve as presenters. **Janet Davis** is Microsoft Chair and Associate Professor of Computer Science at Whitman College, where she serves as the department's founding chair. She co-organized SIGCSE pre-symposium events in 2020 and 2021 on behalf of the SIGCSE-LAC Committee. **David Reed** is a Professor of Computer Science and Chair of the Department of Computer Science, Design & Journalism at Creighton University. He has published widely in CS education, including the text *A Balanced Introduction to Computer Science*, and served on the CS2013 Computer Science Curricula Task Force.

Other Author Biographies

Jakob Barnard is Chair and Assistant Professor of Computer Science & Technology at the University of Jamestown. He is a member of the SIGCSE-LAC Committee and his research involves how curricula has been integrated into Liberal Arts Technology programs. **Grant Braught** is a Professor of Computer Science at Dickinson College. He is a facilitating member of the SIGCSE-LAC Committee, has organized committee events focused on curricula and has published widely on issues related to CS education, particularly within the liberal arts. **Amanda Holland-Minkley** is Chair and Professor of Computing and Information Studies at Washington & Jefferson College. Her research explores novel applications of problem-based pedagogies to CS education at the course and curricular level. She is a facilitating member of the SIGCSE-LAC Committee. **Karl Schmitt** is Chair and Assistant Professor of Computing and Data Analytics at Trinity Christian College. He has served on the ACM Data Science Task Force and various Computing, Technology, Mathematics Education related committees for the MAA and ASA. His interests explore data science education, and interdisciplinary education between computing, mathematics, data, and other fields. **Andrea Tartaro** is an Associate Professor of Computer Science at Furman University. Her computer science education research focuses on the intersections and reciprocal contributions of computer science and the liberal arts, with a focus on broadening participation. She is a member of the SIGCSE-LAC Committee, and has published and presented in venues including the CCSC and the SIGCSE Technical Symposium. **Jim Teresco** is a Professor of Computer Science at Siena College. He has been involved in CCSC Northeastern for almost 20 years and currently serves as regional board chair, and has been involved with the SIGCSE-LAC Committee for 3 years. His research involves map-based algorithm visualization.

Bloom's for Computing: Crafting Learning Outcomes with Enhanced Verb Lists for Computing Competencies*

Conference Tutorial

Cara Tang[1], Markus Geissler[2], Christian Servin[3]
[1]Computer Information Systems
Portland Community College
Portland, OR 97219
cara.tang@pcc.edu
[2]Computer Information Science
Cosumnes River College
Sacramento, CA 95823
geisslm@crc.losrios.edu
[3]Computer Science
El Paso Community College
El Paso, TX 79915
cservin1@epcc.edu

In this tutorial, participants will be introduced to Bloom's for Computing: Enhancing Bloom's Revised Taxonomy with Verbs for Computing Disciplines, a project of the ACM CCECC (Committee for Computing Education in Community Colleges). Due for final publication by the end of 2022, the Bloom's for Computing report offers a total 57 enhanced verbs across all six levels of Bloom's cognitive domain – Remembering, Understanding, Applying, Analyzing, Evaluating, Creating. The enhanced verb list is intended to support crafting more appropriate and less awkward learning outcomes and competencies that express the knowledge, skills, and dispositions required in computing disciplines. The Bloom's for Computing verb list and report is not just for use in future ACM curriculum guideline reports, but is primarily for educa-

tors in computing disciplines who find themselves needing to craft learning outcomes or competencies – whether for programs, courses, or individual modules; whether two-year, four-year, graduate, or K-12 level; whether faculty, instructional designers, or program coordinators.

The presentation and activities in the proposed tutorial session are outlined below:

1. Introductions – tutorial facilitators and participants
2. Refresher on Bloom's Revised Taxonomy, its six cognitive levels, and common verbs lists
3. Interactive discussion on how faculty approach writing learning outcomes and some of the challenges encountered
4. *Bloom's for Computing: Enhancing Bloom's Revised Taxonomy with Verbs for Computing Disciplines*
   (a) Introduce the project and the verbs
   (b) Examples of learning outcomes using the Bloom's for Computing verbs
   (c) Areas where the Bloom's for Computing verbs come in particularly handy
5. Activity where participants write or modify learning outcomes for courses they teach
6. Share out learning outcomes and thoughts on how the enhanced verbs might be used
7. Wrap up

Participants will be given a handout to take home with the complete list of verbs for each cognitive level.

This tutorial is relevant for anyone involved in writing, revising, or updating learning outcomes for programs, courses, or instructional units in computing disciplines such as Computer Science, Information Technology, and Cybersecurity

Teaching Web Development Using ASP .Net Core MVC*

Conference Tutorial

Razvan A. Mezei
The Hal and Inge Marcus School of Engineering
Saint Martin's University
Lacey, WA 98503
rmezei@stmartin.edu

In this tutorial we will demonstrate the use ASP .Net Core MVC to give students an introduction to web development. In it, students will not only be exposed to both client-side languages and tools (such as HTML, CSS, JavaScript, and Bootstrap) but also to server-side ones (C#, Razor Engine) and an object relational mapper (Entity Framework Core).

There are several important reasons why one should teach such a course to their students. Firstly, it allows students to develop a medium/large-sized practical project (a web development) that combines several languages (both client and server side). It also provides a great playing ground for applying most of the Object-Oriented concepts (including inheritance, interfaces, dynamic and static polymorphism, and many other), and expose students to many other important concepts (such as responsive design, authentication, object relational mapper, cookies, session information, HTTP verbs, CRUD operations, unit testing, asynchronous programming, cross-platform development). Lastly, such a course would also provide practical experience with technologies used by some large regional employers (several Washington State job posts currently advertised include this technology [1]).

In this tutorial, we will go over the concepts enumerated above and see how they are used throughout the course. We will then present some professional resources that are available online for free (see [2] and [4]). One of these resources ([2]) can be used as a lab component because it gives learners an opportunity to see how various concepts can be applied to an existing project.

The other resource ([4]) can be used as a resource to help students better understand the concepts covered in class (or go deeper) and complete their own personal project (a Web application using ASP Net Core MVC) based on their own interests. Individualized assignments can help keep students motivated and engaged throughout the course as well as "deter cheating or blindly copying from other students" ([3])

References

[1] Careers.wa.gov. Find a job working for washington state. (n.d.). https://www.careers.wa.gov/.

[2] Microsoft Docs. Get started with asp.net core mvc. https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-6.0&tabs=visual-studio.

[3] M. M. Girgis and L. Petry. An effective quiz strategy for enhancing student engagement while discouraging academic dishonesty. http://people.se.cmich.edu/yelam1k/asee/proceedings/2018/1/52.pdf.

[4] GitHub. Microsoft learning. https://github.com/MicrosoftLearning.

From Torches to Transistor: Using Minecraft to Teaching Processor Architecture*

Conference Tutorial

Jeffrey A Caley and Kieran Kim-Murphy
Department of Computer Science
Pacific Lutheran University
Parkland, WA 98447
caleyjb@plu.edu,kimmurkj@gmail.com

Abstract

This tutorial will demonstrate a series of labs which utilize Minecraft to provide students with a hands-on learning experience about processor architecture. We will discuss the basics of minecraft, a series of labs that have been developed to take students from building basic logic gates all the way to programming an 8-bit computer. We will share lessons learned during that time and suggest variations to the content to expand or shrink the complexity for the students.

Introduction

Providing hands-on learning exercises when teaching processor architecture can be a challenge. Because electricity is invisible, students struggle to 'see' what is going on with physical demonstrations. Simulation software helps students see what's going on, but dimensions the 'hands-on' learning that benefits students. This tutorial will demonstrate a third way by utilizing Minecraft, a sandbox video game, to allow students to build processors from the ground up and experience how they operate in a 3D environment.

This tutorial will cover everything necessary to run these labs. We will cover:

1. The basics of minecraft
2. How to setup a minecraft server
3. 7 labs taking students from basic building blocks to programming an 8-bit computer
4. Possible variations in content
5. Lessons learned

Method

Minecraft is a voxel-based sandbox game where players build things 1 block at a time. By utilizing just a few of the 100's of different block types, combined in particular ways, we can build logic gates, generate 1's and 0's and wire together various built components. Given these basic building blocks, we can combine them to build fully programmable computers.

The series of labs presented in this tutorial covers about 8 weeks worth of lab work. The labs consist of the following:

1. Learning Minecraft and logic gates: This lab is designed to familiarize students with minecraft, the 5 main blocks used in computer construction (redstone, repeaters, torches, switches, pistons).
2. Building an ALU: This lab is a 2 weeks lab where students are told to build an ALU with a set of 10 instructions. It's up to the students to figure out what that looks like. Every ALU turns out unique
3. Building RAM and Register File: Students learn how to build RAM and a register file. Learn about read/write, input/output of data
4. Addressing with MUX: To make the RAM addressable, the memory locations are MUXed. Students learn about MUX's and build one to make RAM addressable
5. Encoder/Decoder or program memory: The Encoder/Decoder allows the students to pick how they encode their instructions in binary. This allows the students to store and execute their programs in RAM. Program memory is a simplified implementation where students 'program' their computer by enabling various gates
6. Wiring everything together: Connecting all the components together and verifying data correctly flows through the machine
7. Debugging and programming: Work on getting your computer working and program a simple program

Conducting Departmental Reviews and Serving as a Reviewer*

Conference Tutorial

Henry M. Walker
Professor Emeritus, Grinnell College
Lecturer, Sonoma State, Napa, CA 94559
walker@cs.grinnell.edu

Many colleges conduct periodic reviews of departments, separate from ABET accreditation. Typically, the department writes a self study and chooses external reviewers, reviewers visit campus and write a report, and the department responds. In this process, faculty sometimes have questions about how to develop a self study, whom to include on a schedule for the external reviewers, and how to construct a reviewers' report. This 90-minute Tutorial provides guidance for performing each component of the process.

Audience: This workshop has two overlapping audiences:

1. Faculty in programs/departments that are anticipating external reviews within the next few years, and
2. Faculty who may serve as external reviewers.

Approximate Agenda and Schedule:

§ Planning a Departmental Review (60 minutes)

1. Making the decision to do a review (10 minutes)
2. Gathering data (15 minutes)
3. Developing a departmental self-study document (20 minutes)
4. Selecting external reviewers (5 minutes)
5. Developing an on-campus schedule for reviewers (10 minutes)

§ Serving as an External Reviewer (30 minutes)

6. Tips for reviewers (15 minutes)
7. Writing a reviewers' report (15 minutes)

Tutorial Format: Following the Approximate Agenda, the Tutorial will contain about 7 segments, each providing background, highlighting approaches, and offering suggestions. For each segment, discussion will encourage Tutorial participants to ask questions and brainstorm options for their local campuses.

Acknowledgements

This Tutorial repeats and possibly expands successful workshops held at CCSC Central Plains 2012 and CCSC Midwest 2014 [3, 6] and at MAA MathFest 2018, Minicourse # 4 [5]. The session also draws upon resources found in [2] and expands discussion found in [4].

Biography

Henry M. Walker has served as an external reviewer of 46 departments of computer science and/or mathematics since 1991, Most recently, the CS Program at Linfield Univ. He is a member of the Committee on Program Review of the Mathematical Association of America (MAA) [1], a past contributor to the MAA Committee on Consultants, and a writer of a LaTeX template for reports of external reviewers for the MAA. He also has actively participated in three reviews of his own department, in the 1970s, 1995, and 2007.

References

[1] MAA Committee on Departmental Review. *Program Review*, 2017 (accessed July 3, 2022). `http://www.maa.org/programs/faculty-and-departments/curriculum-department-guidelines-recommendations/program-review`.

[2] MAA Committee on Program Review. *Resources for External Consultants: Review Templates*, 2008 (accessed July 3, 2022). `https://www.maa.org/programs-and-communities/member-communities/committee-on-departmental-review/resources-external-consultants`.

[3] Henry Walker. *Conducting Department/Program Reviews and Serving as a Reviewer*, March 2012. `http://www.cs.grinnell.edu/~walker/talks/dept-reviews-cp-2012/`.

[4] Henry Walker. Getting started with a program review. *ACM Inroads*, 9(2):26–28, June 2018.

[5] Henry Walker and Rick Gillman. *Leading a Successful Program Reviews*, August 2018. `https://walker.cs.grinnell.edu/talks/maa-minicourse-2018/`.

[6] Henry Walker and Joan Krone. *Conducting Department/Program Reviews and Serving as a Reviewer*, September 2014. `http://www.cs.grinnell.edu/~walker/talks/dept-reviews-mw-2014/`.

Professionals' Perspectives on Liberal Arts and Computing Skills*

Panel Discussion

Shereen Khoja[1], Tammy VanDeGrift[2]
[1]Mathematics and Computer Science Department
Pacific University, Forest Grove, OR 97116
shereen@pacificu.edu
[2]Computer Science, Shiley School of Engineering
University of Portland, Portland, OR 97203
vandegri@up.edu

1 Summary

Faculty members from two institutions will lead a panel discussion of alumni who will share experiences about skills and dispositions that have helped them in their computing professions. The two facilitators have backgrounds in the liberal arts and computing education: Dr. Khoja is the Director of the Core Curriculum and Professor of Computer Science at Pacific University and Dr. VanDeGrift is a Professor of Computer Science and serves on the Core Curriculum Committee at the University of Portland. The panel session will begin with an overview of the liberal arts and computing curricula at the two institutions. The panel session will include questions, such as: Introduce yourselves including where you attended university, where you work, and your current role in the company. Describe the top three skills you acquired through a general, liberal arts education and how those skills are incorporated into your career. Describe the top three computing skills you acquired as an undergraduate and how those skills are incorporated into your career. What skills and experiences helped you get your first job? What courses, co-curricular activities, or international experiences helped you achieve long-term success in your career? What advice would you give first-year students who are looking ahead to their

university curriculum courses? What advice would you give faculty to prepare students for the profession?

2 Panelist Biographies

The panel session features the following software professionals:

Alexa Accuardi graduated in 2019 from University of Portland with a BS in Computer Science and Mathematics. She is an Adjunct Instructor at University of Portland and a Software Engineer at Digits Financial, Inc. At work, Alexa focuses on frontend web development in React and TypeScript. She is also pursuing her MS in Computer Science from Georgia Tech. Alexa attributes her long-term career success to her participation in the MECOP internship program and her experience as a Teaching Assistant in college. She is an active member of the Society of Women Engineers Columbia River Section and is passionate about mentoring and educating future software engineers.

MacKevin Harrison Fey Jr graduated in 2015 from University of Portland with a BS in Computer Science and a BS in Electrical Engineering. He is a Senior Software Engineer at Microsoft and works on experimentation and personalization services for website infrastructure. His responsibilities include feature costing, scoping, design, support, and mentoring junior colleagues. MacKevin attributes several things to his successful preparation: the CS curriculum - holistic approach to problem-solving and hands-on experience; internships and on-campus jobs (tutor, grader, lab assistant; and getting to meet and work with a variety of people.

Larry Jensen graduated in 2019 from Pacific University with a BS in Computer Science. He is a Software Development Engineer at AWS Elemental in Portland, Oregon. At work, he develops a service that enables streaming video providers to create linear streams at broadcast quality from video on demand and live content.

Nicole Peldyak graduated in 2016 from Pacific University with a BS in Computer Science and minors in Mathematics and Music. She is a Senior Software Engineer at Garmin in Salem, Oregon. At work she leads a team of software engineers responsible for designing, implementing, and testing software for certified retrofit avionics. Nicole learned many of the leadership skills needed in her current position from opportunities provided by Pacific University, such as co-founding the Women in Computer Science club and directing the student-led women's a cappella choir. She continues to seek out that type of community at Garmin, where she is part of the leadership team of the Salem site's chapter of the Women's Business Forum.

3 For More Information

1. Fung, B. 2015. Tech companies are hiring more liberal arts majors than you think. *The Washington Post*. https://www.washingtonpost.com/news/the-switch/wp/2015/08/26/tech-companies-are-hiring-more-liberal-arts-majors-than-you-think/.
2. Kowarski, I. 2019. What Can You Do with a Computer Science Degree? *U.S.News and World Report*. https://www.usnews.com/education/best-graduate-schools/articles/2019-05-02/what-can-you-do-with-a-computer-science-degree.
3. Loten, A. 2019. America's Go Talent, Just Not Enough in IT. *The Wall Street Journal*. https://www.wsj.com/articles/americas-got-talent-just-not-enough-in-it-11571168626.
4. Walker, H.M. and Kelemen, C. 2010. Computer Science and the Liberal Arts: A Philosophical Examination. *ACM Transactions on Computing Education (TOCE)*. https://dl.acm.org/doi/10.1145/1731041.1731043
5. Zander, C., Boustedt, J., McCartney, R., Mostrom, J.E. 2009. Student transformations: Are they computer scientists yet? In *Proceedings of the 5th International Workshop on Computing Education Research (ICER)*. https://dl.acm.org/doi/10.1145/1584322.1584337

Reviewers — 2022 CCSC Northwestern Conference

America Chambers ..... University of Puget Sound, Tacoma, WA
Xuguang Chen ..... Saint Martin's University, Lacey, WA
Donald Chinn ..... University of Washington Tacoma, Tacoma, WA
Janet Davis ..... Whitman College, Walla Walla, WA
Zach Dodds ..... Harvey Mudd College, Claremont, CA
Tim Harrison ..... Eastern Oregon University, La Grande, OR
Robert R. Lewis ... Washington State University at Tri-Cities, Richland, WA
Brad Richards ..... University of Puget Sound, Tacoma, WA
Ben Tribelhorn ..... University of Portland, Portland, OR
Tammy VanDeGrift ..... University of Portland, Portland, OR
Steven R. Vegdahl ..... University of Portland, Portland OR
Richard Weiss ..... The Evergreen State College, Olympia, WA
Howard Whitston ..... University of South Alabama, Mobile, AL