The Journal of Computing Sciences in Colleges

Papers of the 27th Annual CCSC Northeastern Conference

April 14th-15th, 2023
Ithaca College
Ithaca, NY

Baochuan Lu, Editor
Southwest Baptist University

Jeremiah W. Johnson, Regional Editor
University of New Hampshire

Volume 38, Number 8 **April 2023**

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

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Bryan Dixon, Southwestern Representative (2023),

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

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

CCSC National Partners

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

Platinum Level Partner

Google Cloud
GitHub
NSF – National Science Foundation

Gold Level Partner

zyBooks
Rephactor

Associate Level Partners

Mercury Learning and Information
Mercy College

Welcome to the 2023 CCSC Northeastern Conference

Welcome to the Twenty Seventh Annual Consortium for Computing Sciences in Colleges Northeast Region Conference held at Ithaca College. We'd like to thank everyone who has been involved with this conference since its inception at the University of Hartford in April 1996.

We have two excellent plenary speakers. Reuben Fischer-Baum from the Washington Post will talk on "How journalists create data visualization," and Jon Kleinberg, Professor of Computer Science at Cornell University, will talk on "Choices and Consequences in Computing." In addition, there will be a variety of activities at the conference including paper presentations, workshops, panels, tutorials, lightning talks, a student programming contest, and faculty and student posters. A special thanks goes out to the many volunteers who have worked on our conference. This includes the conference committee, the CCSCNE board, and the conference reviewers. You will find their names listed below. We also want to thank the National Partners whose support made the conference possible: Google Cloud, GitHub, NSF, zyBooks, and Rephactor. Additional support is provided to the conference for being in cooperation with the ACM Special Interest Group on Computer Science Education (SIGCSE).

There were 23 papers submitted to this year's conference, out of which 15 were accepted with an acceptance rate of 65%. There are also 3 pre-conference workshops, 2 panels, 2 tutorials, 1 lightning talk, 6 faculty posters and 61 student posters. We hope that you enjoy the conference and find it informative and engaging. We look forward to seeing you in 2024, when the conference will be held at the College of St. Rose in Albany, New York.

Ali Erkan
Ithaca College
Lawrence D'Antonio
Ramapo College of New Jersey
CCSCNE-2023 Conference Co-Chairs

2023 CCSC Northeastern Conference Steering Committee

Ali Erkan, Conference Chair ..... Ithaca College
Lawrence D'Antonio, Conference Chair ..... Ramapo College of New Jersey
Mark Bailey, Program Chair ..... Hamilton College
Bonnie MacKellar, Papers Chair ..... St. Johns University
Yana Kortsarts, Papers Chair ..... Widener University
Joan DeBello, Lightning Talks/Panels/Tutorials/Workshops Co-chair .... St. Johns University
Susan Imberman, Lightning Talks/Panels/Tutorials/Workshops Co-chair City University of New York
Ting Liu, Lightning Talks/Panels/Tutorials/Workshops Co-chair ..... Siena College
John Barr, Undergraduate Posters Chair ..... Ithaca College
Adita Kulkarni, Undergraduate Posters Chair ..... SUNY Brockport
Sandeep Mitra, Undergraduate Posters Chair ..... SUNY Brockport
Liberty Page, Undergraduate Posters Chair ..... University of New Haven
Rick Kline, Registration Chair ..... Pace University
Mark Hoffman, Registration Chair ..... Quinnipiac University
Del Hart, Programming Contest ..... SUNY Plattsburgh
Paul Olsen, Programming Contest ..... The College of St. Rose
Adam Lee, Programming Contest ..... Ithaca College
Kevin McCullen, Vendors Chair ..... SUNY Plattsburgh

Regional Board — 2023 CCSC Northeastern Region

James Teresco, Regional Chair ..... Siena College
Jeremiah Johnson, Editor ..... University of New Hampshire at Manchester
Richar Kline, Registrar ..... Pace University
Sandeep Mitra, Secretary ..... The College at Brockport
Adrian Ionescu, Treasurer ..... Wagner College
Delbert Hart, Webmaster ..... SUNY Plattsburgh

PLCC: A Tool Set for Teaching Programming Languages Courses*

Conference Workshop

Stoney Jackson[1], Tim Fossum[2], James Heliotis[3]
[1]CS&IT, Western New England University
stoney.jackson@wne.edu
[2]Professor Emeritus, SUNY Potsdam
plcc@pithon.net
[3]Professor Emeritus, Rochester Institute of Technology
jehics@rit.edu

This is a hands-on laptop-recommended workshop that introduces participants to the PLCC compiler-compiler tool set for building Java-based language interpreters in an upper-level Programming Languages course. One approach to teaching this course is to show how programming language features such as language syntax, variable lifetime, procedure application, parameter passing, recursion, and object-orientation are implemented "under the hood". This workshop is designed for CS educators who want to explore how PLCC can be used to support such an approach.

1 How PLCC Works

The PLCC system contains a compiler-compiler (implemented in Python 3). It also contains a small collection of Java support programs. Input to the PLCC compiler-compiler is a single specification file with three sections that describe the lexical, syntactic, and semantic structure of a programming language. The PLCC compiler-compiler takes this specification and generates Java source files that implement a self-contained interpreter for the language, including a front-end scanner, a recursive descent parser, and an evaluation engine including a read-eval-print loop. The specification file defines language tokens using simple regular expressions, language syntax using straightforward BNF rules, and language semantics as Java code segments. These code segments become

part of Java classes, built by PLCC, that correspond to each of the BNF rules in the syntax specification.

2 The Workshop

Participants will be asked to work in small teams so they can learn from each other and benefit from problem-solving discussions. Each team will share a laptop computer owned by one of the team members. The laptop will be used only to log into a workshop-provided server that contains all of the resources necessary to run PLCC on the workshop examples.

Teams will work through a sequence of examples and problems that illustrate, in turn, each of the essential parts of a language specification:

- Lexical specification examples for token recognition
- Lexical and grammar specifications for illustrating syntax recognition
- Interpretation of programs written in simple programming languages

Each example will be presented by (1) describing a worked-out PLCC example specification that teams can test on the server; (2) proposing a similar problem whose solution requires modification or extension of the given example; and (3) having the participant teams implement the proposed modification on the server by modifying and testing the PLCC specification file. These examples will be used as a springboard to illustrate how to implement language features, including scope, functional programming, call semantics, object-oriented programming, type systems, and logic-based programming.

3 Workshop Preparation by Participants

A laptop is **not necessary** to participate in this workshop. However, the workshop requires a sufficient number of participants with laptops to allow for putting together teams of two or three.

Workshop participants will be given access to a PLCC environment via Wi-Fi and SSH. Participants with laptops will be asked to install an SSH client (such as PuTTY for Windows; MacOS already has one) before attending.

PLCC requires use of a plain text editor to edit language specification files. The workshop server environment will contain **vim**, **emacs**, and **nano**.

All of the PLCC workshop materials will be available on the PLCC website: `https://plcc.pithon.net`.

How Do Computer Systems Work?*

Conference Workshop

Peter B. Henderson, Emeritus
Computer Science and Software Engineering
Butler University, Indianapolis, IN 46208
phenders@butler.edu

This workshop is for computing educators who are challenged with explaining how computer systems work to a wide range of lay audiences. These might include K-12 students, life long learners, undergraduate or graduate students, and teachers[1]. For example, you have been asked by a 5^{th} grade teacher to explain to his/her class how computers work. Your task is to prepare an interactive, engaging, fun, informative one hour presentation - perhaps like CS Unplugged activities[1].

If you begin with an online search, maybe "how do computers work kids", you will discover a broad spectrum of results. However, few explain how computers actually work - e,g., interactions of components, how instructions are run, how data is shared and processed, etc. This workshop will begin by presenting, and discussing, various educational techniques used to explain how computers work for various learners. As preparation, prior to the workshop, participants will do some preliminary research, and formulate their own ideas.

"How Computers Work is very complex," a post on a computing educational blog, captures the challenge precisely. Even for specific groups of learners, e.g., 4^{th} and 5^{th} grade students, computing educators have found this difficult. A holy grail would be a general, scalable, and visually interactive model, with kinesthetic activities, which is suitable for diverse audiences who are curious about how computers systems work[2]. In this context, scalable means both *adaptable* for different groups of learners, and *extendable*, that is, more advanced concepts, building on those already understood, can be introduced. For example, adding an simple integer Arithmetic Unit to a CPU model after a basic CPU model for simple integer I/O is understood.

---

[1]Many K-12 CS teachers don't understand how computers work

[2]One inspirations is Jerome Bruner's assertion[3] "Any subject can be taught to any one at any age in an intellectually honest fashion if their level of development is heeded."

A proposed general model will be introduced, and opened for discussion. This model has been used in presentations for Osher Life Long Learning courses at William and Mary, and for middle school technical courses. Both were virtual Zoom presentations due to Covid restrictions. Several in-person presentations for local K-12 classes are being scheduled prior to this workshop. Here, student activities will include role playing (e.g., input, processing, and output), where it is very important that the roles closely match those of the corresponding components in actual computer systems.

Details of the model will not be given here, however, if you are interested in learning more, including ideas for role playing, you are encouraged to watch a 20 minute outtake from the middle school Zoom presentation[2].

Bibliography: Dr. Peter B. Henderson graduated with an Engineering PhD in 1975 from Princeton University. His academic career started in the Computer Science Department at The State University of New York at Stony Brook in 1975. In 2000 he accepted an offer from Butler University to establish a new department, Computer Science and Software Engineering. He retired from Butler in 2007.

Dr. Henderson has been active in CS and SE education for many years, with a focus mainly on promoting the importance of mathematics in CS and SE education. He has participated in several CS and SE curriculum efforts, was editor of a SIGCSE Inroads column "Math Counts," and a SIGSOFT Notes column "Software Engineering Education" with Mark Ardis.

References

[1] Computer science unplugged. `https://www.csunplugged.org/en/`. Accessed on December 15, 2022.

[2] How do computer systems work? video clip. `https://blue.butler.edu/~phenders/SIGCSE_Special_Projects_Grant/`. Accessed on December 15, 2022.

[3] Jerome Bruner. *The Process of Education, Second Edition.* Harvard University Press, 2009.

Enhancing Computing Accessibility Education Using Experiential Labs: A Focus on Screen Readers and Dexterity Impairments*

Conference Workshop

Su Thit Thazin, Kyle Messerle, Heather Moses,
Domenic Mangano, Samuel Malachowsky, Daniel Krutz
Department of Software Engineering
Rochester Institute of Technology
Rochester, NY 14623
{st5626, klm3580, hlm8500, dm9965, samuse, dxkuse}@rit.edu

The *Accessible Learning Labs* project informs participants on how to properly address web accessibility guidelines, while also demonstrating the need to prioritize accessibility in software development. In this session, an overview of the labs will be provided, along with usage instructions and information for adopters. This tutorial will be beneficial for a wide-range of participants in the software engineering field who would like to gain insight into building inclusive and accessible software. All project material is available on our website: `https://all.rit.edu`

1 Introduction

To address the lack of accessibility education in computing, we have created a series of systematically developed educational modules collectively referred to as the *Accessible Learning Labs (ALL)*. These labs provide a foundational understanding of computing accessibility concepts and raise awareness among participants regarding the importance of creating accessible software.

No software download or installation is necessary to use the labs; only a web browser is required. Due to the labs' self-contained, web-based nature and the inclusion of all instructional materials, they enable easy integration into a

wide variety of curricula ranging from high schools, undergraduate introductory computer science classes to graduate courses.

This workshop will allow participants to: I) Learn about our educational material offerings, II) Provide feedback and insight on methods to enhance the lab experiences, and III) Share thoughts on common challenges and best practices for incorporating the topic of accessibility in computing education. No prior experience in accessibility-related topics will be required.

1.1 Lab Structure

Each lab aims to address one computing accessibility issue and consists of the following components: I) Relevant background information on the topic being addressed, II) An example application showcasing the accessibility issue, III) An activity portion with details on how to fix the issue from a technical standpoint, IV) Testimonials from people on their real-life experiences with using non-accessible software, and V) A quiz where the participant is tested on their acquired knowledge on the topic.

2 Tutorial Session Agenda

Activity 1: Introductions and initial discussion: (30 minutes) We will give an overview of the labs, their objectives, and the components. This will be followed by an initial discussion on the topic of computing accessibility.

Activity 2: Screen reader-focused lab: (60 minutes) This lab informs participants on the importance of screen readers for users with visual impairments, as well as properties of quality user interfaces that are optimized for screen reader usage. Included in the lab is an activity where the user experiences a series of web pages that are not screen reader friendly. The user then learns how to address the issue on each page from a technical standpoint.

Activity 3: Dexterity-focused lab: (60 minutes) This lab introduces students to the concept of making software accessible to users with dexterity impairments. The exercise tasks involve the usage of a mouse and a keyboard which aim to mimick difficulties commonly faced by a user with dexterity impairments. The participant is then shown how to repair the cumbersome quality of the pages according to certain WGAC guidelines.

Activity 4: Lab feedback: (30 minutes) Participants will share their thoughts on the presented material and suggest ways to improve the labs. This feedback will be used to shape the design of future labs.

Acknowledgements

This material is based upon work supported by the United States National Science Foundation under grants #1825023, #2111152 and #2145010.

Presenter Biographies

Su Thit Thazin is the Director of Outreach for the project, and a 5th-year Software Engineering undergraduate student. Su is dedicated to the dissemination of the labs, and has been working with instructors at RIT and external universities to incorporate the labs into their class curricula.

Kyle Messerle is a 2nd-year Software Engineering undergraduate at RIT who leads many of the labs' development efforts.

Heather Moses is a 4th-year undergraduate student studying Software Engineering. She has contributed to the ALL project as a software developer in the past but now acts as a project manager. Heather has also co-authored several papers for the ALL project.

Domenic Mangano is a 2nd-year undergraduate Software Engineering student at RIT who joined the ALL team in the Fall of 2022 and contributes to the project as a junior software developer.

Samuel Malachowsky is the Co-PI of the project that is developing the described accessibility labs. Malachowsky is a Sr. Lecturer at RIT and has authored 9 pedagogically focused publications. He holds a Project Management (PMP) certification and has authored a textbook on team leadership.

Daniel Krutz is the PI of the NSF-funded projects (#1825023, #2111152 and #2145010) that are devoted to creating the presented labs. Krutz has taught approximately ten different graduate and undergraduate software engineering courses and is the author of over twelve pedagogical research papers.

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*

aholl andminkley@washjeff.edu

[5] *Creighton University, Omaha, NE 68178*

DaveReed@creighton.edu

[6] *Trinity Christian College, Palos Heights, IL 60463*

Karl.Schmitt@trnty.edu

[7] *Furman University, Greenville, SC 29690*

andrea.tartaro@furman.edu

[8] *Siena College, Loudonville, NY 12211*

jteresco@siena.edu

The ACM/IEEE-CS/AAAI curricula task force is currently developing an updated set of Computer Science Curricula guidelines, referred to as CS2023. 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, CS2023 will also publish a Curricular 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 CS2023 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 CS2023 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 CS2023, beginning with their institutional and departmental priorities, opportunities and constraints. Ultimately, this guidance will be presented in the committee's article in the CS2023 Curricular Practices volume.

This session will open with an overview and brief discussion of the current CS2023 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 CS2023 recommendations. Participants will leave the session with a better understanding of how CS2023 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 CS2023 Curricular Practices volume.

Presenter Biographies

Two of the eight co-authors of this session plan to serve as presenters.

Grant Braught is a Professor of Computer Science at Dickinson College. He is a facilitating member of the SIGCSE-LAC Committee, has organized committee events focused on curricula and has published widely on issues 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.

Other Author Biographies

Jakob Barnard is Chair and Assistant Professor of Computer Science & Technology at the University of Jamestown. He is a member of the SIGCSE-LAC Committee and his research involves how curricula has been integrated into Liberal Arts Technology programs. **Janet Davis** is Microsoft Chair and 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. **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.

GitKit: Teaching Git and GitHub/GitLab Workflow in an Authentic Context*

Conference Tutorial

Grant Braught[1], Stoney Jackson[2], Karl R. Wurst[3]
[1]Computer Science
Dickinson College
Carlisle, PA 12013
braught@dickinson.edu
[2]Computer Science and Information Technology
Western New England University
Springfield, MA 01119
hjackson@wne.edu
[3]Computer Science
Worcester State University
Worcester, MA 01602
kwurst@worcester.edu

Version control and workflow skills and concepts are central to modern software development, a requirement for free and open source software (FOSS) community participation, and are in high demand [2]. Teaching tools and concepts such as these can be more engaging and successful when done in an authentic context, such as an existing, ongoing FOSS project [4, 1]. To facilitate that we have developed the GitKit, where students learn and practice git and GitHub/GitLab skills and workflows within an existing Humanitarian FOSS (HFOSS) project.

The GitKit is an instance of what we call an HFOSS Kit. An HFOSS kit is a snapshot of an HFOSS project's artifacts (codebase(s), issues, documentation, communications, etc.) taken at a particular point in time, packaged with student learning activities, an instructor guide, and possibly a containerized development environment. Using HFOSS kits in the classroom has several

advantages. While active HFOSS projects change and evolve over time, an HFOSS Kit enables a stable, reusable educational experience. Student learning activities and instructor guides are written against the stable project snapshot thus the high cost of developing and revising activities and guides can be amortized as they are reused across multiple semesters. Students are engaged in a more authentic learning experience as they develop their FOSS skills and concepts using using actual projects and community artifacts. The containerized development environment [3] facilitates immediate context-sensitive feedback as students complete assignments. The instructor or the environment can simulate community interaction.

This tutorial is intended for CS educators of all levels who wish to teach Git and GitHub workflow in an authentic context. We will demonstrate the GitKit – what it provides, and how it can be used in the classroom. We will expand upon the details of the GitKit and the reasons for using it. We will also briefly discuss our motivation for using humanitarian projects, the technical architecture of HFOSS Kits, the tools we have developed to capture snapshots of project artifacts, support for deploying HFOSS Kits, and our future plans. Attendees will be provided a link to a site that contains the GitKit as well as other existing kits, their features, suggested classes they can be used in, and contact information for the developers.

This work was supported under National Science Foundation Grants DUE-1225738, 1225688, 1225708, 2012966, 2013069, 2012979, 2012999, and 2012990. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

[1] Grant Braught, John Maccormick, James Bowring, Quinn Burke, Barbara Cutler, David Goldschmidt, Mukkai Krishnamoorthy, Wesley Turner, Steven Huss-Lederman, Bonnie Mackellar, and Allen Tucker. A multi-institutional perspective on H/FOSS projects in the computing curriculum. *ACM Trans. Comput. Educ.*, 18(2), July 2018.

[2] The Linux Foundation. The 10th annual open source jobs report: Critical skills, hiring trends, and education, September 2022.

[3] Stoney Jackson and Karl R. Wurst. Teaching with vs code devcontainers: Conference workshop. *J. Comput. Sci. Coll.*, 37(8):81–82, apr 2022.

[4] Diomidis Spinellis. Why computing students should contribute to open source software projects. *Commun. ACM*, 64(7):36–38, June 2021.

Curricular Practices for Computing for Social Good in Education*

Panel Discussion

Heidi J.C. Ellis[1], Gregory W. Hislop[2], Grant Braught[3]
[1]Western New England University
Springfield, MA
ellis@wne.edu
[2]Drexel University
Philadelphia, PA
hislopg@drexel.edu
[3]Dickenson College
Carlisle, PA

The development of the CS Curriculum Guidelines, CS2023, includes a parallel and collaborative effort to provide supplemental material that supports the guidelines. A series of peer-reviewed articles are being created by experts in various aspects of the design and delivery of Computer Science programs. The Computing for Social Good committee is developing an article that provides an international perspective on Computing for Social Good in Education (CSG-Ed). This article will include an overview of the global efforts in CSG-Ed, description of models for incorporating CSG into curricula, including examples, and recommendations and best practices for including CSG in curricula. This panel will inform the CCSCNE community about the CSG-Ed effort while also obtaining comments and suggestions about the effort from the computing education community. During the panel session, the panelists will provide an overview of the committee's effort, extracts of content from the article, and examples of research results related to demonstrating the impact of CSG-Ed. Ample time will be allotted for discussion, questions, and suggestions by the attendees.

Infusing Accessibility into Computer Science Education*

Panel Discussion

Robert Domanski[1], Stephanie Ludi[2], Richard E. Ladner[3], Peter Wu[4], Alexandra Feldhausen[1]
[1]NYC Mayor's Office of Talent and Workforce Development, Tech Talent Pipeline
{rdomanski,afeldhausen}@cityhall.nyc.gov
[2]Department of Computer Science & Engineering University of North Texas
stephanie.ludi@unt.edu
[3]Paul G. Allen School of Computer Science and Engineering University of Washington
ladner@cs.washington.edu
[4] Microsoft
petewu@microsoft.com

Summary

How do we ensure that the next generation of technologists creates computing products that are accessible to people with disabilities? As Computer Science educators, this means raising students' basic awareness and literacy about Accessibility principles and teaching more advanced software design principles so that students are empowered to design hardware and software products that can be used effectively by people who have difficulty reading a computer screen, hearing computer prompts, or controlling the keyboard, mouse, or touchscreen, and more. Accessibility is directly relevant to any course involving human-facing applications such as web and mobile development, software engineering, and human-computer interaction.

Furthermore, the landscape of Accessibility in CS education is increasingly active. Accessibility is emerging as a considered topic in accreditation requirements and ACM recommended guidelines. A growing list of opportunities can be found relating to faculty grants, faculty fellowship programs, research and development, study-away opportunities for students, employer-hosted student events, various types of academic-industry collaborations, and more.

This panel will represent numerous perspectives of Accessibility subject-matter experts both within the discipline as well as in the tech industry. Topics will include curriculum, pedagogy, WCAG standards, pointers to resources, opportunities, and more. After the panel discussion and subsequent Q&A, we expect the audience of faculty will have a new or enhanced understanding of Accessibility's importance in CS education and have information to pursue the topic further in making future contributions to this space.

Panel

The composition of the panel will include a CS Professor and Associate Chair, a CS Professor Emeritus, a representative from industry to offer an employer/practitioner perspective, and a representative from government who can speak to broader support behind Accessibility in CS education initiatives.

Structure

The panel will consist of a 75-minute program with four speakers participating in a moderated discussion. The Moderator will summarize panel talks for the audience and have two to three planned questions to start the audience discussion. Twenty minutes will be provided for audience questions and answers.

Panelists

Stephanie Ludi is a professor and Associate Chair in the Department of Computer Science & Engineering at the University of North Texas. She earned her Ph.D. in Computer Science from Arizona State University. Her research interest in accessibility involves refactoring software to maximize access by visually impaired students and programmers. In addition to her work in accessibility, Stephanie conducts research in Software Engineering education. Stephanie has led efforts to increase inclusion in K12 Computer Science and robotics curricula for students with visual impairments via curriculum redesign and tool support. In addition to curriculum support, Dr. Ludi's work involves accessible tool support in areas such as robotics and programming with Lego Mindstorms.

Other projects include refactoring block-based programming tools for accessibility and designing user interface features to support code navigation and code understanding for blind programmers.

Richard E. Ladner is a professor emeritus in the Paul G. Allen School of Computer Science and Engineering at the University of Washington. He earned his bachelor's degree in mathematics at St. Mary's College of California in 1965 and his PhD in mathematics at the University of California, Berkeley in 1971. He has been on the faculty of the University of Washington since 1971. Although he is not teaching anymore, he continues his research in accessible computing. He started his career in theoretical computer science, but has pursued accessibility research for the past 20 years. Since 2006 he has led AccessComputing, a National Science Foundation funded project with the goal of increasing the participation of students with disabilities in computing fields. In 2014, he joined with Andreas Stefik to create AccessCSforAll which works on developing accessible computer science educational technology and helps K-12 computer science teachers include students with disabilities in their classes. He is the winner of a number of awards for his advocacy work including the 2020 National Science Board Public Service Award, the 2015 Richard A. Tapia Achievement Award for Scientific Scholarship, Civic Science and Diversifying Computing, the 2008 A. Nico Habermann Award, and the 2004 Presidential Award for Excellence in Science, Mathematics and Engineering Mentoring. For his research he is the winner of the 2016 Award for Outstanding Contributions to Computing and Accessibility and the 2014 SIGCHI Social Impact Award. He was a Fulbright Scholar and a Guggenheim Fellow. He is a Fellow of the ACM, IEEE, and AAAS.

Peter Wu is a Principal Software Engineer at Microsoft. He has worked at Microsoft for 26 years on many products as both a Software Engineer and Program Manager and as both an individual contributor and manager. He has been focusing on making PowerPoint a more accessible application for the past 7 years.

Robert Domanski, Ph.D., is the Director of Higher Education for the New York City Mayor's Office of Talent and Workforce Development and its Tech Talent Pipeline industry partnership. Rob oversees the "CUNY 2x Tech" initiative – a $20 million dollar investment in the City University of New York (CUNY) to double the university's number of Computer Science graduates within 5 years and connect those graduates to jobs in the field. Additionally, Rob oversees the City's tech Academic Council, working with the Presidents and Provosts of NYC-based colleges and universities to determine how best to align Mayoral goals with the needs of the City's academic institutions. Beyond his work in government, Rob has also taught Computer Science for nearly 20 years at Kean University and the College of Staten Island. His academic re-

search focuses on Digital Government, Internet Governance, and the Politics of Algorithms. He has most recently published on the specific subtopics of Algorithmic Bias and Artificial Intelligence from technical, policy, and ethical perspectives. Recently, he also helped launch the CUNY STEM Pedagogy Institute (SPI) which provides faculty with various supports to promote inclusive- and employment-focused pedagogy in CS classrooms. In 2021, Rob received the prestigious Hayes Innovation Award in recognition of public service towards New York City's pandemic response.

Alexandra Feldhausen (moderator) is the New York City Tech Talent Pipeline's (TTP) Director of Insights and Engagement. She oversees the team's research and analysis to drive better understandings of programmatic outcomes, assess gaps across the NYC tech education ecosystem, and evaluate ongoing industry needs. In addition to this work, she leads the CUNY CS Career Development Council which supports Best Practices in tech career development across CUNY and coordinates tech industry engagement through TTP's volunteer network. Alexandra was a Fulbright Fellow and received a master's degree from the School of International and Public Affairs at Columbia University. Her background and research includes an internal audit of gender equity and mobility at the United Nations, field work on worker's rights in Brazil, and reporting on NYC labor, transportation, and homelessness at WNYC radio.

A Rotation of Data Structures*

Thomas Cook, Alexander Mentis, and Chris Okasaki†
Department of Electrical Engineering and Computer Science
United States Military Academy
West Point, NY 10996
{thomas.cook,alexander.mentis,chris.okasaki}@westpoint.edu

Abstract

This paper provides options for varying the details of certain topics from semester to semester in a data structures course that focuses—at least to some extent—on the implementation of data structures. The data structures considered are variations on balanced binary search trees and mergeable heaps (priority queues).

1 Introduction

Congratulations! You've just finished teaching Data Structures for the first time, and it went pretty well. As you begin to plan your second iteration, you have some ideas for how to teach certain topics better, but you're not sure whether to leave everything else exactly the same or to make changes even to parts of the course that seemed to go well. Here are some suggestions for some easy changes to make and why you might want to.

Curriculum design often operates at two levels of scale, the collection or sequence of topics that make up a course and the collection or sequence of courses that make up an academic major. But there is another scale less often discussed: planned variations across multiple semesters or years of the same course. We consider such variations for a Data Structures course.

Data Structures courses differ across a number of dimensions, such as programming language used, how much emphasis is placed on object-oriented pro-

†The views expressed in this article are those of the authors and do not reflect the official policy or position of the Department of the Army, DOD, or the U.S. Government.

gramming, and the balance between designing and/or implementing data structures versus using data structures provided by some language or library. Despite these differences, certain data structures are extremely commonly taught, including linked lists, hash tables, binary search trees, and heaps. We will focus on the latter two.

Both binary search trees and heaps offer a similar opportunity. After an introductory version for which there is very little significant variation (unbalanced binary search trees and the standard heap data structure[24]), it is common—at least in courses that don't focus mainly on the use of data structures from standard libraries—to progress to a more efficient successor (balanced binary search trees or heaps with an efficient merge). Pedagogically, these more efficient successors serve as useful vehicles for practicing working with non-trivial invariants as well as illustrating analysis techniques, especially those related to $O(\log n)$ running times. However, for both binary search trees and heaps, there are many possible choices for this successor that work essentially equally well, such as AVL trees[1] or red-black trees[10] (as successors for unbalanced binary search trees), or leftist heaps[7] or binomial heaps[23] (as successors for standard heaps).

We suggest varying the choice of successor with each offering of the course, moving systematically through a cycle of choices several years long. This can offer a number of advantages, including decreased opportunities for cheating and increased enrichment for the teaching staff (including teaching assistants, if any).

On the other hand, teachers assigned to a Data Structures course may not be experts in data structures and may not be familiar with a wide variety of alternatives to choose from. In this paper, we provide a curated selection of such choices and also discuss a number of "mix-ins" that can be sprinkled on each main choice.

2 Successors for Binary Search Trees

This section describes eight potential successors to unbalanced binary search trees. For each data structure, we describe the basic ideas—especially any invariants beyond the usual ordering invariants—and include citations where more complete details can be found.

For all of the data structures below, the search, insert, and delete operations run in $O(\log n)$ time, either in the worst case (for the first five) or in the expected case (for the last three).

For the first five alternatives (the ones with worst-case bounds), deletion is substantially more complicated than insertion and may not be worth the class time it would take to present. However, for the last three (the ones with

expected bounds), deletion is surprisingly easy. One tradeoff is that many students may not have enough background in probability to understand the analysis behind those expected bounds.
Here are the data structures:

- **AVL trees**[1][12, pp. 459–464]: AVL trees satisfy the invariant that, for every node, its left and right subtrees differ in height by at most one. When that difference becomes two as the result of an insertion or deletion, the invariant is restored with either a left or right single rotation or a left or right double rotation.

- **2-3 trees**[2]: 2-3 trees contain both 2-nodes (binary nodes with one key and two children), and 3-nodes (ternary nodes with two keys and three children). Because of the ternary nodes, these trees are not strictly *binary* search trees. A 2-3 tree is perfectly balanced in the sense that all paths from the root to a leaf are the same length. When an insert causes a leaf to be placed on a new level, balance is restored either by growing a 2-node into a 3-node or, when a 3-node would otherwise need to grow into a 4-node, by splitting the would-be 4-node into two 2-nodes with a 2-node parent. In the latter case, the changes continue to propagate up the tree.

- **Red-Black trees**[10][15][6, Chp. 13]: In a red-black tree, nodes are colored either red or black, with the invariants that (1) no red node has a red child, and (2) every path from the root to a leaf contains the same number of black nodes. When an insert causes a red node to have a red child, this can be fixed by a combination of rotations and re-coloring.

- **AA trees**[3, 20]: An AA tree is a variation of a red-black tree where a left child cannot be red. This limitation makes AA trees simpler to implement than regular red-black trees. An AA tree can also be viewed as a 2-3 tree where a 3-node is encoded as a black node with a red right child.

- **Weight-balanced trees**[8]: In a weight-balanced tree, no subtree can have a weight smaller than a fixed fraction of the weight of its parent tree, where the weight of a tree is its size + 1. (The +1 makes it easier to handle empty subtrees.) One advantage of weight-balanced trees is that the size information stored for balancing is also independently valuable to users of the data structure.

- **Treaps**[4]: A treap is binary search tree where every node has both a key and a randomly-chosen priority. The nodes of a treap are ordered by keys in the normal search tree ordering, and also ordered by priority in

the normal heap ordering (a child has a smaller priority than its parent, but may have a larger or smaller priority than its sibling). The benefit of treaps is that they are unusually easy to implement (especially delete!), but the tradeoff is that the running time of the standard operations is $O(\log n)$ expected time rather than $O(\log n)$ worst-case time.

- **Skip Lists**[17][13, Chp. 13]: A skip list is *not* a binary search tree, but serves a similar purpose. A skip list is similar to a linked list where every node has a pointer to its successor. However, a node in a skip list has multiple pointers to successors at different distances. These extra pointers allow an algorithm to *skip* ahead in the list. Like treaps, the running time of the standard operations in skip lists is $O(\log n)$ expected time rather than worst-case time. Deletions in skip lists are unusually straightforward.

- **Zip trees**[21]: A zip tree is a binary search tree that is isomorphic to a skip list, with algorithms that more closely resemble the operations on skip lists than the normal operations on search trees. In particular, zip trees replace the rotation operations used by most balanced binary search trees with operations for merging (zipping) and unmerging (unzipping) paths through a tree. Like treaps and skip lists, the running times of the standard operations are $O(\log n)$ expected time rather than worst-case time and deletions are unusually straightforward.

Important: The data structures above were ordered to emphasize commonalities between the various alternatives. This order should *not* be taken as a recommended order to use the data structures in successive offerings of a course. For use in a course, it is probably better to choose less related data structures in successive offerings.

2.1 Mix-ins

Once you have chosen a particular data structure, such as AVL trees, there are a number of smaller decisions that provide further sources of variation. We call these "mix-ins", because you can (mostly) mix any combination of these into the main data structure. Each choice will have small or large effects on the implementation.

- **Mutable or immutable**: Many courses will likely default to mutable implementations. However, all but one of these eight data structures can easily be made immutable using path copying[18] because path copying works very well on trees. The exception is a skip list, which is represented as a dag rather than a tree. But a zip tree can be viewed as the tree-version of a skip list and can easily be made immutable.

- **Header node**: One common use of a header node is to keep track of the current root of a mutable tree. That way, even if a particular operation makes a different node the new root, any part of the code that uses the header node can "see" the change. This use of header nodes typically does not apply to immutable trees because each different version of the tree has a different root. However, even immutable trees might use a header node to hold overall information about the tree, such as the size.

- **Parent pointers**: In a binary tree, a node typically includes pointers to its left and right children and may or may not also include a pointer to its parent. Parent pointers often allow operations like search, insert, and delete to be implemented with loops instead of recursion, with search using one downward loop and insert/delete using two loops: a downward loop followed by an upward loop. Note, however, that parent pointers are almost entirely incompatible with immutable data structures.

- **Key vs. key-and-value**: Almost any binary search tree can be adapted to represent either a set or a map. For a set, each node would store a key; for a map, each node would store a key together with an associated value.

- **Lazy delete**[22, p. 41]: A simple way to implement deletion in almost any binary search tree is to set a flag on a node whose element is being deleted, but to otherwise leave the node in the tree. Navigation through a flagged node goes left or right in the usual way, but if the key in the node matches the search key, the search is considered to fail. This scheme works very well as long as there are relatively few such deleted nodes in the tree, but degrades if there are too many deleted nodes. One way to handle this situation is to rebuild the tree whenever the fraction of deleted nodes exceeds some threshold, such as 50%. The rebuilding step takes $O(n)$ time to remove all the deleted nodes and usually restructures the remaining nodes to be perfectly balanced. Despite the fact that the rebuilding step takes $O(n)$ time, the delete operation still runs in $O(\log n)$ amortized time. (However, this analysis does not apply for immutable trees.)

- **Leaf trees**[12, p. 486][11]: All keys (and values, if any) are stored at the leaves of the tree. Interior nodes contain keys that are used for navigation only—a key stored in an interior node may or may not also appear in a leaf node. For example, inserting 3 into a singleton leaf tree containing 1 might result in a tree with 1 and 3 at the leaves and 1 at an interior node. A search method would go left at an interior node if the search key was less than or equal to the element at the node. An advantage of

this scheme is that deletion is greatly simplified because deletion always involves a leaf node.

3 Successors for Standard Heaps

This section describes five potential successors to the standard heap data structure for implementing priority queues. Unlike the standard heap data structure, each of these successors supports an efficient *merge* operation and is implemented with nodes and pointers.

All of these data structures share the same ordering invariant: an item in a parent node has higher or equal priority to items in its children, where higher priority means a larger value in a max-heap or a smaller value in a min-heap. Items in sibling nodes have no implied priority ordering with each other.

- **Leftist Heaps**[7][12, pp. 149–150]: A leftist heap is a heap-ordered binary tree where every left subtree is larger than its corresponding right subtree, either in height[7, 12] or in size (number of nodes)[5]. The height/size is stored in each node, and whenever a right sibling becomes larger than its left sibling, the two subtrees are swapped. Two leftist heaps are merged by merging the rightmost paths of the two trees (as if they were sorted lists) and swapping the children of any node whose right subtree becomes larger than the left subtree as a result of this merge. The `min` (or `max`) operation runs in $O(1)$ worst-case time, and the `insert`, `merge`, and `deleteMin` (or `deleteMax`) operations run in $O(\log n)$ worst-case time.

- **Maxiphobic Heaps**[14]: A maxiphobic heap is a heap-ordered binary tree where every subtree stores its height or size. When two heaps are merged, the root with the higher priority becomes the new root. Of the three remaining trees (the left and right subtrees of that root plus the other tree being merged), the two smaller trees are merged and become one child of the new root with the larger tree becoming the other child of the new root. The running times of the major operations are the same as for leftists heaps.

- **Skew Heaps**[19][13, Chp. 6]: A skew heap is again a heap-ordered binary tree, but, unlike leftist or maxiphobic heaps, a node in a skew heap does not store its height or size. The merge operation is similar to the merge operation of leftist heaps except that the left and right children of *every* node in the two rightmost paths are swapped during the merge instead of only swapping when the right child becomes larger than the left. The running times of the major operations are the same as for leftist heaps

except that the bounds for `insert`, `merge`, and `deleteMin`/`deleteMax` are amortized bounds rather than worst-case bounds.

- **Binomial Heaps**[23][13, Chp. 7]: A binomial heap (also known as a binomial queue) is a collection of binomial trees where each tree has a unique rank. A binomial tree of rank k is a node with k subtrees of ranks $k-1$ to 0. A binomial tree of rank k has size 2^k, creating a close correspondence between a binomial heap and the binary number representing the size of the heap. Insertion in a binomial heap is similar to incrementing a binary number and merging two binomial heaps is similar to adding two binary numbers. The running times of all the major operations are $O(\log n)$ worst-case time, but the `min` (or `max`) operation can easily be improved to $O(1)$ worst-case time. For mutable binomial heaps, a closer analysis improves the running times of `insert` and `merge` to $O(1)$ amortized time.

- **Pairing Heaps**[9][13, Chp. 7]: Pairing heaps are fast and relatively easy to implement, but are quite difficult to analyze[16]. A pairing heap is implemented as a heap-ordered multi-way tree, often using the first-child/next-sibling representation to make it a binary tree. The `insert` and `merge` operations add the the tree with the lower-priority root as the new first child of the other tree's root. The `deleteMin` operation removes the root and merges its children in two passes: the first pass merges the children in pairs from front to back, and the second pass merges the results from the first pass from back to front.

3.1 Mix-ins

Some of the mix-ins for binary search trees still apply to the varieties of heap structures discussed here, sometimes with minor differences.

- **Mutable or immutable**: Leftist heaps, maxiphobic heaps, and binomial heaps can be made either mutable or immutable, but skew heaps and pairing heaps should be considered mutable-only (at least at the undergraduate level) because the amortized bounds of skew heaps and pairing heaps are problematic as immutable data structures.

- **Header nodes**: Because merge combines two different heaps, care must be taken at the end of a merge to reset one of the two header nodes to be empty (if header nodes are being used).

- **Parent pointers**: If either `delete` or `decreaseKey` (below) are to be supported, parent pointers will likely be necessary.

- **Key vs. key-and-value**: Either approach can be used with any of these heaps.
- **Lazy delete**: See **delete** below.
- **Leaf trees**: Not useful for heaps.

Heaps introduce several extra potential mix-ins:

- **Access to the minimum (or maximum) element**: Because a binomial heap involves multiple trees, it takes $O(\log n)$ time to find the `min` (or `max`). This can be reduced to $O(1)$ time by explicitly tracking the overall minimum (or maximum) element. (The other kinds of heaps above already support finding the `min`/`max` in $O(1)$ time.)
- **Delete**: Some heaps support an operation to delete an element that is not the `min`/`max`. However, because heaps do not support searching for an element, such an operation typically requires returning a *handle* to the new node whenever a new element is inserted. That handle then provides immediate access to the desired node. Working with these handles typically forces the implementation to be mutable and to support parent pointers. (If *lazy delete* is used, parent pointers might not be required.)
- **DecreaseKey**: Some heaps support an operation to increase the priority of an existing node. This is used with min-oriented heaps by Dijkstra's algorithm, where increasing the priority means decreasing the key. Like `delete`, supporting `decreaseKey` typically involves handles and forces the implementation to be mutable and to support parent pointers.

4 Conclusion

We have presented a curated selection of data structures for use in a data structures course as follow-ups to the unbalanced binary search trees and standard heaps that are studied in most such courses. Rotating between the various data structures described here provides variety across successive offerings of the course. This variety will largely be invisible to students within a single offering, but nonetheless provides at least two benefits across multiple offerings:

- Enrichment: Teaching staff—including TAs—can benefit from exposure to different data structures than what they learned when they were students in a data structures course.
- Reduction in Cheating: With eight variations on binary search trees and five variations on heaps, it can easily be multiple years between

repetitions of the same base data structure. Factor in mix-ins that have a significant effect on the code and it might be a decade between exact repetitions. The mix-ins also make it substantially harder to find exact code on the internet. Of course, there is no silver bullet; students can still cheat, but if they do, it is more likely to be from somebody else in the same semester, which is easier to detect compared to copying from an earlier semester or from code on the internet.

References

[1] Georgy Adelson-Velsky and Evgenii Landis. "An algorithm for the organization of information". In: *Proceedings of the USSR Academy of Sciences* 146 (1962), pp. 263–266.

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[3] Arne Andersson. "Balanced Search Trees Made Simple". In: *Proceedings of Algorithms and Data Structures, Third Workshop, WADS '93*. 1993, pp. 60–71.

[4] Cecilia R. Aragon and Raimund G. Seidel. "Randomized search trees". In: *30th Annual Symposium on Foundations of Computer Science*. 1989, pp. 540–545.

[5] Seonghun Cho and Sartaj Sahni. "Weight-Biased Leftist Trees and Modified Skip Lists". In: *ACM Journal of Experimental Algorithmics* 3 (Sept. 1998), 2–es.

[6] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd ed. MIT Press, 2009.

[7] Clark Allan Crane. "Linear lists and priority queues as balanced binary trees". Technical Report STAN-CS-72-259. PhD thesis. Stanford University, 1972.

[8] J. Nievergelt and E. M. Reingold. "Binary Search Trees of Bounded Balance". In: *SIAM Journal on Computing* 2.1 (1973), pp. 33–43.

[9] Michael L. Fredman et al. "The pairing heap: A new form of self-adjusting heap". In: *Algorithmica* 1 (1986), pp. 111–129.

[10] Leo J. Guibas and Robert Sedgewick. "A dichromatic framework for balanced trees". In: *19th Annual Symposium on Foundations of Computer Science*. 1978, pp. 8–21.

[11] Anne Kaldewaij and Victor J. Dielissen. "Leaf Trees". In: *Science of Computer Programming* 26 (1–3 May 1996), pp. 149–165.

[12] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Second. Addison-Wesley, 1997.

[13] Dinesh P. Mehta and Sartaj Sahni, eds. *Handbook of Data Structures and Applications*. Chapman & Hall/CRC, 2005.

[14] Chris Okasaki. "Alternatives to Two Classic Data Structures". In: *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. 2005, pp. 162–165.

[15] Chris Okasaki. "Red-black trees in a functional setting". In: *Journal of Functional Programming* 9.4 (1999), pp. 471–477.

[16] Seth Pettie. "Towards a Final Analysis of Pairing Heaps". In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005)*. IEEE Computer Society, 2005, pp. 174–183.

[17] William Pugh. "Skip Lists: A Probabilistic Alternative to Balanced Trees". In: *Communications of the ACM* 33.6 (1990), pp. 668–676.

[18] Neil Sarnak and Robert E. Tarjan. "Planar Point Location Using Persistent Search Trees". In: *Communications of the ACM* 29.7 (1986), pp. 669–679.

[19] Daniel Dominic Sleator and Robert Endre Tarjan. "Self-Adjusting Heaps". In: *SIAM Journal on Computing* 15.1 (1986), pp. 52–69.

[20] Porter Eugene Smith and James H. Graham. "A Simple Balanced Search Tree". In: *Proceedings of the 1993 ACM Conference on Computer Science*. 1993, pp. 461–465.

[21] Robert E. Tarjan, Caleb Levy, and Stephen Timmel. "Zip Trees". In: *ACM Transactions on Algorithms* 17.4 (2021).

[22] Robert Endre Tarjan. *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics, 1983.

[23] Jean Vuillemin. "A Data Structure for Manipulating Priority Queues". In: *Communications of the ACM* 21.4 (1978), pp. 309–315.

[24] John W. J. Williams. "Algorithm 232: Heapsort". In: *Communications of the ACM* 7.6 (1964), pp. 347–348.

Computational Making with Twoville*

Chris Johnson
Department of Computer Science
James Madison University
Harrisonburg, VA 22801
johns8cr@jmu.edu

Abstract

Twoville is a platform for making physical objects using both code and the mouse. The designer writes code to give the object its overall structure but then uses the mouse to manipulate the drawing directly. The code editor and drawing canvas are linked so that changes in one view are immediately visible in the other. Special care is taken to preserve the designer's original expression structure when parameters are manipulated. The output of the program is a scalable vector graphics file, which is sent to a fabrication device and turned into a physical object. This experience report describes the core values that motivate Twoville's design, explores its bidirectional editing algorithm, and offers some examples of how it has been used in summer camps and workshops in the authors' local community.

1 Introduction

Computational making is the use of computational and mathematical thinking in the design and fabrication physical objects. Designers who engage in computational making quantify spatial properties, describe geometric processes, model repeating patterns, and define objects in terms of abstract parameters. They use code to iteratively shape their objects on a computer, and then submit their design file to a fabrication device. The result is a physical artifact.

Figure 1: The Twoville environment showing a lightning bolt icon programmed as an *ungon*, which is a polygon with rounded corners. The code editor on the left is linked to the drawing canvas on the right. Changes made in one view are immediately reflected in the other.

A considerable amount of digital design software is driven purely by the designer's aesthetic tastes. A curve is smoothed out until it feels right. A box is nudged to be nearer to its neighbor. At the other end of the spectrum of design tools are general-purpose programming languages. Such languages place cognitive load on the designer to mentally interpret the actions of the code and visualize the result. In this paper we introduce Twoville, a design tool that sits in the middle of this spectrum. It combines a code editor with an interactive drawing canvas, as shown in Figure 1.

The Twoville editor supports both indirect manipulation of shapes through code and direct manipulation via the mouse. When the designer runs the code, the editor and canvas content are linked together. When the user moves the cursor in the text editor, the corresponding element in the drawing canvas is selected, and its direct manipulation handles appear. The user drags on these handles to alter positions, lengths, and angles. The shape is updated immediately in both the canvas and the code editor. When the code behind a spatial property is a complex expression and not just a literal value, the original expression structure is preserved. For example, if the rotation handle in Figure 1 is dragged, the value of variable `pit` is updated on line 1.

2 Core Values

The syntax, semantics, and user interface of Twoville are shaped by the following core values.

Be explicit. The function call `circle(5, a, 7.5)`, as might be used with many drawing libraries, is implicit. The designer must hold tacit knowledge of the drawing library to understand the significance of the positional parameters. In Twoville, shape parameters are always named. For example, to make a circle, the designer writes this Twoville code:

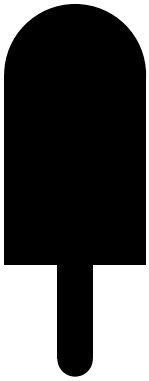
```
circle
  color = :black
  center = [5, a]
  radius = 7.5
```

The indentation indicates that these assignments are applied to properties of the receiver identified at the start of the block. More complete examples are shown in Figure 2.

Encourage exploration of parameter spaces. The first iteration of a design is likely built out of numbers that the designer chooses viscerally. The cost of editing these values should be as low as possible, and changes should produce coherent and immediately visible effects. Iterative refinement helps designers visually explore numerical relationships and develop flexible parameterizations. Twoville allows parameters to be manipulated via mouse events in the drawing canvas.

Support linked views. Editing a design only through text is indirect. Editing a design only through the drawing canvas is laborious and opaque. Twoville offers both algorithmic expressiveness and aesthetic sensibility by linking the code editor and drawing canvas together. Changes in one view are immediately seen in the other. In some design tools, the code generated by actions on the drawing canvas quickly becomes incomprehensible to the designer. To ensure that the designer maintains a cognitive grip on the code, the Twoville drawing canvas only supports actions that edit existing expressions. No actions generate new code structures.

Accommodate the ergonomics of small screens. Young learners tend to access educational software through web browsers on tablets or budget laptops issued by schools. These devices don't offer a lot of screen space. Twoville keeps the code editor and drawing canvas visible at all times. The language



```
rectangle
  center = [0, 0]
  size = [6, 8]
  color = :black
circle
  center = [0, 4]
  radius = 3
  color = :black
rectangle
  center = [0, -6]
  size = [1.5, 4]
  color = :black
circle
  center = [0, -8]
  radius = 0.75
  color = :black
```

(a) Popsicle

```
x = 2
xx = 2 * x
xxx = 3 * x
path
  stroke
    color = :black
    weight = 1
  go.position = :zero
  repeat 5
    line.offset = [xxx, 0]
    line.offset = [0, xxx]
    line.offset = [-xx, 0]
    line.offset = [0, -x]
    line.offset = [x, 0]
    line.offset = [0, -x]
    line.offset = [-xx, 0]
    line.offset = [0, xxx]
```

(b) Frieze

Figure 2: Two Twoville programs. The popsicle is a *Frankenshape*, a composite shape made out of simpler primitives. It requires mathematical understanding to create, but not deeper algorithmic knowledge. The frieze pattern, on the other hand, employs variables, repetition, and relative movement.

grammar minimizes the length of lines to reduce horizontal overflow and keep code visible. For example, in the circle example above, each parameter assignment is required to appear on its own line. This avoids the long lines that may result from named parameters.

Bridge the virtual and physical divide. Many tools for learning computation are entirely virtual. The user interacts with a screen to form an artifact, like an animation or webpage, and the artifact is only ever viewed on that same screen. The making process is disembodied. In Twoville, the artifact is turned into a physical object that has a life and utility independent of computers. Common artifacts include vinyl stickers, acrylic or plywood sculptures, decorated T-shirts, and embroidered greeting cards. These artifacts are fabricated with tools like vinyl cutters, laser cutters, pen plotters, and embroidery machines. Twoville serves all of these tools by generating scalable vector graphics (SVG) files, a format commonly supported by these tools' control software. Two-dimensional fabrication devices are generally faster than 3D printers and CNC routers, and speed is important when serving a classroom full of students.

Serve the local community. Twoville is first and foremost a vehicle for bringing humans together in creative computational activity in libraries or makerspaces. The authors have used it in numerous workshops and summer camps for youth in their local community. Its development follows the needs of the teachers and students using it. It is not meant to be a platform that exists apart from a nearby human teacher.

3 Implementation

Twoville runs in the web browser. It builds extensively on the SVG standard to render the drawing canvas and export design files for fabrication devices. It employs a custom lexer and parser because intimate knowledge of the source code is needed to link the code editor and drawing canvas. Programs are stored locally using the browser's local storage API.

The most interesting aspect of Twoville's implementation is how it supports bidirectional editing. When the designer drags on a manipulation handle, the mouse action is used to update both the underlying SVG element and the code that produces it. One possible approach to updating the code is to replace the right-hand side of the property assignment with a new value derived from the mouse position. For example, the assignment `distance = 2 * x` could be effectively updated to `distance = y`, where `y` is the mouse's distance from the starting position. But such a change would violate the syntactic structure of the designer's original expression. Instead, Twoville preserves the structure as much as possible to ensure that the designer retains a sense of ownership of the code.

The first step in updating the source code is to determine what the new property value p' should be. If the manipulation handle is linked to a distance, then p' is the distance between the mouse's current location and the starting location. If to an angle, then p' is derived from the mouse's angular displacement. If to an absolute position or relative offset, then p' is derived from the mouse's Cartesian location.

The second step is to inspect the expression that produced the original property p. The bidirectional editing algorithm recognizes the following syntactic forms when deciding how to update the code:

1. $p =$ *variable*. The right-hand side of the assignment is not modified. The algorithm instead recurses to the most recent definition of the variable and updates its source expression. Updating this expression will indirectly modify other properties that depend on this same variable.

2. $p = a \oplus b$, where $\oplus$ is a binary operator that can be inverted. The algorithm first decides whether to update expression a or b. The selection

process favors updating literal numbers, as these likely represent "magic" values that the designer is wishing to explore via direct manipulation. If both operands are literals, the rightmost is selected. The designer may lock an operand from being selected by surrounding it with parentheses.

Once an operand has been selected, the mouse position is used to determine the desired target value p'. The operation is inverted to solve for the new operand value. For example, if the operand a is selected, then its new value a' is computed as follows:

$$\begin{aligned} p' &= a' \oplus b \\ p' \ominus b &= a' \oplus b \ominus b \\ p' \ominus b &= a' \end{aligned}$$

3. $p = f(\ldots)$, where f is an operation like a function call or mathematical operation that cannot be easily inverted. The designer's operation is not replaced, but an offset Δ is added to the right-hand side. Its value is solved for as follows:

$$\begin{aligned} p' &= f(\ldots) + \Delta \\ p' - f(\ldots) &= \Delta \end{aligned}$$

4. $p = \textit{number}$. The right-hand side is a literal value that cannot be broken down further. It is replaced by a literal representation of p'.

Bidirectional editing requires considerable runtime support. Each linked expression must be retained in its unevaluated form so that it can be classified. The values of the operands must be retained to solve for the modified values. Each variable assignment must retain the expression that gave it its value.

Some code structures present editing challenges for which the authors have not yet found satisfactory solutions. Loops, in particular, are problematic in that they generate multiple properties and handles that link to the same source expression. A manipulation of one handle may alter the code in undesirable ways for the other instances.

4 Teaching Experiences

Twoville has been used in workshops and summer camps in the authors' local community for several years. Thus far it has only been informally evaluated. Middle school girls who attended an hour-long Twoville workshop completed a non-rigorous followup survey administered by the authors' university. To the question "What was the coolest thing you learned in this workshop?", they offered this sample of responses:

- "how to rotate shapes with code"
- "That it is super easy to create fun shapes & images with coding & then you can print them super quickly."
- "I didnt know that stickers were made with so many intriate shapes"
- "I can make my own waterbottle stickers!"
- "I loved customizing a sticker with code!"
- "How you can make shapes with a grid"
- "Coding isn't that hard and its very fun"
- "The coolest thing that I learned during this workshop is that you can code in lots of numbers and make a pretty design."
- "Finding out the code and effort that goes into this."
- "Codeing"

To the question "What did you like best about the activity?", they offered this sample of responses:

- "seeing my finished sticker"
- "it was very customizable so everyone can pick & create something that suits them."
- "I like how we had freedom to make whatever we wanted into a sticker"
- "ALL of it"
- "I got my own sticker!"
- "You got to choose and make a ton of designs."
- "That I got to make my own sticker."
- "That we got our own stickers"
- "Designing a sticker"
- "making my own design"
- "I liked being able to make my own sticker that I can keep!"
- "That we got to design our own sticker"
- "Coding."
- "getting a sticker"

They frequently invoked the notion of creative ownership by using the words *my*, *our*, *own*, *choose*, and *freedom*. They also found the physical artifact compelling.

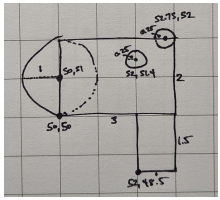
Following are descriptions of several activities the authors have developed that lead to satisfied students and interesting physical objects.

4.1 Frankenshape Stickers

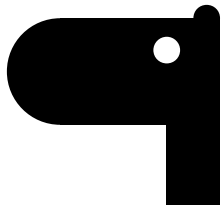
The introductory activity of every workshop and camp builds on participants' prior knowledge of rectangles and circles. Participants design a Frankenshape,

```
rectangle
  corner = [50, 50]
  size = [3, 2]
  color = :black
circle
  center = [50, 51]
  radius = 1
  color = :black
circle
  center = [52.75, 52]
  radius = 0.25
  color = :black
rectangle
  corner = [52, 48.5]
  size = [1, 1.5]
  color = :black
circle
  center = [52, 51.4]
  radius = 0.25
  color = :white
```

(a) Twoville Program



(b) Sketch



(c) Vector Image

(d) Vinyl Sticker

Figure 3: Animal Head

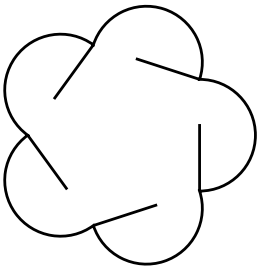
which is a complex shape built out of simpler shapes. In early outreach events, the instructors began by demonstrating how to piece together a cloud or a slice of bread from rectangles and circles. After seeing participants flounder under the combined weight of math and coding, the instructors inserted a *precoding* activity. Before Twoville is shown, the instructor demonstrates drawing a Frankenshape on graph paper and revisits the Cartesian coordinate system. Participants then design their Frankenshape on their own graph paper, labeling the coordinates and lengths. By the time they open Twoville, writing the program feels more like translating than coding from scratch. Examples of Frankenshapes are shown in Figures 2a and 3.

4.2 Paper Sculptures

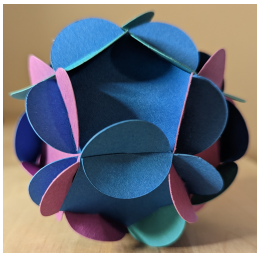
Twoville supports many shapes beyond the circles and rectangles used in the leading Frankenshape activity. These include polygons, polylines, mosaics for stained-glass window effects, text, piecewise curves, and geometric nets. The last of these are flattened polyhedra. Fold lines are scored rather than cut by the cutting tool to produce crisp bends. Tabs are automatically inserted by Twoville on marked sides so that the shape can be glued back into a solid. Most shapes may be described through either Cartesian or turtle geometry. They

```
path
  stroke
    color = :black
    weight = 0.5
  turtle
    position = :zero
    heading = 0
  repeat 5
    curl
      degrees = 180
      radius = 10
    turn.degrees = -108
  back
  turtle
    position = :zero
    heading = 90
  repeat 5
    walk.distance = 12
    fly.distance = 8
    turn.degrees = 72
```

(a) Twoville Program



(b) Vector Image



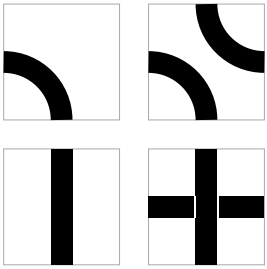
(c) Paper Sculpture

Figure 4: Lobed Dodecahedron

can be transformed and mirrored. Figure 4 shows the versatile `path` command which is used to trace out a pentagonal path. Each side of the pentagon is not a straight line but a half-attached semi-circle. Twelve of these shapes slide together to form a self-locking dodecahedron.

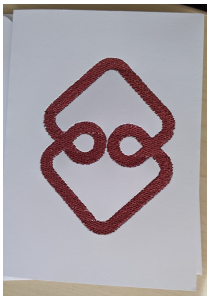
4.3 Knot Design

Mathematical knots seem like an ideal fit for programmatic drawing. They are aesthetically pleasing, culturally significant, and subtly algorithmic. In early incarnations of these workshops, the authors showed participants completed knots and challenged them to reverse engineer what they saw. The results were generally disappointing. Students struggled to identify the points of repetition and failed to accurately relate the lengths of stretches of the knot. In more recent workshops, the authors have pivoted to a different precoding activity: forward engineering with constraints. Participants are given a pile of paper tiles as shown in Figure 5a. They piece together the tiles to form a knot. Once their tiles form a closed curve, they open up Twoville and walk the curve using turtle geometry. This activity has been used to create the plywood maze and



(a) Vector Image

(b) Woodcut Maze



(c) Embroidered Card

Figure 5: Knotwork

embroidered greeting card shown in Figure 5.

4.4 T-shirts

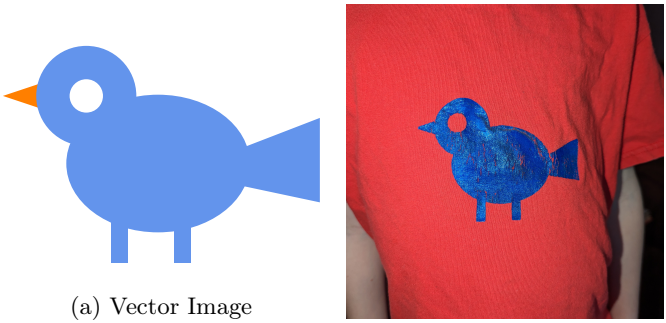
Most participants in Twoville events have been delighted to design and take home a vinyl sticker. This is good because that's about all there's time for in an hour-long workshop. Participants in the week-long summer camps, which permit more intensive projects, have exhibited a deeper degree of satisfaction in making their own T-shirts. They create a design in Twoville and cut it out of vinyl. They keep the positive image as a sticker, but apply the negative image as a stencil to a blank T-shirt. They then spread fabric paint inside the stencil. The shirt dries after a day or two, and the partipants head home with a wearable memorial of their creative coding endeavors. An example T-shirt design and modeled T-shirt are shown in Figure 6.

5 Related Work

The prior work that has informed the development of Twoville can be clustered into three main categories: computational making, programming languages for fabrication, and direct manipulation in user interfaces.

5.1 Computational Making

Rode et al. [14] define computational making as the integration of computational thinking and several making-related activities and skills, namely "aesthetics, creativity, construction, visualizing multiple representations, and understanding materials." Learners who engage in computational making inte-



(a) Vector Image

(b) Painted T-shirt

Figure 6: Bird T-shirt

grate different kinds of thinking, including virtual and physical, algorithmic and aesthetic, and abstract and concrete. Jacobs and Buechley [8] discuss several advantages that computational design systems provide, including precision and automation, generativity and randomness, and parameterization.

Chytas et al. [5] held a series of workshops on parametric design and analyzed the participants' OpenSCAD [13] and BlocksCAD [2] programs to understand the how parametric design contributes to the learning of programming. Half of the projects did not include any loops, and very few used conditional statements. These low frequencies can be explained to some degree by a bias toward novice programmers in experimental environments. The author has also found that many interesting objects can be made without flow control. Not every programming language feature is important in the initial stages of computational making.

A considerable number of projects have engaged learners in computational making via programmable textiles [14, 3, 11]. In some of these projects, the constructed object is made of two distinct elements: a programmable embedded system and a physical housing, like a garment of clothing or a puppet. With Twoville, the physical artifact is itself the product of the computation. Once fabricated, the object is no longer tied to a computer.

Other textile projects allow designers to program the designs themselves. Yu and McCann [17] describe an interface that allows users to see the connection between lines of code and their associated stitches in a visualization of a programmed knitting pattern. Twoville offers a similar linked editor. When the cursor is placed in the code, the corresponding component of a shape is highlighted in the canvas and can be edited visually.

5.2 Programmatic Fabrication

A growing but small number of programming environments have been developed for creating physical artifacts. Some tools confine the design space to two dimensions. The FlatCAD system of Johnson [10], for example, is used to programmatically trace out flat shapes that can be laser-cut and assembled into more complex objects. Turbak et al. [16] describe two blocks languages for generating design files that can be cut or engraved with laser cutters. Other tools support fabrication in three dimensions. Beetleblocks [12] and Madeup [9] both generate printable 3D models using imperative languages inspired by Logo. OpenSCAD and BlocksCAD provide declarative languages in which complex models are composed using simpler primitives.

Johnson [10] observes that programming is not always the most natural means of interacting with a design. A more versatile design tool would additionally allow sketching and direct manipulation.

5.3 Direct Manipulation

Schneiderman [15] explored direct manipulation across a diverse set of professional domains and summarized it as an interactive mode of editing where actions are rapidly executed, immediately observable, and easily reversed. Physically interacting with the system produces direct, visual results. These principles of interactive design have since been applied to integrated development environments (IDEs) to ease the generation and editing of code.

Adam et al. [1] compared direct manipulation interfaces to text interfaces in a study of novice programmers. Half of the subjects used a tool that generated code as they directly interacted with the data. The other half used a text editor. The direction manipulation group solved fewer programming exercises than the text group but achieved higher scores on the exercises they did complete. In exercises involving conditionals and repetition, each editing mode had its advantages. Adam et al. hypothesize that an ideal interface would combine both methods of input.

Hundhausen et al. [7] investigated whether direct manipulation makes programming more accessible than text and whether the knowledge acquired using a direct manipulation interface to program transfers to a text interface. They found that students in an introductory programming course who used direct manipulation completed coding tasks more accurately and quickly than students using text. Further, when this experimental group switched to text for the final exercise, they continued to outperform the control group. *Dual-coding theory* [6], which states that knowledge is strengthened when it is encoded in multiple ways, may explain this positive result.

The Sketch-and-Sketch project of Chugh et al. [4] and Twoville have sim-

ilar goals. Both projects are environments for indirectly coding and directly manipulating vector graphics designs. When a Sketch-and-Sketch program is evaluated, each visual property of a shape—like size and position—is stored with a *trace*, which is an unevaluated representation of the expression that generated it. This expression is composed of one or more parameters. When the designer manipulates the shape using the mouse, the new value of the property and the source expression are used to solve for one of the parameters. The new value replaces the old value in the source code. Twoville differs in several ways from Sketch-and-Sketch. It is an imperative language rather than a functional language, and it relies on simpler heuristics to update ambiguous or complex expressions. These adaptations were intentionally chosen to cater to an audience of computational makers and young learners.

6 Conclusion

Twoville is a freely available web app for learning mathematics and computer science through the lens of making physical objects. Its development is driven by a core set of values that reflect the physicality of learners. An interface driven by direct manipulation permits visceral exploration of parameter spaces. Precoding activities conducted away from the computer build on learners' prior experiences with shapes, turning coding into more of a translation step than a completely new activity. Twoville's physical output has a presence that outlives the computing session, motivating learners who wish for something more than virtual.

For future work, the authors plan to better scaffold the making process. Others have demonstrated that sketching is key to developing spatial skills. While sketching has already been a part of precoding activities, designers sometimes lose sight of their goal while tracing the design with code. The authors believe they can help designers plan their next steps by adding a sketching layer to the drawing canvas. The learners will draw a rough outline of their shape in this layer prior to coding. Additionally, some designers have expressed a wish to be able to load reference images into the canvas that they can trace. This need is especially felt when shapes are sculpted with Bézier curves, which are indirectly manipulated by one or two control points.

References

[1] Michel Adam, Moncef Daoud, and Patrice Frison. "Direct Manipulation versus Text-Based Programming: An Experiment Report". In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '19. Aberdeen, Scotland Uk: Associa-

tion for Computing Machinery, 2019, pp. 353–359. ISBN: 9781450368957. DOI: 10.1145/3304221.3319738. URL: https://doi.org/10.1145/3304221.3319738.

[2] BlocksSCAD. http://www.blockscad3d.com. Accessed 16-April-2020.

[3] Leah Buechley. "LilyPad Arduino: rethinking the materials and cultures of educational technology". In: *Learning in the Disciplines: Proceedings of the 9th International Conference of the Learning Sciences, ICLS '10, Chicago, IL, USA, June 29 - July 2, 2010, Volume 2*. Ed. by Susan R. Goldman et al. International Society of the Learning Sciences / ACM DL, 2010, pp. 127–128. URL: http://dl.acm.org/citation.cfm?id=1854566.

[4] Ravi Chugh et al. "Programmatic and direct manipulation, together at last". In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2016* (2016). DOI: 10.1145/2908080.2908103. URL: http://dx.doi.org/10.1145/2908080.2908103.

[5] Christos Chytas, Ira Diethelm, and Alexandros Tsilingiris. "Learning programming through design: An analysis of parametric design projects in digital fabrication labs and an online makerspace". In: *2018 IEEE Global Engineering Education Conference (EDUCON)*. IEEE. 2018, pp. 1978–1987.

[6] James M Clark and Allan Paivio. "Dual coding theory and education". In: *Educational psychology review* 3.3 (1991), pp. 149–210.

[7] Christopher D. Hundhausen, Sean F. Farley, and Jonathan L. Brown. "Can Direct Manipulation Lower the Barriers to Computer Programming and Promote Transfer of Training? An Experimental Study". In: *ACM Trans. Comput.-Hum. Interact.* 16.3 (Sept. 2009). ISSN: 1073-0516. DOI: 10.1145/1592440.1592442. URL: https://doi.org/10.1145/1592440.1592442.

[8] Jennifer Jacobs and Leah Buechley. "Codeable Objects: Computational Design and Digital Fabrication for Novice Programmers". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2013, pp. 1589–1598. ISBN: 9781450318990. URL: https://doi.org/10.1145/2470654.2466211.

[9] Chris Johnson. "Toward Computational Making with Madeup". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 2017, pp. 297–302.

[10] G. Johnson. “FlatCAD and FlatLang: Kits by code”. In: *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. 2008, pp. 117–120.

[11] Yasmin B. Kafai et al. “A Crafts-Oriented Approach to Computing in High School: Introducing Computational Concepts, Practices, and Perspectives with Electronic Textiles”. In: *ACM Trans. Comput. Educ.* 14.1 (Mar. 2014). DOI: 10.1145/2576874. URL: https://doi.org/10.1145/2576874.

[12] Duks Koschitz and Eric Rosenbaum. “Exploring algorithmic geometry with ’Beetle Blocks:’A graphical programming language for generating 3D forms”. In: *Proceedings of the 15th International Conference on Geometry and Graphics*. 2012, pp. 380–389.

[13] OpenSCAD. http://www.openscad.org. Accessed 16-April-2020.

[14] Jennifer A. Rode et al. “From Computational Thinking to Computational Making”. In: *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. UbiComp ’15. Osaka, Japan: Association for Computing Machinery, 2015, pp. 239–250. ISBN: 9781450335744. DOI: 10.1145/2750858.2804261. URL: https://doi.org/10.1145/2750858.2804261.

[15] Ben Shneiderman. “1.1 direct manipulation: a step beyond programming languages”. In: *Sparks of innovation in human-computer interaction* 17 (1993), p. 1993.

[16] F. Turbak et al. “Blocks languages for creating tangible artifacts”. In: *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*. Sept. 2012, pp. 137–144. DOI: 10.1109/VLHCC.2012.6344500.

[17] Tianhong Catherine Yu and James McCann. “Coupling Programs and Visualization for Machine Knitting”. In: *Symposium on Computational Fabrication*. SCF ’20. Virtual Event, USA: Association for Computing Machinery, 2020. ISBN: 9781450381703. DOI: 10.1145/3424630.3425410. URL: https://doi.org/10.1145/3424630.3425410.

Experiential Learning in Undergraduate Accessibility Education: Instructor Observations*

Saad Khan, Heather Moses, Samuel Malachowsky, Daniel Krutz
Department of Software Engineering
Rochester Institute of Technology
Rochester, NY 14623
{sk6786,hlm8500,samvse,dxkvse}@rit.edu

Abstract

Although creating accessible software is imperative for making software inclusive for all users, problematically, the topic of accessibility is frequently excluded from computing education. This leads to scenarios where students are not only unaware of how to create accessible software, but also do not see the need to create accessible software. To address this challenge, we have created a set of educational Accessible Learning Labs (ALL) that are systematically designed to not only inform students about fundamental topics in creating accessible software, but also to demonstrate the importance of creating accessible software.

1 Introduction

Approximately 20% of the world population has a disability [8]. Unfortunately, much of the software being developed today is not sufficiently accessible for all users [12, 14, 15]. Furthermore, research indicates that while many computing instructors have a desire to include accessibility in their curriculum, they often lack access to teaching materials to include this important topic [19, 33]. To

address this limitation in computing accessibility education, we have created five educational accessibility labs that we refer to as the Accessible Learning Labs (ALL). Each of these labs addresses a different disability and has the primary goal of demonstrating the need to create accessible software, while also providing students with technical concepts for creating accessible software. The labs have been systematically designed to be easily integrated into a wide variety of learning scenarios, affording them the ability to positively impact a range of learners. One only requires an internet connection and a web browser in order to access the lab activities and materials, making adoption easy for students, instructors and individual learners. Created labs and project materials are publicly available on the project website [4]. The labs each address a fundamental accessibility challenge and contain the following components: I) Background information on the accessibility issue being addressed, II) A sample application emulating the accessibility challenge to the user, III) Empathy-creating supplementary material, and IV) Concluding quiz.

Over two recent semesters, we have included our labs in 16 Computer Science 2 (CS2) offerings, with 321 students participating at our university. Additionally, we have partnered with instructors in several other computing and non-computing focused courses, both at our university and collaborating universities to include our ALL material in their curricula. The material was included in both conventional and online course offerings.

This publication reports on observations from the inclusion of our material in these settings. Some of our primary observations include: I) Instructors enjoyed the à-la-carte nature of the labs, II) The 30-60 minute time required for lab completion supported their adoptability, III) Instructors and students stated their appreciation for the hosted, self-contained nature of the labs, and IV) Accessibility 'empathy-creating' material can be a motivating factor for students. In contrast to existing publications [32, 31] relating to our Accessible Learning Labs (ALL), this submission differentiates itself by providing insights and observations relating to the inclusion of our material from the instructor's prospective rather than evaluating the labs from a statistical perspective.

To summarize, this work makes the following contributions:

- **Introduction to the created educational material:** These self-contained educational accessibility labs are publicly available on the project website[1] and require no special tools or custom configurations for adoption since they may be accessed using only a browser.
- **Preliminary instructor observations from the inclusion of the created material:** We describe preliminary instructor observations from the use of our labs. These observations demonstrate many of the positive aspects of our material and information for potential adopters.

[1] `https://all.rit.edu`

- **Observational recommendations about how to best include the material:** We discuss recommendations from which adopting instructors and those teaching accessibility-related curriculum can benefit.

2 Related Work

Several projects have developed accessibility-related educational materials, focusing on various methods of accessibility education [10, 13, 28, 5]. Our labs differ from this existing material in several ways, including: I) None of these existing materials offer a complete educational experience for the adopter, II) Our labs are hosted, making them publicly available using only an internet connection and browser, and III) Our labs contain empathy-creating material to motivate participants about the importance of this topic.

There are other educational accessibility materials that are publicly available. The 'Teach Access Tutorial' provides designers and developers with lessons and exercises that teach basic accessible web development practices [35]. *AccessComputing* [9] is an alliance that supports students with disabilities to learn computing. The group helps make computing courses more accessible to students with disabilities, along with providing assistance to instructors teaching accessibility-related topics. For example, AccessComputing provides curriculum resources (*e.g.,* educational components) that instruct how to create accessible mobile applications [16]. However, to our knowledge, no existing resources provide a complete educational experience (*e.g.,*, experiential activity, lecture slides, etc.) as we have in our Accessible Learning Labs (ALL).

Our material is not the first attempt to integrate accessibility into existing courses, such as web design [30, 36], HCI [28], and software engineering courses [26] using various pedagogical methods such as lectures [36], programming activities [16], and projects [28, 26, 23]. Research has found that when students interact with individuals with disabilities (*e.g.,* project stakeholders), they better understand and apply accessibility principles in their work [26, 23]. Despite these efforts and previously published work, incorporating accessibility in computing courses is still an individual effort that is driven by faculty who have experience in accessibility or a related field, *e.g.,* HCI [29], constituting only approximately 2.5% of instructors [33]. The self-contained, easy-to-adopt nature of our experiential labs will support these instructors who are not accessibility experts in including accessibility into their curriculum.

Our labs utilize experiential learning principles that have been demonstrated to be effective in computing education [11, 21]. Compared with other approaches, experiential learning has been shown to be more engaging for students [22], and to better support student retention of information [34, 18].

Several of the challenges for teaching and learning accessibility in computing

education were identified by Lewthwaite *et al.* [24]. This research stated that accessibility education in computing creates a set of challenging and unique characteristics. Our work differs in that we do not focus on identifying specific challenges in computing accessibility education, but focus on presenting and evaluating a set of unique experiential educational materials.

3 Lab Structure, Topics and Goals

In the following sections, we will describe the primary objectives of the labs and their structure.

3.1 Lab Goals

The educational accessibility labs have several primary goals:

1. *The labs do not require special software or hardware:* Easy adoption of the material is supported by the fact that all material is hosted on our project servers, and requires only a browser and internet connection for usage.
2. *Adopters need only basic programming skills:* Students possessing a diverse set of experience levels and skill sets should be made aware of the importance of creating accessible software and be provided foundational skills to create accessible software.
3. *The labs should fit into already crowded computing courses:* Each lab is designed to take approximately 30-60 minutes, and the instructor may select the lab components that they would like to utilize in an à-la-carte fashion inside or outside of the classroom. The succinctness of the labs will enable them to fit into courses that are already heavily time-constrained.
4. *The labs include all instructional content necessary for adoption:* To support instructors and participants to easily utilize the created material, all necessary content (*e.g.,* lecture slides, background information, activity, quiz, etc.) are provided.
5. *The labs should demonstrate the importance of creating accessible software:* Teaching participants about *how* to create accessible software is important, but participants should also be made aware *why* it's essential to create accessible software.

3.2 Lab Structure and Components

Each lab is comprised of several components designed to inform and motivate students about the topic of accessibility. The steps for lab completion are described below.

1. **Background Instructional Material:** The lab explains the accessibility topic addressed in the activity. The lab introduces the accessibility guideline that should have been applied to avoid the accessibility defect.
2. **Activity:** Students interact with the web-hosted lab following these steps:
   (a) **Students interact with software:** Students interact with the software without any accessibility emulation feature, experiencing the software as a person of typical ability would. Students are asked to perform a simple task in the application.
   (b) **Students experience accessibility challenges through an emulation feature:** Each lab contains a feature to emulate the addressed accessibility topic as closely as possible. The objective is to demonstrate the adverse impacts of inaccessible software first-hand.
   (c) **Details are provided on how to repair the application:** Students are then provided best practices and guidelines to repair the encountered accessibility issue.
   (d) **Students repair the accessibility problem:** Students repair the inaccessible portion of the lab and gain experience of refactoring software.
   (e) **Students use the repaired version of the software with the emulation feature active, but with their modifications:** The student experiences and evaluates the impact of their alterations in making the software more accessible and evaluates the impact of their changes. This instills confidence in the student to develop accessible software.
3. **Empathy-creating supplementary material:** Empathy-creating content has been found to demonstrate the importance of creating accessible software [32, 31]. In our labs, this is accomplished through testimonials from people (age 18-22) with the addressed accessibility challenge.
4. **Quiz:** Each lab contains an optional web-based quiz, providing a readily available mechanism to assess student learning and the completion of the defined educational objectives. A screenshot of the participant's quiz score may also be submitted to the instructor to demonstrate their completion of the lab activity.

3.3 Lab Topics

The five created labs address the topics of:

- Lab 1: Visual cues to make software accessible to Deaf/Hard-of-Hearing users.

- Lab 2: Making software accessible to users who are colorblind.
- Lab 3: Making software accessible to users who are blind.
- Lab 4: Making software accessible to users who face dexterity issues.
- Lab 5: Making software accessible to users with cognitive impairments.

4 Classroom Inclusion Details

4.1 Conventional Classroom Inclusion Details

The created material was used in 16 sections of a CS2 course at our institution with a total of 321 students participating. Lab #1 (Deaf/HH) was used in our spring course offerings, while Lab #2 (Colorblindness) was used in the fall sections. These labs were conducted in 16 single-day class sessions of the CS2 course, approximately at the week 11 mark of the 14-week course term. The discussed observations (Section 5) are derived from these offerings. Approximately one-third of the sections were led by members of the project team, one-third were taught by a team member in cooperation with the regular course instructor, and the remaining sections were conducted primarily by the regular course instructor where feedback on the offerings were provided to the project team. Lab #1 and Lab #2 were also used in online course offerings at several collaborating institutions.

4.2 Online Course Inclusion Details

As with most institutions, courses at our university were moved online due to the COVID-19 pandemic. Our labs were included in a small summer session of "Ethics in the Digital Era" (DHSS 103) [7] . The primary focus of the course is to examine various contemporary and global issues of digital citizenship and new ethical challenges raised by digital technology [3] . This course offering was entirely online and the activity was completed asynchronously. The inclusion of our ALL material in this course enabled us to not only evaluate our labs in an online course format, but also to understand their impact on non-computing focused students.

5 Inclusion Observations

We describe our observations from the inclusion of our created material, along with those from collaborating institutions with early adopters of our Accessible Learning Labs (ALL). The objective of presenting our own classroom inclusion observations is to assist not only those adopting our material, but also for others incorporating other accessibility material. We recount observations from working with other instructors during their adoption of our material in order to

assist others creating and disseminating experiential education material who may encounter similar challenges. The discussed observations and findings are unique from our previous publications [32, 31] in that this work distinctly focuses on instructor observations and qualitative information regarding the inclusion of the material.

5.1 Observations From Our Classroom Inclusion

We will next discuss some of the primary instructor observations from the inclusion of our material into 16 sections of CS2 at our institution. This information is derived directly from instructor observations and student conversations. The following are several of the primary instructor observations, including student quotes supporting these observations.

1. **Even a brief experiential accessibility activity can spark student interest in the topic:** Before using each of the labs, we spoke with students about how they felt about the topic of software accessibility. Many students seemed disinterested in the topic, and even at the notion of conducting the activity. Soon after beginning the activity, it was easy to see that students became more interested in the topic and became more engaged in the activity. It was also interesting to see students becoming excited about the activity, their inabilities to interact with the inaccessible software and how their alterations made the software much more usable. Instructors also recognized a noticeable difference in student engagement between their conversations before and after conducting the activity. These conversations also led to many students asking about future courses in these areas.
2. **Each accessibility topic provided similar benefits:** Regardless of the lab topic, we did not observe any discernible difference in the benefits provided to the participant from either an observational or statistically significant perspective. This is fairly surprising as RIT is home to the National Technical Institute for the Deaf (NTID) [27], and is home to a much larger population of Deaf/Hard-of-Hearing students than is found at most institutions. Therefore, most students frequently attend classes and conduct group projects with their Deaf/HH classmates. Due to this existing familiarity with Deaf/HH students, instructors found it reasonably surprising that this activity provided similar benefits as the 'blindness' lab did (as students were much less familiar with students from this population). Before conducting the activity, there was doubt about whether it could instill empathy due to the existing familiarity with the Deaf population.

3. **Empathy-creating material is important in accessibility education, especially when the student with the disability is a peer:** The empathy-creating component was very helpful in demonstrating the need to create accessible software. Students also appreciated that this content came from students of their age group. Instructors noticed that while student interest in accessibility grew throughout the activity, a catalyst for this interest appears to have been this empathy-creating material, as students became more discernibly engaged after interacting with it. In conversations with students, they unsurprisingly stated that they found this content to be much more relatable since it came from their age group.
4. **The 30-60 minute time required to complete the labs made them easily adoptable in the classroom:** As in many introductory computing courses, including yet another topic in an already crowded curriculum was challenging. The labs required approximately 30-60 minutes to complete. The concise nature of the labs made it easier to convince instructors to incorporate them in their curriculum.
5. **The à-la-carte nature of the labs supports easy adoption:** The components may be utilized in an à-la-carte fashion among adopters, allowing them to select various lab components as they desire. We have found that this is beneficial in several ways:

    - **When pressed for time, instructors may only select some components:** The adaptable nature of the labs enable instructors to utilize only specific components if they are already pressed for time. Foundational computing courses are frequently pressed for time, making the à-la-carte nature of the labs more imperative.
    - **The adaptable nature of the labs supports easy inclusion in a variety of settings:** Instructors may choose to assign only individual components of the lab, enabling them to customize the learning experience as they see fit. The lab material supports inclusion in classroom settings that are time-limited through the use of more lab components.
    - **Instructors can choose the lab components that they feel are most appropriate for the course:** While we only included our material in a CS2 offering, instructors have stated that they will be more likely to include the labs in their other course offerings since they can not only choose the topics that they feel are most appropriate, but they can also tailor the specific lab components to the needs of their course.

6. **Instructors and students stated their appreciation for the hosted, self-contained nature of the labs:** Since lab content is entirely

hosted on our servers and requires only a browser and internet connection for usage, adoption is extremely easy and requires virtually no setup or configuration time. Students and instructors stated that they appreciated this ease of use and that it made it easy to simply begin the learning activity without any necessary configurations. Instructors stated that they felt that this made it much more likely that they would include these labs in their curriculum, especially since they would not have to devote any time for preparation or configuration.

7. **The self-contained nature of the material made it easier to persuade instructors to adopt it:** Instructors of courses such as CS2 are quite understandably not always amenable for including new material in their curriculum. This is frequently due to time restrictions, or even lack of confidence in teaching a new area. Although accessibility is a vital computing topic, it is often excluded from formal undergraduate education [20, 6]. When speaking with potential adopting instructors, we found that while most instructors stated that they recognized the importance of accessibility, it was the easy-to-adopt nature of the ALL material that convinced them to allocate course time to the material. While impossible to completely discern, this was likely due to a mixture of not requiring the adopting instructor to learn new material, that the content would only require 30-60 minutes of class time, that it could be incorporated in the class in a variety of fashions (*e.g.,* in person, online, blended), and that it did not require the instructors or students to install anything.

These observations are supported by various student quotes collected at the conclusion of the activity:

> "The interactive portion allows better understanding for the need to consider accessibility when creating software."
>
> "The colorblindness activity was very informative and provided a good example of how to easy it is to create something more accessible."
>
> "I got to actually test the contrast between two colors and see how a colorblind person perceived the colors. It was eye-opening!"
>
> "I like the new knowledge about color blindness. I have seen the option in one of my games. Now, I understand why that option was available."
>
> "The activity taught me that there are people who could benefit a lot by taking their deficiencies into consideration in terms of software."

> "I think the awareness that I got about how important it is to like consider color blindness and how essential it is to know how to create software which is like considering their condition. This really gives me something to really research on."

> "I learned how easy it is, and how important it is to take into account different disabilities that potential consumers might have."

The labs were included in several ethics-focused virtual course offerings in the summer of 2020. The fact that the labs do not require substantial technical background made it much easier for the non-computing focused students to complete the labs. Due to the small number of participants in the virtual offering, it was not practical to perform any statistical analysis on student feedback. However, informal discussions with students and the adopting instructor indicated general satisfaction with both the labs and their format. Interestingly, several students began to profoundly question why accessibility-related problems continue to remain a problem in much of the software created today. This led to further discussion in the virtually-held ethics class regarding the ethical implications of inaccessible software.

5.2 Reflections from Interacting with Adopting Instructors

We will next describe some of our observations from working with instructors not only at our institution, but at other early adopting institutions as well. The objective of describing these observations is to not only present insights from those working with our material, but to present some observations that can help those creating and disseminating other experiential educational material.

While everyone that we spoke to about even possibly adopting our educational accessibility material agreed that creating accessible software was an important topic, it was still frequently challenging to convince them that their courses had enough time and/or resources for the inclusion of accessibility-focused material. Instructors of foundational computing courses(*e.g.,* CS1 & CS2) are understandably typically very pressed for time in their classes and do not have much time for new material. This issue was exacerbated by the COVID-19 pandemic, due to the new challenges and increased workload being encountered by instructors across the United States. While instructors were definitely interested in providing robust educational material to their students, most were primarily weary of any additional work that they would need to do to include new material into their courses. This consisted of concerns regarding the need to learn the material, or technical resources that would be required.

During interactions with collaborating instructors, we frequently found that their biggest concern was ease of adoption and technical competency for non-computing students. To alleviate many of the concerns from potential adopting

instructors, we found that it was even more paramount to discuss the ease of adoption for instructors. Potential adopters appreciated the hosted web-based nature of the material that supports its adoption from a technical perspective since there is nothing that they or their students are required to install. We found it to be reasonably surprising that few potential adopters were enthusiastic about the ability of the material to support out-of-class activities, such as homework assignments.

5.3 Discussion and General Reflections

We learned several valuable lessons regarding the best practices for including accessibility-focused materials in classrooms. These recommendations will likely prove beneficial for not only those including our created material, but for those including other accessibility-focused material in their curriculum as well. Educators should place special relevance on the importance of creating accessible software. As supported by our previous publications [32, 31], this could come in the form of creating empathy or even from more traditional methods. For example, the instructor began the session by telling a short story about inaccessible software to demonstrate the relevance and importance of software accessibility. Additional discussion points include how the task of ensuring the creation of accessible software is everyone's responsibility, and not just those of accessibility 'experts'.

Discussions regarding how industry values accessibility and prefers to hire an accessibility-literate workforce have also been found to be beneficial, especially with lower division students. Discussions regarding how accessibility is similar to other topics such as cybersecurity, testing, etc. in being a central component of software development have also been found to be beneficial. Finally, creating accessible software is creating 'inclusive' software. Software that is not accessible restricts its usage by any portion of the population and contains both ethical and legal/monetary implications (*e.g.,* less accessible may mean less customers/users).

Since RIT houses the National Technical Institute for the Deaf (NTID) [27], we regularly had the opportunity of having many Deaf/HH and other students with self-disclosed disabilities participate in the sessions that we offered. During these sessions, we made several interesting observations that may be beneficial not only for those utilizing our material, but for those discussing accessibility in education in general as well. First and foremost, instructors should allow students to determine their own self-disclosure policy of their disability, and not publicly identify, or 'out', the student with disabilities themselves, even in cases when it may seem fairly obvious the student has a disability (*e.g.,* they are using an interpreter). This is due to the need for instructors to take lead in the creation of an inclusive classroom as well as protecting the student and in-

structor against possible legal ramifications. Fairly surprisingly, we found that many students publicly self-disclosed a disability during classroom discussions of that disability, even when it was not obvious to the class (*e.g.,* colorblindness). They were frequently willing to provide supporting evidence/stories about how inaccessible software adversely impacted their lives.

One of our goals for the project is to eliminate the misconception that creating accessible software is 'for someone else'. It should be demonstrated to students that all members of the project, regardless of roles, have a part to play in creating sufficiently accessible software. This ranges from software designers who should choose 'safe' colors [25], to testers who should construct test plans to check for proper accessibility. One challenge for the construction and inclusion of our created materials was how to properly demonstrate this to students.

An unexpected benefit was that the labs fostered discussions and an understanding of what it meant to be a person with a disability. Unfortunately, we observed that many students (especially lower division) didn't even understand what software accessibility was. We believe that a crucial first step for developers in creating accessible software is to have a more fundamental understanding of the needs of their entire target audience, and from a more fundamental perspective, understand what it means to be a person with a disability.

We firmly believe that in order to create inclusive software, developers should have a foundational understanding for the needs of others. While far from being a perfect or exhaustive activity, we have observed that material such as ours can help to foster such discussions that will result in a better understanding of the needs of others.

There is a demonstrated government and public need to create accessible software [1, 2, 14, 17]. Unfortunately, educators generally do not sufficiently address these limitations by adequately including topics regarding accessibility in their curricula [19, 33]. Based on our observations, we believe that the easily adoptable nature of our lab material is paramount in reducing these limitations and can help lead the creation of more inclusive software.

We also firmly believe that the adoption barrier of accessibility-focused educational material should be lowered as much as possible to support the inclusion of this crucial and fundamental topic. We found that discussing the importance of using accessibility practices during every stage of software development (ie. requirements, design, development) helped to interest students who may not have realized the relevance of accessibility in those areas.

5.4 Analysis Results

In our previous study involving 276 students, our created material and lab format demonstrated their abilities to both inform and motivate students in a CS2 classroom about the benefits of accessibility. In this work, a statistical analysis (t-test) found that our material is more effective in informing students about foundational accessibility principles, while activities containing empathy-creating material have a higher universal positive effect on students. Further details regarding this analysis may be found in our previous, analysis-focused publications [32, 31].

6 Conclusion

This work provides instructor observations from the inclusion of our Accessible Learning Labs (ALL) in 16 sections of a CS2 course at our institution comprised of 321 students, along with the more limited inclusion in several other courses and institutions.

Acknowledgements

This material is based upon work supported by the United States National Science Foundation under grant #1825023.

References

[1] The ADA and section 508. https://508compliantdocumentconversion.com/americans-with-disabilities-act/.

[2] Assistive technology act law. http://www.ataporg.org/ATLaw.

[3] Digital humanities and social sciences bs - curriculum. https://www.rit.edu/study/curriculum/454b714d-0dcd-452c-97f4-3da43cfe7593.

[4] Hidden for anonymous review.

[5] Mozilla accessibility. https://developer.mozilla.org/en-US/docs/Learn/Accessibility/.

[6] Teaching accessibility: A call to action from the tech industry. http://yahooaccessibility.tumblr.com, September 2019.

[7] Rit digital humanities and social sciences bs - curriculum, 2020.

[8] World report on disability. https://www.who.int/disabilities/world_report/2011/report/en/, 2020.

[9] AccessComputing. https://www.washington.edu/accesscomputing/resources/teach-access, September 2019.

[10] P. R. Bohman. *Teaching accessibility and design-for-all in the information and communication technology curriculum: Three case studies of universities in the United States, England, and Austria.* Utah State University, 2012.

[11] W. T. Botelho, M. d. G. B. Marietto, J. C. d. M. Ferreira, and E. P. Pimentel. Kolb's experiential learning theory and belhot's learning cycle guiding the use of computer simulation in engineering education: A pedagogical proposal to shift toward an experiential pedagogy. *Computer Applications in Engineering Education*, 24(1):79–88, 2016.

[12] S. Burgstahler. Designing software that is accessible to individuals with disabilities. `http://www.washington.edu/doit/designing-software-accessible-individuals-disabilities`.

[13] S. E. Burgstahler. *Universal design in higher education: From principles to practice*. ERIC, 2015.

[14] R. Calvo, F. Seyedarabi, and A. Savva. Beyond web content accessibility guidelines: Expert accessibility reviews. In *Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*, DSAI 2016, pages 77–84, New York, NY, USA, 2016. ACM.

[15] D. R. Commission. *The Web: Access and Inclusion for Disabled People ; a Formal Investigation*. Stationery Office, 2004.

[16] Y. N. El-Glaly, A. Peruma, D. E. Krutz, and J. S. Hawker. Apps for everyone: Mobile accessibility learning modules. *ACM Inroads*, 9(2):30–33, Apr. 2018.

[17] V. L. Hanson and J. T. Richards. Progress on website accessibility? *ACM Trans. Web*, 7(1):2:1–2:30, Mar. 2013.

[18] K. Hawtrey. Using experiential learning techniques. *The Journal of Economic Education*, 38(2):143–152, 2007.

[19] S. Kawas, L. Vonessen, and A. J. Ko. Teaching accessibility: A design exploration of faculty professional development at scale. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, pages 983–989, New York, NY, USA, 2019. ACM.

[20] S. Keith, G. Whitney, and A. Petz. Design for all as focus in european ict teaching and training activities. 2009.

[21] A. Y. Kolb and D. A. Kolb. Learning styles and learning spaces: Enhancing experiential learning in higher education. *Academy of management learning & education*, 4(2):193–212, 2005.

[22] L. Laird and Y. Yang. Engaging software estimation education using legos: A case study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 511–517, New York, NY, USA, 2016. ACM.

[23] J. Lazar. Using community-based service projects to enhance undergraduate hci education: 10 years of experience. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pages 581–588. ACM, 2011.

[24] S. Lewthwaite and D. Sloan. Exploring pedagogical culture for accessibility education in computing science. In *Proceedings of the 13th Web for All Conference*, page 3. ACM, 2016.

[25] J. Liu. Color blindness & web design. https://www.usability.gov/get-involved/blog/2010/02/color-blindness.html, March 2010.

[26] S. Ludi. Introducing accessibility requirements through external stakeholder utilization in an undergraduate requirements engineering course. In *29th International Conference on Software Engineering (ICSE'07)*, pages 736–743. IEEE, 2007.

[27] NTID. The national technical institute for the deaf. http://www.ntid.rit.edu/.

[28] G. M. Poor, L. M. Leventhal, J. Barnes, D. R. Hutchings, P. Albee, and L. Campbell. No user left behind: Including accessibility in student projects and the impact on cs students’ attitudes. *Trans. Comput. Educ.*, 12(2):5:1–5:22, Apr. 2012.

[29] C. Putnam, M. Dahman, E. Rose, J. Cheng, and G. Bradford. Teaching accessibility, learning empathy. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers; Accessibility*, ASSETS '15, pages 333–334, New York, NY, USA, 2015. ACM.

[30] B. J. Rosmaita. Accessibility first!: A new approach to web design. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, pages 270–274, New York, NY, USA, 2006. ACM.

[31] W. Shi, S. Khan, Y. El-Glaly, S. Malachowsky, Q. Yu, and D. E. Krutz. Experiential learning in computing accessibility education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, ICSE '20, page 250–251, New York, NY, USA, 2020. Association for Computing Machinery.

[32] W. Shi, S. Malachowsky, Y. El-Glaly, Q. Yu, and D. E. Krutz. Presenting and evaluating the impact of experiential learning in computing accessibility education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '20, page 49–60, New York, NY, USA, 2020. Association for Computing Machinery. Distinguished Paper Award.

[33] K. Shinohara, S. Kawas, A. J. Ko, and R. E. Ladner. Who teaches accessibility?: A survey of u.s. computing faculty. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 197–202, New York, NY, USA, 2018. ACM.

[34] L. B. Specht and P. K. Sandlin. The differential effects of experiential learning activities and traditional lecture classes in accounting. *Simulation & Gaming*, 22(2):196–210, 1991.

[35] T. A. Tutorial. https://teachaccess.github.io/tutorial/.

[36] Y. D. Wang. A holistic and pragmatic approach to teaching web accessibility in an undergraduate web design course. In *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, pages 55–60, New York, NY, USA, 2012. ACM.

Practical Machine Learning for Liberal Arts Undergraduates*

Mark Sherman[1], Alyssa Hogan[1],
Jamison O'Sullivan[2], Samantha Schumacher[1]
[1]Emmanuel College
Boston, MA 02115
{shermanm,hogana,schumachers}@emmanuel.edu
[2]Boston College
Boston, MA 02467
osullijg@ubc.edu

Abstract

Liberal arts education provides students with many interdisciplinary problem-solving skills, but utilizing high-level computational tools like machine learning (ML) remains largely inaccessible without a significant background in computer science. We present a course to bridge that gap: a full-semester undergraduate course that teaches the concepts of ML and deep learning with emphasis on real-world application and ethical considerations. The course is low-code, exploring concepts and applications through a collection of visual tools. Early outcomes show that after this course students are equipped to learn code-based systems, feel empowered to understand and identify misunderstandings in popular AI discourse, and can design ML-based solutions for data-oriented problems in their fields.

1 Introduction

Computer science has evolved through interdisciplinary application which can fit naturally within the liberal arts curriculum [17], creating a mutually ben-

eficial relationship for both computer science and the liberal arts. Recently, machine learning (ML) has been applied to many classically liberal arts disciplines, such as psychology, biology, philosophy, and linguistics [17]. Without an understanding of machine learning, it is becoming increasingly difficult to apply these skills in future careers.

Our course is accessible to non-computing majors by democratizing concepts of machine learning that are prevalent in the professional world. This course gives students a hands-on understanding of ML via popular ML architectures and an ethical framework for the use of data and artificial intelligence (AI). It is based on the main ideas of *Deep Learning Practicum* at MIT, taught by Dr. Natalie Lao and Dr. Hal Abelson. Their work demonstrated that ML can be made accessible for non-experts who use these tools for self guided discovery [9], which we have since adapted for the audience of liberal arts students.

A multitude of resources to teach and understand ML concepts and applications exist, but *these resources are targeted towards an audience with computer science expertise.* This generates a disconnect as two general populations form: one that understands the implications and concepts of the information in their field, and another that comprehends the ML applications that can be used to process that data. This hinders ML development and creates an unnecessary division between computer science and other fields of study.

Integrating computing into liberal arts pedagogy can result in extensive interdisciplinary connections which benefit the field by inviting more students who are not traditionally involved. Inequitable participation stagnates innovation by arbitrarily limiting contribution to the field [5]. We are introducing students to ML through its systematic impacts rather than just the technical implementation. Barretto found that cultural implications such as these would serve as a motivator for underrepresented students to enroll in an AI/ML course [2]. Our course aims to introduce these underrepresented groups into the world of AI/ML by meeting students where they are technically and culturally knowledgeable.

This paper also presents a case study of some of the adaptations for liberal arts students and environment. As we hope this contribution to the field will help others build similar courses for their institutions, we have published all of our course materials at `https://github.com/Emmanuel-Practical-Machine-Learning/index`.

2 Background

Literature on ML courses for non-majors is slim and mainly focuses on technical institutions; however, there has been a growing interest in recent years as students in all disciplines have begun to recognize the importance of ML

literacy [3]. As a result, current literature such as Sulmont et al. has noted the dramatic increase of demand for a more versatile approach to ML.

Studies have revealed multiple learning barriers which hinder non-majors' ability to learn ML. Common examples of these obstacles include the students' lack of perception on what ML actually entails, math anxiety, and limited programming experience [14]. Several projects have worked to lower these barriers, such as Bryan Loh and Tom White's SpaceSheet [12] and Google's Teachable Machine [15]. These resources can help students visualize theoretical concepts, algorithms, and model structures, which has been shown to help reduce obstacles related to math anxiety; Sulmont et al. found that students performed better when interacting with algorithms they could visualize, such as decision trees or k-nearest neighbors [14]. Resources that eliminate barriers for students enable them to construct their own model without having to start from scratch, allowing for a lower barrier to entry [7]. Visual tools and having limited prerequisites can be specifically helpful in a liberal art environment and have been shown to motivate and maintain student enrollment throughout the program. Additionally, assignments based on real world issues engaged students during class, allowing them to make meaningful connections between the course and their area of study [16].

Final projects can be an effective affordance for majors and non-majors alike. In a CS1 course, students reported greater confidence in their skills to build complex systems after working on a large final project [6]. In another course, the independence that students experience in choosing a unique final project allowed them to relate the curriculum to their disciplines and lives [4].

Allen et al. interviewed 10 professors on their experience with teaching ML to discover what pedagogy has been successful. They found that the most common responses included using real-world applications and group work which made students engage with the material, as well as foster a better class environment. Teaching about neural networks was identified as a *threshold concept* for students, meaning neural networks introduced an accessible way of thinking that changed student's perspective of ML. They hypothesized that neural networks can help engage students in class since they are comparable to the human thought processes, something students are somewhat familiar with and can visualize [1]. Findings also showed that area of study was not a determining factor in a student's success throughout the course; rather, the survey responses suggested that motivation and ability to dedicate time to course material determined the students academic success [8].

3 Presenting: Practical Machine Learning in a Liberal Arts Environment

Like traditional machine learning courses, In *Practical Machine Learning* at Emmanuel College, students are taught the fundamentals of AI, ML, and deep learning. The course aims to teach students how to analyze a problem, choose the proper artificial intelligence solution, and apply the solution to the problem. Using pre-lecture readings, in-class discussion, lectures, and assignments, students gain hands-on access to ML tools. Each reading, discussion, and assignment is grouped into modules, divided by the eight topics covered in the course: k-nearest neighbors and transfer learning, multilayer networks, bias and attacks, convolutional neural networks, generative models, recurrent neural networks, generative adversarial networks, and reinforcement learning. Each topic builds off the previous module, giving students a connected basis of knowledge. Typically, the course spends one to two weeks on each topic, with readings assigned before the lectures. The course materials are available at `https://github.com/Emmanuel-Practical-Machine-Learning/index`.

Differing from traditional ML classes, Emmanuel's practical machine learning curriculum adopted Lao's curriculum from MIT, *Deep Learning Practicum* [10]. Lao's work was chosen with the understanding that she designed it specifically for "non-majors." That audience at MIT, however, had more background in calculus and programming than a typical student at a liberal arts college, though Lao's initial work did provide a cornerstone of accessibility for our team to build upon. The most significant difference was the application of information. In Lao's course, the students were expected to use the information and eventually apply it to a final project of their own. These final projects mirrored many of the resources that were used in the class, such as text recognizer, lyric creator, or image classifier. As later expanded upon, the modified version made final projects that applied the concepts to everyday life. Changing these requirements encapsulated the difference in objective between the original and modified courses: MIT primarily drew students who would use Machine Learning as an essential part of their field, while non-majors should be understanding the systems to then apply to their own fields in the appropriate capacity.

The day-to-day lecture structure is flexible to accommodate questions, ideas, or clarifications, and emphasizes collaboration among students. Lectures are rarely scheduled to use the entire period, and topics may "float" between specific dates depending on level of discussion and questions generated from students. The intention to build a collaborative environment in both lectures and assignments encourages a "community of practice" [11]. This is also reflective of professional practice, as engineers often work collaboratively to develop new ideas or find solutions to recurring problems.

The open-discussion format also allows students to express concerns or opin-

ions about commonly contested points of ML, such as inherent bias, ethical uses of certain products, or transfer learning artwork. Demo assignments are completed throughout the week of lectures and readings. The observation and write-up portions that follow rarely have a right or wrong answer; rather, they aim to stimulate thought for the student and provide a baseline to interpret, observe, and hypothesize about the demo that is being used.

In addition to mirroring a computer science community, this format also supports the liberal arts mission of Emmanuel College; social justice agendas are rarely individual work, and rather than forcing the agenda of Emmanuel to fit a stereotypical machine learning format, our model mimics the style of non-major courses while teaching machine learning content.

From a technical standpoint, a major obstacle for machine learning at a liberal arts institution is access to resources and tools. After experimenting with Lao's work in the Emmanuel setting, it became clear that a significant difference is the access to high-performance computing, additional teaching support such as teaching assistants, and math and coding prerequisites to ensure a baseline of knowledge. While Emmanuel could certainly require these prerequisites, it would be counter-intuitive to force non-majors onto a math-based academic track for the sake of the minor. Thus, we had to make adjustments to assignments to ensure that they were accessible from a laptop, and students could successfully predict and create outcomes that they understood.

A majority of the tools are web-based and visual in nature. This allows students to visualize mathematically complex ML concepts, which has been shown to improve student understanding by mitigating math anxiety [14], while still interacting with real TensorFlow models. The course introduces a new single-purpose tool every two weeks to demonstrate a specific ML topic. This gives students the opportunity to interact with actual ML models, and develop understanding and intuition with their properties, without needing the programming skills to build these models from scratch.

At this point, we have run *Practical Machine Learning* for two spring semesters, and it will continue to run annually. The first year had 10 students and ran in the spring semester of 2021. This was during the COVID-19 quarantine, so the class was held fully remotely, and students had to depend on their personal computers. This was hugely limiting, especially when we discovered that both Zoom video services and ML training exercises relied on GPU. While this is not an issue with the curriculum, it did highlight the need for performant lab computers to overcome technical barriers such as slow processing times. The second section of the course ran in the spring semester of 2022 with 10 students. This time, the course was conducted in person and students had access to a high-performance campus computer lab. The second year also had the benefit of a peer tutor who was a graduate of the first cohort.

4 Ethics

Practical Machine Learning was designed with the intention of exploring current ethical issues with ML, and reinforcing those issues throughout the technical aspects of the course. We introduce the ethics discussion in the third unit, titled "Bias and Attacks." The center of this unit was an in-class discussion where students were broken into groups of four to five, and each person was given a different reading. In small groups, they discuss what each person read to formulate questions and thoughts in preparation for a class wide discussion. The readings focus on examples and expert analysis of systematic bias in ML systems, including recommendations on how to identify and diminish potential bias. The discussions are structured by prompt questions provided by the instructor, and the whole class collaboratively writes a "handbook" document on bias.

The readings in this unit also include ML-specific attack vectors, such as the 3D-printed turtle that is universally recognized by image classifier systems as a rifle [13]. These examples, with the examples on bias, seed a robust conversation on the trustworthiness of ML systems and the importance of fully understanding their weaknesses when choosing to apply them to situations that may seriously impact people.

Ethical considerations remain a part of the course from this module forward. The readings and discussion provide the dialect to easily discuss bias, vulnerability, and responsibility on the subsequent examples seen in the course. It is most strongly reinforced in the final project. In it, students needed to include a well-reasoned ethical analysis with their system design, including attention to how they would source their data, how the system would be trained, and how the system's inferences would be applied to the real world.

5 Outcomes

5.1 Final Projects

The main learning objective of Practical Machine Learning was to create a course that applied liberal arts education goals to the study of machine learning. This objective was epitomized by the final project–an assignment that allowed students to apply machine learning to any kind of problem or societal issue. This project aimed to proactively engage students with the material and promote application to the students' field of study or their general interests. There were very few constraints on what could be done, and it focused on the hypothetical application, rather than proof-of-concept demonstration. Since many students are non-majors, the final project provided space for students to

explore the real-life applications of artificial intelligence in their fields, rather than coding for the sake of coding.

While there was a wide range of topics, the underlying theme among all was that it paired the student's interest with what they had learned, producing a hypothetical that could be legitimately applied. For example, one project applied GANs to improve MRI diagnostics, while another used a convolutional neural network to teach cursive writing. Radiology and education are two vastly different fields, yet both students were able to find a common link between their interests and machine learning. An important requirement of the final project was to assess possible threats of bias and ethical concerns in the projects. The bias in an MRI diagnosis versus bias in teaching cursive are incomparable. MRI machine learning would have implications surrounding patient information and confidentiality, selection bias, and sample size, which the students highlighted in their project presentation. In contrast, cursive education using CNNs has fewer threats of bias but ethical concerns such as the amount of screen usage for children and the standard of education.

The students' ability to execute a successful final project was dependent on their ability to use their accumulated of knowledge from the course. During the course, students used a variety of machine learning tools. While the final project did not expect them to use these tools, application of the underlying ideas in a novel context was a feasible way to assess understanding. The final project was intentionally designed to elicit students to apply the concepts they had learned through these explorational tools. The final projects highlighted their competency in the material via a showcase of their transfer of knowledge.

5.2 Post-Survey

We administered a survey to all graduates of the course to gauge their overall experience, their current comfort with ML, and their feelings of belonging in the ML community of practice. The survey protocol was granted an exemption by our local IRB. We emailed 18 students with the solicitation and we received responses from five students. We attribute the low response rate to the timing of the survey (summertime) and a limited window of response time. Three of the authors of this paper are graduates of PML, however none participated in the survey.

All respondents reported having either minimal experience with programming and machine learning or previous knowledge from other courses within the Data Analytics minor. When asked about their motivations to enroll in the course, three students noted the Data Analytics minor, while one mentioned a positive relationship with the professor, and the last respondent was encouraged by a family member. Furthermore, when answering this question four respondents recognized the importance of ML and data literacy in their

career. When questioned about their experience with the course, four students described a positive experience in *Practical Machine Learning*, and one student chose not to respond. Of the four who responded, three said they were moderately to highly comfortable with the course material, while one said they had low comfort.

When prompted to explain how their perception of ML changed once taking the course, four respondents described having a new-found appreciation of ML. One response specifically noted being able to detect ML models and its impact in their everyday life and two students noted having a new perspective on their discipline, as well as an eagerness to continue learning about ML. This echoes findings from others [4, 6], which is encouraging.

Lastly, students were asked if they felt a sense of belonging in a machine learning community of practice. The survey, by design, did not define "community of practice," leaving the students to find their own meaning. The responses identified two communities. The first, and more interesting, was a local community, identified by one student: researchers and students at Emmanuel College who are involved with understanding and applying artificial intelligence to their work and education. The second was the global community, comprised of experts and enthusiasts connected by the internet. We acknowledge that students are less likely to become part of the professional global community, but two respondents felt that they could engage with the global ML community in a meaningful way. One student said they did not know what an ML community entails and felt disconnected from it due to imposter syndrome.

6 Future Work

While the number of respondents is limited and anecdotal, our post-survey results suggest that our affordances for accessibility do not hinder a student's ability to grasp the concepts of ML. This reflects trends throughout existing literature which have found that working with theoretical concepts and limited code can help students engage with the material in a unique and meaningful way [6]. Similar to Guerzhoy [7] and Sulmont [14], our respondents also found visual representations of concepts helpful to their understanding by focusing their time and attention on course goals rather than programming skills which are outside of the scope of the course. Moving forward, we plan on conducting a pre and post-course survey to better assess course effectiveness. The pre-survey will include questions about popular misconceptions towards AI/ML. We also plan to add to our existing post-survey by asking students if they could envision themselves applying the concepts learned to their discipline or major. This will be an attempt to better meet students where they are, letting us make the course more accessible to our demographic of students.

7 Conclusion

We developed an accessible course in ML theory and application for liberal arts students with minimal background in computing. We ran this course twice so far, and in doing so, discovered many barriers and adaptations to help overcome them. This work is ongoing, but we hope that our design experience, teaching experience, and student outcomes will aid other institutions to develop their own practical ML courses. Our current course materials are available as a starting point at `https://github.com/Emmanuel-Practical-Machine-Learning/index`.

There were significant technical barriers in the first years of this course that were independent of the learning outcomes. This was partly a result of the nascent, fast-changing state of ML technology. Most of the technical barriers were avoided by finding accessible tools that demonstrated, exercised, and provided student experience with industry-standard ML systems. These tools were not general, and each module had completely different tools that expounded a specific ML topic. Having different tools every two weeks may seem initially to be itself a barrier, but the students were unfazed by this. We believe the topic order and focus on underlying concepts helped students pick up new tools easily, as the engine behind the particular tool was the focus. This is also reflective of the overall goal of the course: to equip students to go out into the world with the understanding of the workings and nature of ML such that they can apply that understanding to whatever toolsets are available.

8 Acknowledgements

The authors thank Michael Barron, Drew Kotarski, the Emmanuel College Faculty Development Committee, and all of the *Practical Machine Learning* students.

References

[1] Becky Allen, Andrew Stephen McGough, and Marie Devlin. "Toward a Framework for Teaching Artificial Intelligence to a Higher Education Audience". In: *ACM Trans. Comput. Educ.* 22.2 (Nov. 2021). DOI: 10.1145/3485062. URL: https://doi.org/10.1145/3485062.

[2] Daphne Barretto et al. “Exploring Why Underrepresented Students Are Less Likely to Study Machine Learning and Artificial Intelligence”. In: *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE ’21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 457–463. ISBN: 9781450382144. DOI: 10.1145/3430665.3456332. URL: https://doi.org/10.1145/3430665.3456332.

[3] Clare Bates Congdon. “Machine Learning in the Liberal Arts Curriculum”. In: *SIGCSE Bull.* 32.1 (Mar. 2000), pp. 100–104. ISSN: 0097-8418. DOI: 10.1145/331795.331824. URL: https://doi.org/10.1145/331795.331824.

[4] Jessica Q. Dawson et al. “Designing an Introductory Programming Course to Improve Non-Majors’ Experiences”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE ’18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 26–31. ISBN: 9781450351034. DOI: 10.1145/3159450.3159548. URL: https://doi.org/10.1145/3159450.3159548.

[5] Wendy M. DuBow et al. “Efforts to Make Computer Science More Inclusive of Women”. In: *ACM Inroads* 7.4 (Nov. 2016), pp. 74–80. ISSN: 2153-2184. DOI: 10.1145/2998500. URL: https://doi.org/10.1145/2998500.

[6] Adrian A. de Freitas and Troy B. Weingart. “I’m Going to Learn What?!? Teaching Artificial Intelligence to Freshmen in an Introductory Computer Science Course”. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 198–204. ISBN: 9781450380621. DOI: 10.1145/3408877.3432530. URL: https://doi.org/10.1145/3408877.3432530.

[7] Michael Guerzhoy. “AI Education Matters: Teaching with Deep Learning Frameworks in Introductory Machine Learning Courses”. In: *AI Matters* 4.3 (Oct. 2018), pp. 14–15. DOI: 10.1145/3284751.3284756. URL: https://doi.org/10.1145/3284751.3284756.

[8] Boris Kerkez. “Robotics and Machine Learning in a Core College Curriculum”. In: *J. Comput. Sci. Coll.* 24.1 (Oct. 2008), pp. 103–109. ISSN: 1937-4771.

[9] N. Lao, I. Lee, and H. Abelson. “A Deep Learning Practicum: Concepts and Practices for Teaching Actionable Machine Learning at the Tertiary Education Level”. In: *ICERI2019 Proceedings*. 12th annual International Conference of Education, Research and Innovation. Seville,

Spain: IATED, Nov. 2019, pp. 405–415. ISBN: 978-84-09-14755-7. DOI: 10.21125/iceri.2019.0137. URL: https://dx.doi.org/10.21125/iceri.2019.0137.

[10] Natalie Lao. "Reorienting machine learning education towards tinkerers and ML-engaged citizens". PhD thesis. Massachusetts Institute of Technology, 2020.

[11] Jean Lave and Etienne Wenger. *Situated Learning: Legitimate Peripheral Participation*. Learning in Doing: Social, Cognitive and Computational Perspectives. Cambridge University Press, 1991. DOI: 10.1017/CBO9780511815355.

[12] Bryan Loh and Tom White. "SpaceSheets: Interactive Latent Space Exploration through a Spreadsheet Interface". In: (June 2020). DOI: 10.26686/wgtn.12585305.v1. URL: https://openaccess.wgtn.ac.nz/articles/journal_contribution/SpaceSheets_Interactive_Latent_Space_Exploration_through_a_Spreadsheet_Interface/12585305.

[13] Bruce Schneier. "Attacking machine learning systems". In: *Computer* 53.5 (2020), pp. 78–80.

[14] Elisabeth Sulmont, Elizabeth Patitsas, and Jeremy R. Cooperstock. "What Is Hard about Teaching Machine Learning to Non-Majors? Insights from Classifying Instructors' Learning Goals". In: *ACM Trans. Comput. Educ.* 19.4 (July 2019). DOI: 10.1145/3336124. URL: https://doi.org/10.1145/3336124.

[15] Nikhil Thorat. *How to build a Teachable Machine with TensorFlow.js*. Observable, June 2018. URL: https://observablehq.com/@nsthorat/how-to-build-a-teachable-machine-with-tensorflow-js (visited on 08/17/2022).

[16] Timothy Urness and Eric Manley. "Building a Thriving CS Program at a Small Liberal Arts College". In: *J. Comput. Sci. Coll.* 26.5 (May 2011), pp. 268–274. ISSN: 1937-4771.

[17] Henry M. Walker and Charles Kelemen. "Computer Science and the Liberal Arts: A Philosophical Examination". In: *ACM Trans. Comput. Educ.* 10.1 (Mar. 2010). DOI: 10.1145/1731041.1731043. URL: https://doi.org/10.1145/1731041.1731043.

Functional Programming, in the Data Structures Course, in Java*

John MacCormick
Dickinson College
Carlisle, PA
jmac@dickinson.edu

Abstract

We describe a new curriculum design, termed *functional embedding*, for embedding functional programming within a core sequence of courses taught in a non-functional programming language such as Java. We present evidence that functional embedding has been successful in practice, based on a survey of student perceptions and analysis of student exam performance. An analysis of college computer science curriculums demonstrates that at least 59% of colleges can benefit from the approach.

1 Introduction

There is wide agreement among computer science educators that it is important for students to acquire proficiency in a variety of programming paradigms. Of these paradigms, perhaps the most important are the object-oriented and functional paradigms. Modern computer science curriculums rarely have any difficulty in ensuring that all students are exposed to the object-oriented paradigm, but the same is not true of the functional paradigm. One relatively common curriculum design has a core sequence of three or four courses containing no functional programming. This core sequence of exclusively imperative/object-oriented content may comprise, for example, CS1→CS2→DSA (where DSA is a single course covering data structures and algorithms), or CS0→CS1→Data

Structures→Algorithms (where CS0 is a course skipped by students with prior programming experience). This curriculum design typically includes a separate Programming Languages (PL) course which contains a substantial functional programming component (say, a minimum of 30% functional—but we observe up to 100% functional programming in some PL courses). If the PL course is required for the computer science major, all CS majors are thus guaranteed reasonable exposure to functional programming.

It is not uncommon, however, for the PL course to be an elective. If the core sequence is exclusively imperative/object-oriented, this raises an important curricular problem: students who do not take the elective PL course can graduate with no awareness of the functional paradigm. The CS education community is divided on whether this should be considered a severe problem; section 5 gives more details on why it may be considered problematic that an undergraduate computer scientist can graduate with no knowledge of functional programming. In this paper, we propose a partial solution to this problem and report on our experiences with it.

The proposed solution is to insert a suitable amount of functional programming into the core sequence, without changing the (imperative, object-oriented) programming languages in which the core sequence is taught. Our department has experimented successfully with this approach since 2019, by making alterations to our CS2 data structures course, which is taught in Java. By reallocating material between our separate data structures and algorithms courses, we made room for a one-week unit of functional programming material. We do not introduce the overhead of studying a truly functional programming language. Instead, we use Java lambda expressions and other features in `Java.lang.function`, combined with practical applications for efficient parallel data processing via Java's Stream API.

This explains the title of this paper: we teach some aspects of functional programming, in the data structures course, in Java. The remainder of the paper will describe the implementation details and results of that approach. Nevertheless, we believe it would generalize to other situations. For example, it is certainly possible to create similar units of functional programming in courses that use Python or C++. And these units can easily be inserted into courses other than Data Structures, including CS1 or Algorithms. Indeed, one of our items of future work is to weave functional programming into most other required courses, now that we can be sure students have seen it before.

2 Related work and a taxonomy of functional programming curriculums

In this section, we briefly survey other approaches to incorporating functional programming within the computer science curriculum, and we propose a taxonomy of five possible approaches: functional-first, functional-required, functional-elective, functional-minimal, and functional-embedded.

Many previous researchers have investigated the use of functional programming within the core sequence of programming courses in a computer science major. One highly successful and much-studied approach is known as *functional-first* [7]: CS1, the first core programming course, is taught either entirely or largely in the functional paradigm. Famous examples of such curriculums include those of Grinnell College [1] and Brown University [4], amplified by the influential textbook *How to Design Programs* [3]. The functional-first approach has numerous well-documented advantages, but it also presents challenges. Empirically, we find that it has not been widely adopted. For example, in our sample of CCSC curriculums described later (section 4), we found that zero out of 29 institutions had adopted functional-first.

As described in the introduction, a common curricular approach is to include a substantial topic on functional programming within a programming languages (PL) course. If the PL course is required for the major, we refer to this as the *functional-required* approach; if the PL course is an elective, we describe the approach as *functional-elective*.

There are some computer science curriculums in which functional programming does not appear to be an explicit goal. Either it is not present at all, or it makes an appearance as a byproduct of certain elective courses not focussed on programming languages (such as an AI course that makes use of LISP). We refer to this curricular approach as *functional-minimal*.

In this paper, we advocate a new curricular approach to functional programming: a modest but nontrivial amount of functional programming is embedded into one or more of the courses in the core sequence, such as CS2/data structures. Moreover, embedded functional programming content is emphasized as an important topic that will be reflected in other parts of the curriculum. We refer to this approach as *functional-embedded*.

3 Method for embedding functional programming in a Java-based data structures course

In this section we give details of our current approach to embedding functional programming in a Java-based data structures course. Section 3.1 describes a low-overhead way of introducing functions as first-class objects, and section 3.2

describes a practical application for cementing student engagement via the Java Stream API. We provide these details for concreteness only; we believe many other approaches could achieve the same goals.

3.1 Rudiments of functional programming in Java

It can be argued that the most important concepts underlying functional programming are *immutability* (lack of side effects) and the treatment of functions as *first-class objects*. The brief survey of functional programming described here focuses on functions as first-class objects, mentioning immutability only as it applies to existing Java classes such as `String` and `Stream`. In practice, students learn that functions can be: (i) parameters of other functions; (ii) return values of other functions; and (iii) created and employed as local variables. Simple demos of these three features can easily be achieved using Java's functional programming package, `java.util.function`, which has been available since Java version 8 was released in 2014. For example, we can define the function $f(x) = 3x^2 + 5$ and then evaluate $f(2)$ via the following snippet of Java:

```
Function<Integer, Integer> f = x->3*x*x+5;
System.out.println(f.apply(2)); // prints "17"
```

There is one piece of syntactic ugliness that is unavoidable in the above snippet: to evaluate $f(2)$ we need to invoke the `apply()` method on the object `f`. Thus, we write `f.apply(2)` rather than using the more natural notation `f(2)`. This reflects the fact that Java is not a functional language: one cannot define a Java object that is a function. Instead, we create an object that implements a *functional interface*. The interface includes the `apply()` method, and that is how the function must be invoked.

To avoid this potential source of confusion, we prefer to first introduce the notion of functions as first-class objects using Python. We have found this works well, even in a course that is otherwise 100% Java and does not assume any prior knowledge of other programming languages. The syntax of Python is close enough to standard pseudocode conventions that students with no knowledge of Python can leap right in. In our experience, they can actively participate in an in-class, browser-based mini-lab on the topic of functions as objects after only a few minutes' explanation by the instructor. For example, the above Java snippet becomes

```
def f(x): return 3*x*x+5
print( f(2) ) # prints "17"
```

The above snippet can be formulated using a lambda expression in Python, and in our approach students will certainly learn how to do that. But we prefer

to delay the introduction of lambda expressions until after the idea of functions as objects has already been demonstrated.

For example, the concept of passing a function as a parameter is always a challenging new abstraction the first time it is seen by a student. But a hands-on mini-lab, using snippets such as the following, can help students quickly adapt to this new idea:

```
def isIncreasingOn123(f): return f(1)<f(2) and f(2)<f(3)
def add5(x): return x+5
print( isIncreasingOn123(add5) ) # prints "True"
def applyTo9(f): return f(9)
print( applyTo9(add5) ) # prints "14"
```

Once the idea of treating functions like any other data item is familiar, we can switch back to Java. We recommend omitting the details of functional interfaces, instead treating the `apply()` method as a required piece of Java syntax that is not explained further. Thus, the above Python snippets become:

```
public static boolean isIncreasingOn123(Function<Integer, Integer> f) {
       return f.apply(1) < f.apply(2) && f.apply(2) < f.apply(3);
}

public static int applyTo9(Function<Integer, Integer> f) {
       return f.apply(9);
}

public static void main(String[ ] args) {
       Function<Integer, Integer> add5 = x->x+5;
       System.out.println( isIncreasingOn123(add5) ); // prints "true"
       System.out.println( applyTo9(add5) ); // prints "14"
}
```

Some further fundamentals of functional programming, including moderately advanced lambda expressions, can be introduced in a similar fashion. Further details are available in our publicly-available textbook chapter [9].

3.2 A practical application of functional programming: parallel processing of data streams

It is a well-known educational principle, not just within computer science, that learning outcomes for a theoretical concept are improved if students perceive the concept as applicable or useful [5]. We believe, therefore, that it is important for students to apply their knowledge of functional programming in Java to some real-world situations. For this, we use Java's Stream API; this

is provided in `java.util.stream` and was introduced by Java version 8, at the same time as the functional programming facilities, in 2014.

In Java, `Stream<T>` is an interface for performing operations on sequences of objects of type `T`. There are two types of operations on streams: intermediate operations and terminal operations. In the brief coverage provided in our data structures course, we explore only eight of these operations: the four intermediate operations `filter()`, `map()`, `sequential()`, and `parallel()`; and the four terminal operations `count()`, `foreach()`, `reduce()`, and `sum()`. Any computation based on a stream performs a sequence of intermediate operations followed by one terminal operation. Because many of these operations accept lambda expressions as parameters, they are an excellent way to practice the application of functional programming. For example, the following snippet counts the number of words in a stream that begin with '`c`' and end with '`t`', by using two `filter()` operations followed by the `count()` operation; it employs two lambda expressions for the filters:

```
Stream<String> s = Stream.of("cat", "bat", "catch", "chat");
long numWords = s.filter(word -> word.startsWith("c"))
                 .filter(word -> word.endsWith("t"))
                 .count(); // returns 2
```

The Stream API provides opportunities for students to implement realistic experiments from the world of big data, based on only modest guidance from the instructor—certainly less than one class meeting, in our experience. For example, students can apply the map-reduce [2] framework to large data sets, and/or demonstrate the speed-up from using parallel versus sequential streams. This latter experiment is trivial to implement: one simply prepends the call "`.parallel()`" to the sequence of stream operations. Further details are available in our publicly-available textbook chapter [9].

4 Results

In this section we present results from three investigations which suggest that the functional-embedded approach (teaching functional programming in the data structures course and/or elsewhere in the core) is efficacious and suitable for incorporation in a substantial proportion of existing computer science curriculums.

Investigation 1: Student performance. We analyzed the final exam scores of students in the fall 2021 instance of the data structures course. A total of 25 students completed the course, and all are included in this analysis. The final exam included a three-part question requiring students to write code using the Java Stream API and employing lambda expressions. Students averaged

28/30 points on this question. On the most challenging part of the question, which required a non-trivial application of the map-reduce pattern, 22 of the 25 students (88%) scored 13/15 or higher. This demonstrates that a key application of functional programming was mastered by a strong majority of students.

Investigation 2: Student perceptions. We surveyed students who completed our functional-embedded data structures course, assessing their perception of the 9 topics in the course; 32 students participated in the survey. They rated each topic based on how "interesting" and (separately) "useful" they perceived it to be. Ratings employed a 5-point Likert scale, from 1 = "not at all interesting" to 5 = "extremely interesting" (and similarly for "useful"). Results are shown in figure 1. Functional programming was certainly perceived as interesting and useful: it averaged 3.7 and 3.5 respectively. That is, the average response places functional programming between "moderately interesting" and "very interesting" (and similarly for "useful"). The level of interest in functional programming was commensurate with most other topics; only the binary search tree/heaps topic received a statistically significant higher score. However, functional programming was ranked ninth out of the nine topics in terms of usefulness, substantially lower than classic data structures topics such as lists/stacks/queues (average 4.4) and graphs (4.3); these two differences have high statistical significance (t-test p-value $< 10^{-3}$). The low ranking of functional programming in terms of perceived usefulness may indicate that we could do a better job of demonstrating the importance of functional programming. Or maybe the student perception is correct: perhaps fundamentals such as stacks, queues, and graphs really are more useful than functional programming.

Investigation 3: Curriculum survey. We analyzed the curriculums for the computer science major of the 32 colleges and universities whose faculty contributed to the four most recent CCSC conference proceedings [8]. Three institutions that do not offer a four-year computer science major were excluded. For the remaining 29 institutions, we examined publicly-available requirements for the major and course descriptions to classify each institution's curriculum according to the taxonomy described in section 2. Figure 2 shows the results.

Notwithstanding the well-known examples of functional-first curriculums mentioned earlier, we find that, in this sample, none of these institutions employs the functional-first approach. Unsurprisingly, no institutions employ the functional-embedded approach either: this is the new approach used at our own institution and which we are advocating in the present paper.

The remaining three categories demonstrate that there is significant diversity in how these institutions cover functional programming: 79% include it in a programming languages (PL) course, split almost equally between the

topic	average Likert score for... "interesting"	"useful"
Binary search trees/heaps	4.2*	4.3**
Sorting algorithms	4.1	4.4**
Hash tables	4.0	3.9*
Graphs	3.9	4.3**
Lists/stacks/queues	3.8	4.4**
Functional programming	**3.7**	**3.5**
Recursion	3.7	4.1**
Algorithm Analysis	3.5	4.2**
Generics	3.1**	3.6

Figure 1: **Results of student perception survey.** Topics are sorted by the average rating for "interesting." Starred values indicate a statistically significant p-value for a two-tailed paired t-test comparing the given result with the result for functional programming: single * for $p < 0.05$; double ** for $p < 0.01$.

Approach	frequency	percentage
functional-first	0	0%
functional-embedded (proposed here)	0	0%
functional-required	12	41%
functional-elective	11	38%
functional-minimal	6	21%

Figure 2: Results of the functional programming curriculum survey.

functional-required and functional-elective approaches. That is, about half of the 79% with a PL course (41%) require the PL course for the major, and the other half (another 38%) offer PL as an elective. Finally, 21% of the institutions employ the functional-minimal approach, meaning they do not offer a significant amount of functional programming.

We believe these results demonstrate ample opportunity for the functional-embedded approach advocated here. Certainly, the 59% of institutions where CS majors can graduate with no exposure to functional programming could rectify this using functional-embedding. In addition, we believe that the 41% in the functional-required category would also benefit from embedding a certain amount of functional programming into one or more core courses such as CS2. This has the benefit of allowing the functional mindset to start earlier and resonate in other parts of the curriculum; perhaps students will view it as a more

fundamental and integral technique, rather than an approach of mainly theoretical interest that has been sidelined to a challenging upper-level course. To summarize, 100% of the institutions in this sample may benefit from adopting the functional-embedded approach, and 59% would eliminate the possibility of students graduating with no exposure to functional programming.

5 Discussion

This paper makes the assumption that it is desirable for undergraduate computer scientists to be exposed to functional programming. It is beyond our scope to justify this claim, which has been the subject of debate for decades. Functional ideas have become increasingly important even within primarily imperative/object-oriented languages, and this is one argument for teaching functional programming explicitly. Another argument is that the ACM/IEEE 2013 Curriculum Guidelines ([7], page 156) require 7 core lecture hours of functional programming, equivalent to 2–3 weeks of classes. The more recent 2020 guidelines, known as CC2020 [6], express required content in terms of competencies rather than lecture hours. The guidelines list 84 competencies, of which 3 mention functional programming explicitly ([6], page 114; more details below). This can be converted very approximately to comprehensible units as follows. In a 10-course major, one of the courses would need to devote $3/84 \times 10 = 36\%$ of its time to covering the three functional programming competencies—equivalent to 5 weeks in a 14-week semester. This level of coverage is probably not a realistic goal for every CS program. But these curriculum guidelines imply that it is highly desirable for every computer science undergraduate to receive some exposure to functional programming ideas. As discussed in the previous section, our curriculum analysis shows that about 60% of institutions in our sample do not currently achieve this goal, and they could do so by using the functional-embedded approach.

One potential limitation of our functional-embedded approach is that we are not teaching the full range of functional thinking: our emphasis is on the practical use of lambda expressions. It lies beyond the scope of this paper to debate this in detail, but this is an interesting line of future research. If we are limited to a very small amount of time for functional programming, what are the most essential and useful ideas to convey? As a partial justification for the approach advocated here, we note that reasonable progress towards two of the three CC2020 functional programming competencies mentioned above can be achieved using only the material covered in our functional-embedded CS2 data structures course. In detail, the two competencies covered are PL-A (implement a function that takes and returns other functions) and PL-D (use operations on aggregates, including operations that take functions as arguments).

The competency not covered is PL-E (contrast the procedural/functional approach with the object-oriented approach).

Another potential limitation is that, at present, we devote only one week to functional programming in the data structures course. In fact, we have been satisfied with the level of conceptual coverage, but the one-week timeframe does seem short for a valuable topic. We do commit to functional programming as an important concept, listing it in the official bulletin description of the course and in one of the department-approved learning goals.

Hence, an area of future work is to thread some functional programming ideas into additional courses. At our institution, students also encounter lambda expressions in a software engineering course that relies on JavaScript web frameworks. We believe they could benefit from further exposure in the required algorithms course and in electives such as artificial intelligence. As a specific example of this, note that the `sorted()` function in the standard Python library accepts a functional parameter for extracting the key from each item. Hence, the instructor has an opportunity for a quick refresher on lambda expressions whenever sorting a list in Python. But an important issue for further investigation is, how can we embed deeper functional programming concepts, such as immutability?

6 Conclusion

We have described *functional embedding*, a new curriculum design that teaches a modest but meaningful amount of functional programming within a core course taught in a non-functional programming language. An analysis of CCSC college curriculums showed that about 60% of colleges in the sample do not currently ensure that CS majors have exposure to functional programming, and that they could achieve this with relative ease via functional embedding. We argued that the remaining 40% of colleges may also gain benefits from functional embedding. A survey of student perceptions shows that functional programming is perceived as interesting and useful, albeit to a lesser extent than classical topics in data structures. An analysis of student exam results demonstrates solid achievement on sophisticated learning goals such as the map-reduce paradigm, even when only one week of functional programming is embedded. We hope functional embedding will be adopted at other institutions and that future work can refine the embedding of functional programming throughout the typical computer science curriculum.

References

[1] Sarah Dahlby Albright, Titus H. Klinge, and Samuel A. Rebelsky. "A Functional Approach to Data Science in CS1". In: *Proc. SIGCSE*. 2018, pp. 1035–1040.

[2] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[3] M. Felleisen et al. *How to Design Programs: An Introduction to Programming and Computing*. 2nd edition. MIT Press, 2018.

[4] Robert Bruce Findler et al. "DrScheme: A programming environment for Scheme". In: *Journal of Functional Programming* 12.2 (2002), pp. 159–182.

[5] L. Dee Fink. *Creating significant learning experiences: An integrated approach to designing college courses*. 2nd edition. John Wiley & Sons, 2013.

[6] CC2020 Task Force. *Computing Curricula 2020: Paradigms for Global Computing Education*. Association for Computing Machinery, 2020.

[7] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, 2013.

[8] Baochuan Lu, ed. *Journal of Computing Sciences in Colleges*. Vol. 37. 6,7,8,10. Consortium for Computing Sciences in Colleges, 2022.

[9] John MacCormick. "Functional Programming and Streams". Available as https://arxiv.org/abs/2302.09403. 2023.

A Student-Oriented Simulator for the Arm64 Instruction Set*

Robert Montante
Mathematical and Digital Sciences
Bloomsburg University
bloomu.prof@gmail.com

Abstract

This paper describes an Arm64 CPU simulator for use in Computer Organization courses, and student exercises to use it. The simulator is for the Arm64 Instruction Set Architecture (also known as the AARCH64 ISA), and is written in C so that it can run under e.g. Windows on a student's laptop. It implements a sufficient subset of the Arm64 instructions to simulate execution of some simple Arm64 assembly programs that have been assembled on a Raspberry Pi 3 or 4/400 that is running the 64-bit RaspiOS operating system. The simulator is text based, can step through machine instructions singly or all together, and displays register contents as desired or after each instruction. Its behavior is similar to a debugger such as gdb (although this simulator is much simpler than gdb).

The simulator has been used in the author's Computer Organization course twice. Exercises for it involve running the example executables and observing the effect on registers, identifying unimplemented instructions and implementing them, and extending the simulator with additional instructions. Arm64 executables are included; they range from a minimal-NOP "program" to assembly-only implementations of the Fibonacci and Factorial series and a basic loop. Source code is available in a git repository via this https://github.com/bloomu-prof/Arm64-Simulator.git link.

The current version of the simulator implements the CPU's datapath only conceptually; advanced exercises (or a new version of the simulator) might implement a more concrete representation of the "fetch-execute cycle" or extend the simulator to the NEON instruction set.

1 Introduction

A course on computer organization focuses on computer hardware, which is uncomfortable for students who are more used to software issues and considerations. Compared to a programming course, there are limited opportunities for hands-on activities, and most exercises consist of written problems. While digital design topics offer numerous design and assembly activities, they can fill an entire semester with "engineering-like" circuit- and component-design material; higher-level topics such as pipelining, cache and memory organization, or multi-core/heterogeneous CPU designs receive less attention.

The author's computer science program emphasizes software engineering and most of the students are more comfortable with software issues than with hardware issues. In addition, most of their experience has been on x86-architecture systems (usually running the Microsoft Windows operating system). The curriculum includes a C programming course as a prerequisite to a computer organization course, both intended for sophomores who have been exposed only to Java previously. The computer organization course broadens their experience by examining architectures other than the x86 "CISC" (Complex Instruction Set Computer) approach. An example textbook for this course is "Computer Organization and Design: the Hardware/Software Interface" by Patterson and Hennessy [3], which covers "RISC" (Reduced Instruction Set Computer) architectures in detail. In particular, it develops a pipelined version of the datapath of a RISC processor such as the Arm or MIPS designs.

This paper describes a simulator for the Arm64 instruction set architecture (ISA). It reads and parses an executable file, displays the contained memory image, and simulates the execution of each machine instruction while displaying the effect on the processor's registers. As such it can illustrate the behavior of a processor's fetch-execute cycle. The simulator itself is written in C, and structured in such a way as to be readily modifiable or extensible by anyone who has had a course in C that covers pointers. The code is discussed in detail below.

Inputs for the simulator are in the form of executable files from a Raspberry Pi running 64-bit RaspiOS. These programs files include some instructions that are decoded but lack implementations of their execution. Adding these instructions to the simulator provides a hands-on programming exercise for students while encouraging them to work with machine instructions at the

instruction-format level. Student exercises start with tracing the execution of very simple programs and proceed to "debugging" them by determining what instructions are not simulated properly, and then extending the simulator's execution function to implement the missing instructions.

A comparable program, the "Graphical Micro-Architecture Simulator" [2], has been released by Arm Limited. It is closely coupled with the Patterson and Hennessy book, recognizes that book's LEGv8 subset, and provides a user interface that relates instruction execution to the single-cycle and pipelined datapaths developed in that book. It works with the LEGv8 assembly language rather than Arm64 machine instructions, and directly illustrates each instruction's effect on the processor's datapath and registers. It is not clear how easily the program can be extended to additional instructions beyond the LEGv8 subset.

2 Arm64 Executables and Exercises

The simulator's inputs are executable programs written for a Raspberry Pi running 64-bit RaspiOS. They are written in Arm64 assembly language; simple C programs could also be compiled and linked statically, but with less control over which machine instructions to use. The executables so far created use a fairly small subset of the Arm64 instruction set.

(It should be mentioned that the LEGv8 subset, as defined in the Patterson and Hennessy, uses a few instruction mnemonics that are inconsistent with the standard mnemonics for the full Arm64 instruction set. A real assembler that accepts the LEGv8 mnemonics should still produce machine instructions suited to the ARMv8 ISA, or to this simulator.)

2.1 Executables

The simplest executable is a "program" that consists of a handful of "NOP" instructions, two "MOVZ" instructions, and an "SVC" supervisor call. This "**nop**" program is sufficient to demonstrate the use of the simulator and serves as a "hello world" exercise for using the simulator. The supervisor call illustrates that some instructions are simulated by simply performing the requested action in non-architecture-dependent way; in this case the supervisor call is a request to terminate execution, and the Arm64 simulator simply stops trying to process instructions after performing a "sys_exit" supervisor call.

The "**demostr0**" executable actually does something – it uses "LDR" memory-access instructions and another "SVC" service to produce a text-string output. It also includes some "LDR" instructions that have no effect, but do demonstrate additional addressing modes. Stepping through this program shows how

instruction executions affect register contents; using the simulator's "Run" command runs the whole program and produces its output.

Another output program, "**simplestring**", uses some utility subroutines to produce output, thereby demonstrating the performance of some branch instructions. It also produces spurious "garbage" output because on an unimplemented instruction. Students must find the unimplemented instruction, which is trivial as the simulator reports instructions that it has decoded but cannot execute. They begin extending the simulator by adding code needed to implement the instruction. The missing instruction is a form of "SUBTRACT" instruction, and is implemented easily by copying and modifying the implementation of a different "SUBTRACT" instruction in the simulator's execution routine.

The "**hexsmall**" and "**hexbig**" programs show output of numeric values as hexadecimal strings. "**hexsmall**" correctly displays a 16- to 32- bit value that is hardcoded in. "**hexbig**" attempts to do the same for a 33- to 64-bit value, but requires the "MOVK" instruction to produce correct output. Implementing this instruction is another student exercise, and reinforces understanding of the relationship between the 32-bit registers and the 64-bit registers.

The "**dialog**" program prompts for a text-string input and echoes it back out. This requires implementing an additional system service, "sys_read". When this is implemented, the simulator begins to be capable of running meaningful programs with input and output.

"**writeint**" displays a numeric value in base 10, which is much more involved than displaying it in hexadecimal. This executable introduces the first "new" instruction that isn't just a variation on what's already implemented, namely "MADD" (or simply "MUL"). While the instruction requires significant additional circuitry in a real processor, it is simply another register-operations instruction from the simulator's perspective.

Finally, the "**factorial**", "**fibonacci**", and "**averageloop**" programs demonstrate actual computation. Although they are more complex as (recursive) assembly-language programs, they require no new instructions beyond those required for the simpler programs. Their correct operation provides a little bit of "gratification" at getting the simulator to work properly and do something interesting.

2.2 Exercises

Student exercises are based on the simulator. Building and verifying the simulator serves as a lab exercise. One involves identifying the missing instructions and adding them to the simulator. This exercise could be split into multiple exercises, perhaps with additional exploration of the assembly-language code if desired.

The first two times this simulator was used in a course, the source code was provided in the form of web pages that could be copied and pasted into source files, while presumably being studied. This was done in two steps: the first is build to a program that can read and parse an ELF-formatted executable. A successful program will display a report of the ELF-file section contents, and place the machine instructions into a memory array, and the exercise writeup asks questions about the output. The second step is to add a fetch-decode-execute cycle that processes the memory array of instructions. The first two executables, "**nop**" and "**demostr0**", will run successfully and serve to prove that the simulator is correctly built. These two steps have been treated as separate exercises. The Discussion section below addresses this.

The next exercise is to find and implement the missing instructions needed by the other executables. Students in previous years have had prior exposure to Arm86 assembly language, and so the executables themselves are not examined in any detail. Going forward, more time should be spent on the assembly language for students who have not seen it before.

3 The Simulator

The simulator is organized in two stages, the ELF-file loader stage and the fetch-decode-execute stage.

Linux executables are kept on disk as ELF files. The "gcc" compiler and the "as" assembler both produce ELF-formatted files on Arm64 systems as well as on x86-64 systems. ELF ("Executable and Linkable Format") is a format specification for storing executables and other binary information. It starts with section headers that describe the file's sections and their contents. Machine instructions, static data, and blank memory space ("bss" segments, not stored as such in the file) each get their own section, along with sections to support dynamic linking information and debugging data. The machine instruction sections are of course different for each ISA, but the format's headers are the same.

The loader stage produces a report on the executable program's memory usage as determined from the executable file, and fills a memory array with the binary contents (machine instructions and data). The memory array is sized to include space for the heap and stack, as specified by the ELF format. Optionally the whole memory array can be written as a simple binary data file.

The second stage of the lab exercise adds a "fetch_decode_execute()" function that runs the executables instruction-by-instruction. This function actually creates a "read-evaluate-print loop" that processes single-letter commands from the user's keyboard; commands execute the next instruction in the executable, display information about the processor's registers or the program

memory, and control the simulator itself. If the user commands to run one or more instructions, the loop first "fetches" four bytes from the memory array, at an index calculated from the Program Counter's virtual memory address. (This index calculation is internal to the memory subsystem, only the program's virtual addresses are used within the processor.)

"Decoding" is done by a helper function which creates a struct representing the values of the bit fields in the fetched machine instruction. It also contains the instruction's corresponding mnemonic, as found from a table of instruction mnemonics and bit patterns [4]. This mnemonic can be helpful for "disassembling" the instructions in the executable, and is used for convenience in the "execution" of the instruction. The instruction table is fairly complete although the Arm architecture's "NEON" set of SIMD[1] instructions is not included. Any missing instructions can be added straightforwardly by inspecting the table's structure in the "opcode_patterns.h" header file.

While "opcode_patterns.h" matches instruction bit patterns to mnemonics, the meanings of the various bits in an instruction is documented by Arm Limited in on their developer website [1].

3.1 Instruction Execution

"Executing" the instruction is done by another helper function. This function receives the struct of bit fields and mnemonic, as generated by the "decode" function. In the simulator's current version it then performs a large "if – else if – else" test on the instruction's mnemonic. When it finds a match, the corresponding statements do whatever work the instruction requires (doing arithmetic on register contents, accessing a memory location, branching to a different memory address, etc.). If no match is found, an error message is displayed. It is this error message that informs the user about an unimplemented instruction. A portion of this "if – else if – else" struct is shown for illustration in Figure 1.

This is where the student coding activity occurs. Once the unimplemented instruction is identified, its desired behavior must be understood, then another "else if" clause can be added to do whatever the instruction needs.

4 Discussion, Further Work

This is "version 3" of the simulator, incorporating the "read-evaluate-print" loop to provide a text-based user interface. The available user commands include single-stepping through an executable, displaying information about the register contents following an instruction, running the entire executable to

[1]Single Instruction, Multiple Data

```
    if (!strcmp(ir->mnemonic, "nop")) {
        fprintf(logout, "\n%s (DO NOTHING)\n", ir->mnemonic);

    //---- ALU operations:

    } else if (!strcmp(ir->mnemonic, "add_32")
                || !strcmp(ir->mnemonic, "add_64")) {
        registers[ir->rd].dword = size_mask & (ALUinN + ALUinM);

    } else if (!strcmp(ir->mnemonic, "add_i")) {
        long unsigned result;
        ALUinN = (ir->rn == 31  ?  stack_pointer  :  registers[ir->rn].dword);
        ALUinM = (ir->lshift  ?  (ir->uimm12) << 12  :  ir->uimm12);

        result = (ir->regsize_mask & ALUinN) + (ir->regsize_mask & ALUinM);

        if (debug) {
            fprintf(logout, "  ALUinN %#lx  ALUinM %#lx  result %#lx\n",
                ALUinN, ALUinM, result);
        }
        if (ir->rd == 31)
            stack_pointer = result;
        else
```

Figure 1: A portion of the "execute()" function.

completion, displaying the executable's memory usage in hexadecimal format, verbose operation, help, and quitting the simulator.

The exercise of building the simulator was split into two parts because the task of reading and manually copying the source code appeared to be more than one lab session's worth of effort. The first part built the ELF-file loader portion and verified it. The second part added the fetch-decode-execute portion of the program, and then proceeded to ask for implementations of the missing instructions.

In use, most students did not read the code in any depth before highlighting, copying and pasting it. Only after running it and finding an unknown instruction did they pay attention to what any of the code did. On the other hand, the second part of the building process also included running all the executables, and adding the required instruction implementations to the "if – else if – else" structure. This made for a long and challenging second exercise.

The exercises can be reorganized to better reflect the amount of effort involved and the distribution of pedagogical content. In its next use the author will combine building the loader portion and the fetch-decode-execute portion into a single, larger exercise. This can be combined with more discussion of an executable's organization as code ("text") and data sections, and the way it is placed into memory. New, minor exercises will be added to explore the

operation of example instructions, and illustrate the rhythmic nature of "fetch – decode – execute" by single-stepping through some simple executables. Another executable will be written that does a bit more work including some memory accesses. Adding the unimplemented instructions will be presented as one or more exercises that explore the assembly/machine language in more detail, perhaps while examining the decode process.

The simulator itself may be rewritten to more closely emulate the datapaths described in [3]. This would replace the mnemonic comparisons with operations based purely on the control signals generated by the decode stage. The next version of this simulator is planned to operate this way, simulating the various components of the datapath such as the ALU and its control logic.

5 Conclusions

This Arm64 ISA simulator has been useful in providing hands-on activities for students in a Computer Organization course. It has been used twice so far, and students have reacted positively to the hands-on nature of the exercises. Its current version is good for demonstrating the overall fetch-decode-execute cycle. It is available on the github website at `https://github.com/bloomu-prof/Arm64-Simulator.git`.

References

[1] Arm Limited. *Arm A64 Instruction Set Architecture.* `https://developer.arm.com/documentation/ddi0596/2021-12/Base-Instructions`. Accessed 17 June 2022.

[2] Arm Limited. *Graphical Micro-Architecture Simulator.* `https://www.arm.com/resources/education/education-kits/legv8`. Accessed 13 June 2022.

[3] David A Patterson and John L Hennessy. *Computer organization and design ARM edition: the hardware software interface.* Morgan kaufmann, 2016.

[4] Unknown. *ARMv8 (AArch64) Instruction Encoding.* `http://kitoslab-eng.blogspot.com/2012/10/armv8-aarch64-instruction-encoding.html`. Accessed 3 March 2021.

A Kubernetes Framework for Learning Cloud Native Development*

Austin Reppert, Brian Montecinos-Velazquez,
Harrison Kahl, Rackeem Reid,
Danielle Rivas, Dominic Spampinato,
Hudson Zhong, Linh B. Ngo
West Chester University of Pennsylvania
West Chester, PA, 19380
{ar970724,bm935325,hk869465,rr935981}@wcupa.edu
{dr903713,ds946528,hz940494,lngo}@wcupa.edu

Abstract

This work describes a Kubernetes framework that can be automatically deployed on CloudLab, a federal cloud resource, to support learning activities in cloud computing education. The framework enables instructors and students to study cloud services' full product development lifecycle, including aspects such as automated deployment, availability, and security. This framework is easy to deploy, is freely available to academic institutions, and can be extended to support more advanced learning scenarios. The effectiveness of this framework is demonstrated through two *complex and full-stack* student projects.

1 Introduction

The gradual integration of cloud computing in computer science curriculum has resulted in an extensive set of knowledge areas (KAs) and learning objectives (LOs) [3]. Delivery of these KAs and LOs are further facilitated through the availability of cloud resources for education from both industry (e.g., Amazon

AWS, Google GCP, and Microsoft Azure) and academic (e.g., CloudLab [17] and Chameleon [15]). Recently developed cloud courses cover topics such as fundamental cloud concepts in virtualization of computing resources [8, 14, 21] and *everything-as-a-service* aspect of cloud computing [10, 19]. In [16], a course covering both fundamental cloud concepts and cloud-based computing models were developed. The course includes both infrastructure and service-related projects, and students carry out work using Amazon AWS resources. Similarly, [9] combines concepts on cloud computing fundamentals and big data infrastructures. The work of [5] provides a road-map for a course that can be adapted for either beginners or advanced students. The syllabus template leverages Amazon AWS, Google Cloud Platform, and Chameleon Cloud. To streamline the interaction between on-site, industry-based, and academic-based cloud resources, a management toolkit called EasyCloud has been developed [4].

Both industry-based and academic clouds have their own limitations. For industry-based cloud, resources for educational purposes are limited for and can require legal agreements at institutional level. These resources are more often than not proprietary and specific to each platform. Academic clouds are more freely available, but require more technical efforts to set up open-source platforms. Our work contributes to the development of cloud computing curriculum contents on academic platforms through the creation of a Kubernetes-based framework augmented with additional components such as secured registry, automation servers, traffic ingress, and certification services. These components support an environment that is conducive to project-based learning of *-as-a-service* topics.

The remainder of this paper is organized as follows. Section 2 presents the design and development of the framework. Section 3 describes how the framework enabled the development of two students projects. This section also discusses students' feedback and experience in using the framework. Section 4 concludes the paper and suggests future applications and improvements to the framework.

2 Framework Development

The framework starts out as an attempt to create a Kubernetes cluster to support cloud computing education on CloudLab. Over time, as students' projects become more complicated, there have been demands for bringing additional components, such as private registry and automation servers. The initial intention was to include these components as part of students' projects. However, installing and configuring these components are non-trivial and frequently detract from the primary goal of the projects, which is the develop-

ment and deployment of cloud services. This motivates the development of the framework.

The framework is designed to be deployed on CloudLab, an academic cloud. Funded by the National Science Foundation in 2014, CloudLab was built to provide researchers with a robust cloud-based environment for next generation computing research [17]. As of Summer 2022, CloudLab boasts an impressive collection of hardware. At University of Utah, there is a total of 785 nodes, including 315 with ARMv8, 270 with Intel Xeon-D, and 200 with Intel Broadwell. The compute nodes at University of Wisconsin include 270 Intel Haswell nodes with memory ranging between 120GB and 160GB and 260 Intel Skylake nodes with memory ranging between 128GB and 192GB. At Clemson University, there are 100 nodes running Intel Ivy Bridges, 88 nodes running Intel Haswell, and 72 nodes running Intel Skylake. All of Clemson's compute nodes have large memory (between 256GB and 384GB), and there are also two additional storage-intensive nodes that have a total of 270TB of storage available [18].

With CloudLab, users are able to program the necessary computing infrastructures, startup commands, and how they all fit together using Python. This *profile* is used to generate a resource description document, from which the actual cloud-based experiment is deployed. An example profile is shown in Figure 1.

2.1 Manual deployment

An early implementation of the framework focuses on installation rather than configuration. It was straight forward to setup automated installation scripts as part of the CloudLab profile [12]. Students were provided with instructions to configure Jenkins, an add-on automation server to support continuous integration/continuous delivery (CI/CD), and Kubernetes.

While this approach lessens the configuration burden, obstacles remain. Each new CloudLab experiment is randomly deployed on different hardware at different sites and is only active for 16 hours. This requires students to spend time to repeat the configuration process. It can take from 45 minutes up to one hour for this procedure to complete and can be a source of frustration for students.

2.2 Naive Helm-based deployment

In the next version of the framework, we leverage Helm [6], a package management framework for Kubernetes, to allow configuration parameters to be embedded in YAML files. Supporting components are now deployed and configured as Kubernetes pods via Helm. Bash scripts are used to detect the

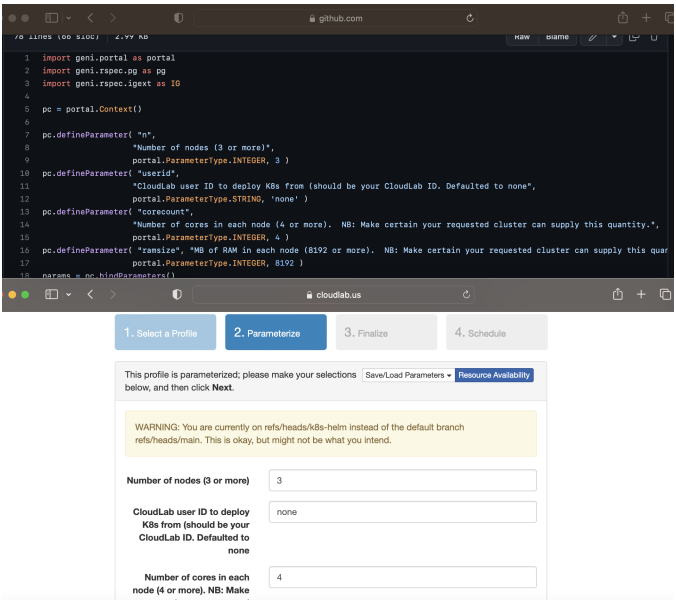


Figure 1: CloudLab's Python-based template and web interface

experiment's parameters such as hostnames and IP addresses and replace corresponding placeholders in the YAML configuration files with this information.

This approach streamlined many configuration process. It reduced the total amount of time to fully deploy the experimental infrastructure, which include the Kubernetes cluster and the supporting components, to approximately 20-25 minutes. However, there remains several limitations, one of which is the use of Docker Hub, a public container image repository, for the Publish stage of the automation process. Docker Hub provides users with access token to facilitate automated publishing. For security purpose, these tokens cannot be included with the CloudLab profile and have to be entered into Jenkins' settings after the experiment is fully deployed. This is yet another error-prone and tedious configuration step for students.

2.3 Enhanced Helm-based deployment

The current enhanced version of the framework is a result of an attempt to provide support for projects' security aspects. From this perspective, all projects' components are now kept completely inside the Kubernetes cluster (no more direct mapping to external host port via NodePort). An ingress service is deployed to help manage ingress/egress traffic, and a private Docker registry, in-

cluding certification managers, is setup. Two enhanced deployment approaches have been developed.

In the first approach, the Docker registry is manually installed directly on one of the computers that is part of the experiment and not inside the Kubernetes cluster. Certificates for the registry are generated and locally signed via automated scripts. In order for the Kubernetes containers to recognize the registry's self-signed certificates, they are deployed as accompanying DaemonSet components of all Kubernetes pods.

In the second approach, the Docker registry is now deployed via Helm inside the Kubernetes cluster. An automated script is developed to launch a certificate manager as a Kubernetes service. This script also generates a certificate to be signed by LetsEncrypt [2]. The signed certificate is then uploaded into the Kubernetes certificate manager and made available to all services. Having verified certificates reduces the amount of overall setup work because using self-signed certificates almost always requires a work-around for each piece of deployed software. Self-signed certs are also not trusted by default in browsers and may even be blocked by firewalls.

Both approaches include the ingress controller, an important aspect of Kubernetes deployments. Ingress controllers provide an easy way to attach certificates and route traffic to services. Additionally, by default Kubernetes deploys services on large port numbers like 30000. Non-standard ports may be blocked by firewalls. To route traffic from port 80/443 to the ingress controller a simple Nginx server on the master node was deployed via an automated script.

Both approaches accomplish their intended goal, which is to create a framework on which students can develop and deploy cloud services. The tradeoff between the two approaches include the placement of the private Docker registry and the creation of security certificates. In the first approach, a direct installation on hardware allows the registry to take advantage of resources like storage and IP addresses. The second approach follows a best practice of containing everything inside Kubernetes, including the private registry. However, additional configurations are needed, including the addition of Persistent Volumes and Persistent Volume Claims to support the registry storage and complex ingress traffic setup. The self-signed certificates are convenient, and in the context of a large class with multiple student groups, the risk of reaching LetsEncrypt certification request limit is low. On the other hand, self-signed certificates require additional efforts to be recognized by Kubernetes and its pods/containers. The details of these two approaches are available as public GitHub repositories [12, 11].

3 Student Projects

The efficacy and acceptability of the platform is demonstrated through the following two projects. These are student projects from previous semesters which could not be fully completed due to the inability to install and configure the supporting components of Kubernetes on the earlier manual version of the framework. Incidentally, key members of these projects are among the authors of this paper, and they provide the motivation for the improvements of the framework, which in turn, enable further completion of these projects.

3.1 A10dance

The Attendance Tracker (A10dance) is a full-stack web application that was used to expand the team's understanding of cloud computing [1]. A10dance provides an online service where students and professors can create and log into their accounts for attendance checking purposes. Students may increment their attendance score when they come to class, and professors can then check the attendance score of all students. The front-end of A10dance was built using ReactJS and includes a fully functioning webUI that allows the user to interact with the back-end and database and display the results of such interactions. The back-end and database was created using a combination of NodeJS and PostgreSQL, and they work in conjunction to execute functions such as creating or logging into an account.

In the beginning, A10dance had a number of minor bugs in the webUI and was missing some key features such as a functioning check-in button, filtered attendance list, and appropriate authorization so that professors and students only have access to specific items. At this point in time, security was not part of the project's consideration. While A10dance was able to successfully deploy on the early Kubernetes framework using DockerHub as the registry and Jenkins for CI/CD, one major limitation of the project was the manual process in which the deployment scripts and cluster configuration files need to be edited any time a new experiment was launched.

The enhanced Kubernetes framework allows the A10dance application to be deployed in a generic way. The deployment details can be abstracted into environment variables/secrets. By using the framework, few changes are needed to deploy the application to CloudLab when compared to a local deployment. The only changes required when deploying to CloudLab are the addition of secrets and the Jenkins pipeline. The project's major components are complete and operational. The group is now looking at further enhance the access control capability of the project by integrating more components such as Keycloak [20].

3.2 Chatbot

The University Chatbot is a full-stack, cloud-native project executed over CloudLab infrastructure [7]. It will provide users with an interface to make specific queries about campus events through a chat app, which rely on existing open-source frameworks for natural language processing. The front-end is a simple chat interface allowing users to ask about campus-related events. The back-end involves a deep-learning bot that can process user input and query a database for information. The database is populated by information scraped from the university's related sites. The project displays the utility of cloud-based development with concepts such as automation, containerization, and CI/CD. These are implemented through open source tools such as Docker, Kubernetes, and Jenkins.

Chatbot started out with a fully functional back-end as a proof of concept for the project. The WebScraper is containerized and consists of a Python script to scrape data and populate the Database. The Chatbot consists of a Natural Language Understanding (NLU) processing server and actions server developed within the Rasa framework for chatbots [13]. The NLU server was configured to receive a connection from the WebUI while the actions server is able to query the Database based on requests from the NLU server. Both servers were containerized and deployed within a multi-container pod. The front-end WebUI component was not at a functional state. It was deployed as an Nginx service made accessible through a NodePort protocol. The development of a *socket.io* application to interface with the Chatbot was unsuccessful. Each of the four components were configured for automated deployment through CI/CD pipelines in Jenkins. However, they lacked automated integration and required a manual build step.

The original development of the project was considerably slow due to the hurdle of configuring the infrastructure required for cloud-native development. A particular shortcoming was the manual setup of the Jenkins automation server required at the start of each CloudLab experiment. This tedious process dissuaded the development team from taking advantage of the benefits of CI/CD pipelines earlier in the development process, as evidenced by the lack of an automated build step in the pipeline implementation. The new framework's use of Helm to automate the deployment the Jenkins server made the concept of CI/CD more attainable. The project now benefits from full CI/CD pipeline integration, making the continued development of Chatbot more efficient. Another shortcoming in the original project was the lack of consideration for cloud security practices. Security deficiencies include the use of a public Docker registry for application images, the lack of ingress control and certification for the cluster, and hard-coded configuration values in project files. The inclusion of infrastructure to handle such security concerns has enriched the development process to be in line with production level security standards.

3.3 Discussion

The above framework (including both approaches) provide a secure cloud platform for students to develop and deploy complex full-stack services. By reducing the amount of efforts needed to install and configure the relevant add-on components for the Kubernetes platform, the development and deployment processes become streamlined. As a result, students are able to focus more on the technical details of their cloud services rather than the cloud platform.

The consistency and reliability of the framework also provides a solid starting point for students that need time to ramp up on their technical skill. In a discussion, one student said, *"having no knowledge on cloud-based applications before the class, I found that working through the hands-on experiments in my own time helped me to understand all of the different components that would be utilized in the project. There are plenty of parts in the project that I still ask to be clarified, but I understand the main functions each component plays in making the entire application run ..."*. Another stated, *"I found the development process of a cloud-based application to be very insightful. At work, the build and deploy process is partially abstracted through layers of automation, but the project helped improve my ability to troubleshoot build issues..."*. In the end, the framework facilitates the completion of these two projects and also provides opportunity for students to further improve upon their original design.

4 Conclusion and Future Work

The enhanced Cloudlab-based Kubernetes framework provides a convenient and reliable environment for learning to develop and deploy complex cloud services. The effectiveness of the framework has been demonstrated through the improvement of two student projects that could not be completed with an earlier version of the framework that lacked the automated installation and configuration of several supporting components. There remains many improvements that can be made to improve the framework. These include, but are not limited to, the followings:

- Additional parameterizations of Jenkins' credentials and automation of username and password hash generation to improve security.
- Deployment of load balancer and DNS servers inside Kubernetes.
- Development of rigorous user guide and documentation.

These improvements will contribute toward supporting more complex projects and have the potential for the framework to approach being a production-level

Kubernetes cluster. We intend to continue involving students in the implementation of these improvements in order to facilitate hands-on learning opportunities.

References

[1] *A10dance.* https://github.com/djrivas7400/A10dance.git/. 2022.

[2] Josh Aas et al. "Let's Encrypt: an automated certificate authority to encrypt the entire web". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 2473–2487.

[3] Joshua Adams et al. "Cloud computing curriculum: Developing exemplar modules for general course inclusion". In: *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*. 2020, pp. 151–172.

[4] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. "Easycloud: a rule based toolkit for multi-platform cloud/edge service management". In: *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2020, pp. 188–195.

[5] Cosimo Anglano, Massimo Canonico, and Marco Guazzone. "Teaching Cloud Computing: Motivations, Challenges and Tools". In: *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2020, pp. 300–306.

[6] Andrew Block and Austin Dewey. *Managing Kubernetes Resources Using Helm*. Packt Publishing, 2022.

[7] *Chatbot.* https://github.com/bmv0161/csc603-project.git/. 2022.

[8] Ling Chen et al. "Introducing cloud computing topics in curricula". In: *Journal of Information Systems Education* 23.3 (2012), p. 315.

[9] Debzani Deb, Muztaba Fuad, and Keith Irwin. "A module-based approach to teaching big data and cloud computing topics at cs undergraduate level". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019, pp. 2–8.

[10] Lee Gillam, Bin Li, and John O'Loughlin. "Teaching clouds: Lessons taught and lessons learnt". In: *Cloud computing for teaching and learning: Strategies for design and implementation*. IGI Global, 2012, pp. 82–94.

[11] *Kubernetes framework: Containerized Registry.* https://github.com/AustinMReppert/csc603cloud/tree/k8s-helm/. 2022.

[12] *Kubernetes framework: Registry on Hardware.* https://github.com/CSC603-WCU/csc603cloud/tree/k8s-helm. 2022.

[13] Paula Lauren and Paul Watta. "A conversational user interface for stock analysis". In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 5298–5305.

[14] Peng Li and Lee W Toderick. "Cloud in cloud: approaches and implementations". In: *Proceedings of the 2010 ACM conference on Information technology education*. 2010, pp. 105–110.

[15] Joe Mambretti, Jim Chen, and Fei Yeh. "Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn)". In: *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. IEEE. 2015, pp. 73–79.

[16] M Suhail Rehman et al. "A cloud computing course: from systems to services". In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 2015, pp. 338–343.

[17] Robert Ricci, Eric Eide, and CloudLab Team. "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications". In: *; login:: the magazine of USENIX & SAGE* 39.6 (2014), pp. 36–38.

[18] Robert Ricci et al. *CloudLab Phase II: Community Infrastructure To Expand the Frontiers of Cloud Computing Research.* https://www.nsf.gov/awardsearch/showAward?AWD_ID=1743363. 2017.

[19] Ian Sommerville. "Teaching cloud computing: a software engineering perspective". In: *Journal of Systems and Software* 86.9 (2013), pp. 2330–2332.

[20] Stian Thorgersen and Pedro Silva. *Keycloak-Identity and Access Management for Modern Applications*. Packt Publishing, 2021.

[21] Xinli Wang et al. "Introducing cloud computing with a senior design project in undergraduate education of computer system and network administration". In: *Proceedings of the 2011 conference on Information technology education*. 2011, pp. 177–182.

How Many Languages Does It Take to Be a Programming Languages Course?*

Michel Charpentier and Karen H. Jin
University of New Hampshire
Durham, NH 03824
{Michel.Charpentier,Karen.Jin}@unh.edu

Abstract

Programming languages courses are an important component of computer science curricula that aim at building students' transferable skills in learning new languages. These courses tend to come in two main flavors: the interpreter/compiler style on the one hand, with a focus on language definition and implementation, and survey courses on the other hand, which compare languages through the features they support. Typically, interpreter/compiler courses are narrow and deep in their focus, while survey courses favor a broader coverage. In this paper, we report on our experience in teaching a course that is neither centered on programming language implementation, nor based on a comparison of multiple languages. We believe that our course offers a good depth/breadth balance, avoids pitfalls common to the more classic approaches, and nicely complements the larger needs of a standard computer science curriculum. We see our approach as the result of a trend in modern, general-purpose programming languages, notably the emergence of a common set of features, and the arising of so-called hybrid or multi-paradigm languages.

Keywords: Programming languages, language features, paradigms, upper-level undergraduate education.

1 Background

1.1 Overview

An important component of a student's computing education is the ability to work with new programming languages. It is well-known that novice learners have difficulties transferring their knowledge from one language to another [7, 8]. Programming languages courses are often used to introduce language features and programming paradigms to help students build transferable skills that facilitate the learning of new languages.

Our institution offers an upper-level course designed primarily for Computer Science undergraduates and titled *Programming Language Concepts and Features*. Among students and faculty, it is universally known as the *Programming Languages* course. This may seem like a reasonable contraction, but it changes the grammatical quantity of the word *language* from singular to plural. Given that the course is currently taught using a *single programming language*, this shift is matter for reflection. This article discusses the apparent contradiction of a "programming languages" course (PL course, in short) that uses a single language. The paper describes the motivations behind the design of the course in its current form and discusses what we believe to be the major benefit of our approach—namely, an extensive coverage of essential programming paradigms in a hands-on fashion that prepares students for the complexities of modern programming languages. The discussion draws on more than twenty years of experience teaching the course—using one or multiple languages.

1.2 Illustration

Alexis and Brady are two students. Alexis enrolls in a PL course that uses Java, Python, and JavaScript as its sample languages. In this course, students are given the exercise of implementing a solution to the following problem:

> *Fill an array of N random integers between 0 and 99. Count how many integers in the array are less than 50.*

Alexis, whose background is in Java, quickly produces a first solution:

Java

```
var nums = new int[N];
for (int i = 0; i < N; i++)
    nums[i] = rand.nextInt(100);
var count = 0;
for (int i = 0; i < N; i++)
    if (nums[i] < 50) count += 1;
```

Alexis then moves on to JavaScript and writes code that, except for the use of a non-integer random generator, is not very different from the Java version:

JavaScript

```
const nums = []
for (let i = 0; i < N; i++)
    nums[i] = Math.floor(Math.random() * 100)
let count = 0
for (let i = 0; i < N; i++)
    if (nums[i] < 50) count += 1
```

The final program is written in Python, where the loop looks a little different, and code blocks are structured using indentation:

Python

```
nums = list(range(N))
for i in range(N):
    nums[i] = rand.randrange(100)
count = 0
for i in range(N):
    if nums[i] < 50: count += 1
```

Brady takes a different course, in which students are asked to solve the same problem in Scala. Brady's first program also builds on a Java background:

Scala

```
val nums = new Array[Int](N)
for i <- 0 until N do
    nums(i) = rand.nextInt(100)
var count = 0
for i <- 0 until N do
    if nums(i) < 50 then count += 1
```

Students are then taught about *higher-order functions*, *lambda expressions*, and *for-comprehensions*, and then asked to rewrite their programs, leading Brady to produce the following code:

Scala

```
val nums = Array.tabulate(N)(i => rand.nextInt(100))
var count = 0
for n <- nums do
    if n < 50 then count += 1
```

Later, the course covers *lazy evaluation*, *η-abstraction* and more *higher-order* functions, after which Brady writes yet another Scala variant:

Scala

```
val nums = Array.fill(N)(rand.nextInt(100))
val count = nums.count(_ < 50)
```

Alexis is left with the feeling that programming languages are similar, and that it is relatively straightforward to learn a new language. Brady understands that programming languages define numerous constructs, and that there is more than one way to crack an egg. Both lessons are valuable. Programming languages do share a lot of syntax and semantics, and experienced programmers often switch languages seamlessly. But modern languages also offer powerful features that can improve productivity and code quality. Programmers are expected to adapt and to repeatedly transfer programming skills from one language to another, and skilled programmers know what features to look for in a new language and how to use them effectively.

As it happens, Brady takes on a developer job in Python after graduation, and soon faces the programming problem already solved in class. After browsing the Python documentation with Scala in mind, Brady writes this implementation:

Python

```
nums = [rand.randrange(100) for i in range(N)]
count = len([n for n in nums if n < 50])
```

Note how this code is arguably superior to Alexis's variant despite Brady having learned no Python in the PL course.

2 A Modern PL Course

2.1 Flavors of PL Courses

Established Computer Science curricula (e.g., [2]) prescribe the coverage of programming languages, usually in terms of outcomes. In practice, the range of courses vary, but most can be fit into one of two categories:

- *Programming languages definition and implementation.* These courses have a strong focus on the creation and implementation of programming languages: parsing, semantic analysis (including type-checking) and code generation. They often involve an interpreter or compiler project, as in a typical "write a Scheme interpreter in Scheme" course.

- *Programming languages survey and comparison.* By contrast, these courses sidestep implementation issues to focus on the practical use of a variety of languages. Assignments typically involve writing programs using programming languages that illustrate distinct styles, or *paradigms*. A typical course covers functional, logic and object-oriented programming.

In all cases, the course's broad objective is to help students become better programmers. Implementation courses are founded on the idea that many students have a better grasp of a concept if they have some understanding of its implementation. These courses tend to have a deep but narrow focus on the essential aspects of a language—subprograms, scoping, type-checking, exceptions, garbage-collection, etc.

On the other hand, survey courses consider languages from the standpoint of a user—a programmer. The goal here is to understand what programming language constructs are available, what they do, and how they can be used effectively when writing programs. How the features are implemented is peripheral.

There are strengths and weaknesses to each of these approaches, and schools may use one, the other, or both in their effort to improve their students' programming skills. If one has to choose—given how much a typical computer science curriculum needs to cover, one indeed often *has* to choose—we would advocate against courses centered on programming language implementation.

Our main objection is that the approach is not cost effective. Because they discuss each programming language aspect in great detail, these courses tend to cover fewer features than usage-based courses. They are also limited to simplified variants of features, easy enough for students to implement, but which do not necessarily reflect the reality of a modern programming language [10]. Furthermore, this simplification can give students a false sense that they know exactly what real compilers and run-time systems do, and lead them to shape their program accordingly, resulting in ineffective "optimizations" and overall worse code quality.

Most programmers can use a whole range of language features without knowing how they are implemented. Programming languages are abstractions, designed to insulate the programmer from machine-specific details. Being able to deal with abstractions is an essential skill that we need to teach our students. One cannot expect programmers to restrict themselves to features they understand because they have implemented them.

Recommendation: Focus on effective use of programming language features and leave language design and implementation issues to electives, such as courses on formal semantics or compilers.

2.2 Programming paradigms

It has long been observed (e.g., [4, 3]) that typical curricula have shifted from *programming languages* to *programming paradigms*. If the course taken by Alexis is limited to the *imperative* paradigm—using various languages such as Java, Python, and JavaScript—it won't be a very good course.

The *object-oriented* paradigm has long been mainstream, and students are often exposed to it in multiple courses, besides the PL course (e.g., an introductory *objects-first* course, and/or a software engineering course that covers object-oriented design). Depending on other courses in the curriculum, a PL course may still discuss some aspects of object-oriented programming, while safely assuming that others have already been covered. The same argument obviously applies to the imperative paradigm, unless students move directing from a *functions-first* introductory course to the PL course, which is unlikely.

Almost all PL courses include the *functional* paradigm, which has recently been gaining a lot of attention. Unless a curriculum includes an entire core course on functional programming, this paradigm ought to be thoroughly covered in a modern PL course.

The last traditional paradigm is *logic* programming, invariably discussed in the context of Prolog. The intent here is to expose students to the notion of *declarative* programming. However, logic programming is only one very specific flavor of declarative programming. It is doubtful that their experience with Prolog—which is arguably more a search engine than a programming language—will help students deal with the other forms of declarative programming they may encounter (e.g., database query with LINQ or SQL, XML processing using XSLT, or any particular Domain Specific Language).

Some courses have begun to include *concurrent programming* as one of the standard paradigms. Several curriculum standardization bodies [2, 6, 1] now emphasize the importance of parallel and distributed computing in the undergraduate curriculum. Although many related concepts are often covered in other courses, such as *operating systems* or *networks*, a PL course can also be a place where students gain practice in concurrent or distributed programming.

Finally, *event-based* or *reactive* programming is a paradigm that is becoming crucially important (and is listed explicitly in [2]). There are overlaps between this paradigm and both functional and concurrent programming, but it increasingly deserves its own coverage in a modern curriculum.

Recommendation: Depending on its place in the curriculum, a PL course may assume basic understanding of *imperative* and *object-oriented* programming, and then focus on their more advanced features. *Functional*, *concurrent*, and *reactive* programming are becoming a must; *logic* programming can be skipped in favor of more current flavors of *declarative* programming.

2.3 Programming languages

The choice of languages in a typical survey course is driven by the need to cover multiple paradigms. Object-oriented programming is often covered in a mainstream language like C++, C# or Java, or in a purer language like Smalltalk or Eiffel. Functional programming is usually presented in the context of Lisp, Scheme, Haskell or an ML variant such as SML or OCaml. As mentioned earlier, logic programming is always in Prolog. As for concurrent programming, courses may rely on mainstream language with threads, like Java, or more specialized languages like Erlang, Ada or Go.

In [5], the author makes the interesting remark that, while instructors focus on concepts and features, students are very much focused on languages, and learn the concepts through the languages. There is a risk, however, of them associating the features with the syntax, and of confusing languages with paradigms [9]. It would be a crippling mistake for a student to assume that a functional program necessarily uses deeply nested parentheses, or that a concurrent program must use symbols like `wait`, `!` or `?`. Even if Alexis's course continues with a functional programming assignment in Scheme, there remains a danger that students will associate functional programming with Scheme exclusively and not realize that some of these functional patterns can also be written in Python, JavaScript or Java.

Fortunately, the advent of so-called *hybrid* or *multi-paradigm* programming languages has made it possible to discuss multiple paradigms within the same language. Writing imperative, object-oriented, functional, concurrent and/or reactive programs with the same syntax can help differentiate the features from the languages. As an added benefit, limiting the number of languages reduces the number of "context switches" students need to make in terms of syntax, idiosyncrasies, development tools and ecosystems.

Recommendation: Use *fewer* languages than paradigms and practice with multiple paradigms within the same language as a way to avoid a one-to-one mapping of languages to paradigms in students' minds.

2.4 Scope

Many students think of programming languages primarily in terms of *syntax*. While it is understandable that beginners might regard learning new syntax as a hurdle, the syntax of a programming language is not where difficulties lie and should certainly not be a focus of teaching. It has been observed (e.g., [10]) that the barrier some experienced programmers face with today's increased use of functional programming has very little to do with syntax, but stems

from a failure to understand the fundamental principles underlying functional programming.

There is a danger, in a PL course, to overly focus on the language and its features and to neglect the purpose and proper use of these features. For example, the concept and syntax of a lambda expression in a particular language is uninteresting in itself. Power comes from using well-chosen lambdas as arguments to well-chosen higher-order functions, a skill that requires learning functional programming principles, at least at an introductory level. Similarly, coverage of support for concurrent programming—e.g., locks, semaphores, futures, promises—cannot be narrowly limited to programming language constructs, but ought to include a broader discussion of fundamental aspects of concurrency like asynchronicity, non-determinism, and race conditions.

Without becoming a full-fledged object-oriented programming, functional programming or concurrent programming course, a PL course must cover enough of these programming paradigms to help students make sense of supporting language constructs.

Recommendation: Include an introductory coverage of language-independent programming paradigms in the scope of a PL course.

2.5 Level

Although the author of [10] argues, like we do, against using simplified, toy languages in teaching, he also warns that hybrid languages are too complex for teaching purposes, and that they could result in overly high cognitive loads. A counter-argument, however, is that programming languages *are* becoming more complex, and that PL courses have not always kept up with this trend. Many current and future mainstream languages will incorporate multiple paradigms, and their complexity makes it all the more necessary that students be prepared for them. Kotlin, for instance, a popular emerging language with support for object-oriented, functional, and concurrent programming, will be easier for students to learn if they already understand the concepts involved in its programming constructs. Even venerable Java defines constructs like `Stream` or `CompletableFuture` that mix functional and concurrent patterns in non-trivial ways. We believe that a mastery of modern programming languages requires a solid grasp of concepts that deserve to be included in our curricula.

Recommendation: Do not shy away from advanced features of programming languages, as these tend to be the most difficult for students to learn on their own at a later time.

2.6 Assignments

It is an undisputed fact that students find it easier to understand the code they read in class than to write their own. Students who know enough of a programming language feature to read code that uses it, may not understand that feature well enough to choose it and apply it effectively when faced with a particular programming problem. For example, most students have no difficulty with the mechanics of encapsulation, or recursion, or polymorphism, but many struggle with the proper use of these features. The time spent in class to explain the syntax and semantics of a particular language feature is usually dwarfed by the time needed for in-class examples and homework practice.

In some ways, PL courses face challenges similar to that of introductory courses. Instead of students discovering basic features of imperative programming, as they would in a typical first-year course, they are now learning new, more advanced features, which feel to them just as new and confusing as loops or conditionals were in their first programming course. The same hands-on approach is needed for students to become familiar with the new features and to know when and how to use them.

Recommendation: Practice, practice, practice.

2.7 Performance consideration

The discussion of code performance issues with students is always delicate. On the one hand, we want to avoid students obsessing over performance. This is often pointless—"optimizing" the implementation of a deficient algorithm will have little effect on performance—or counterproductive, resulting in code that is harder to read and maintain. On the other hand, modern, feature-rich programming languages, while they can increase a programmer's productivity and improve software quality, also have the potential to lead to very inefficient code, in CPU or memory usage. There are just more ropes to hang yourself with, performance-wise, in Java or Kotlin that there are in C or FORTRAN. Students need to be warned of such dangers, lest they unfairly dismiss powerful languages as being "slow."

Recommendation: Forewarn students of potential pitfalls and penalize them for blatant inefficiencies, such as quadratic implementations of functions that should be linear.

3 Conclusion

In a magical world of infinite credit-hours, a PL course could combine the strengths of Alexis's course—exposure to multiple languages—with the outcomes of Brady's class—understanding and proper use of advanced features—and even include a discussion of implementation concerns. In reality, however, courses will have to choose what to cover.

We believe that students are best served by a course that covers a large set of features, along with the important programming paradigms that underpin them—imperative, object-oriented, functional, declarative, reactive and concurrent programming. A focus on feature-rich, multi-paradigm languages, like Kotlin or Scala, makes it possible for a course to teach advanced concepts in a practical, hands-on manner. These concepts tend to be out of reach of courses that rely on simpler (or even "toy") languages. They also would be the hardest for programmers to learn on their own and therefore deserve to be included in a modern curriculum. The principles and structure we advocate in this paper can be used as a template for such a course.

References

[1] ABET. *Criteria for Accrediting Computing Programs*. Tech. rep. ABET Computing Accreditation Commission, Nov. 2019. URL: https://www.abet.org.

[2] ACM/IEEE-CS Joint Task Force on Computing Curricula. *Computer Science Curricula 2013*. Tech. rep. ACM Press and IEEE Computer Society Press, Dec. 2013. DOI: 10.1145/2534860. URL: http://dx.doi.org/10.1145/2534860.

[3] Kim N. King. "The Evolution of the Programming Languages Course". In: *Proceedings of the Twenty-Third SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '92. Kansas City, Missouri, USA: Association for Computing Machinery, 1992, pp. 213–219. ISBN: 0897914686. DOI: 10.1145/134510.134553. URL: https://doi.org/10.1145/134510.134553.

[4] P. A. Luker. "Never Mind the Language, What about the Paradigm?" In: *Proceedings of the Twentieth SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '89. Louisville, Kentucky, USA: Association for Computing Machinery, 1989, pp. 252–256. ISBN: 0897912985. DOI: 10.1145/65293.71442. URL: https://doi.org/10.1145/65293.71442.

[5] Saverio Perugini. "Emerging languages: An alternative approach to teaching programming languages". In: *Journal of Functional Programming* 29 (2019), e13. DOI: 10.1017/S095679681900011X.

[6] Sushil K. Prasad et al. "NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing: Status Report". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE '18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 134–135. ISBN: 9781450351034. DOI: 10.1145/3159450.3159632. URL: https://doi.org/10.1145/3159450.3159632.

[7] Ethel Tshukudu and Quintin Cutts. "Semantic Transfer in Programming Languages: Exploratory Study of Relative Novices". In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '20. Trondheim, Norway: Association for Computing Machinery, 2020, pp. 307–313. ISBN: 9781450368742. DOI: 10.1145/3341525.3387406. URL: https://doi.org/10.1145/3341525.3387406.

[8] Ethel Tshukudu et al. "Teachers' Views and Experiences on Teaching Second and Subsequent Programming Languages". In: *Proceedings of the 17th ACM Conference on International Computing Education Research*. ICER 2021. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 294–305. ISBN: 9781450383264. DOI: 10.1145/3446871.3469752. URL: https://doi.org/10.1145/3446871.3469752.

[9] Michael R. Wick and Daniel E. Stevenson. "A Reductionist Approach to a Course on Programming Languages". In: *SIGCSE Bull.* 33.1 (Feb. 2001), pp. 253–257. ISSN: 0097-8418. DOI: 10.1145/366413.364595. URL: https://doi.org/10.1145/366413.364595.

[10] Daeng Zuhud. "Some prospective approaches for the shift of programming paradigms". In: July 2013, pp. 87–93. DOI: 10.1145/2503859.2503873.

Alternative Assessment for Computer Science Classrooms*

Gabriel De Pace and Edmund A. Lamagna
Computer Science and Statistics
University of Rhode Island
Kingston, RI 02281

Abstract

Computer Science is being taught in more K-12 classrooms nationwide than ever before. Teachers must have the tools and be trained to assess student understanding and ability. The lack of high quality, readily available assessments makes the problem even more difficult. Our research uses a concept map task as a classroom assessment. The task can be administered online and is scored structurally by a computer program. The study uses Code.org CS Discoveries Units 2 and 3 in two different classrooms: one with a teacher at an urban school who was newly trained in computer science and the other with a veteran computer science teacher at a suburban school. We found evidence of correlation between the traditional assessments and the scores generated by the concept map task. Alternative assessment methods benefit all students. With our results we believe that our assessment is a practical complement to other irreplaceable forms of assessment such as writing code.

1 Introduction

As more and more students across the country are learning about computer science, it is putting a strain on teachers who likely were not trained in this

subject. According to a 2016 survey study of computer science teachers in K-12 in the US, "the lack of access to quality computer science assessment tools makes it difficult for them to accurately gauge what students are learning" [37]. Some of the teachers in the study reported that a book or resource with effective questions about topics or concepts, such as looping or if statements, would be very helpful.

Furthermore, the students come from varied backgrounds and levels of preparation. One of the major thrusts of computer science at all levels is to increase the participation of these diverse learners. It benefits students and instructors alike to have access to alternative means of assessment. The students gain because they have more opportunity to demonstrate what they know. Instructors can ignite creativity, inspire confidence and have more tools at their disposal to measure student success.

Concept maps have been fairly thoroughly studied in the literature for assessing student knowledge. A barrier to their adoption has been the difficulty in reliably scoring them. The contributions of this work are the following:

- to use concept maps as assessment in the computer science K-12 classroom for the first time,
- to offer an automated scoring program to easily and reliably score concept maps, and
- to suggest their inclusion in the computer science classroom as an alternative and supplementary assessment.

2 Related Work

2.1 Novak's Concept Maps

Concept Maps are graphical tools to organize and represent knowledge [25]. They consist of concepts and the relationships between concepts. The concepts are drawn in ellipses and the relationships are drawn as lines linking two concepts with a word or short phrase. The combination of concept, linking phrase and concept forms a proposition, which should be a true statement. The concepts should be arranged in a hierarchy, with general or broad concepts at the top, and more specific ideas nearer the bottom of the graph. In order to give context to the concepts, a focus question is provided. An example of a concept map made by a student in this study is shown in Figure 1.

Concept maps have successfully been used to summarize complex information in a compact and efficient way [25, 27], for increasing reading comprehension and learning, for facilitating note-taking, for acquiring a foreign language, and for summarizing knowledge [32, 36, 24]. Some of the concept maps were created as a pre-writing activity, something authors did to help them summarize their knowledge and enhance organizational thinking before beginning to

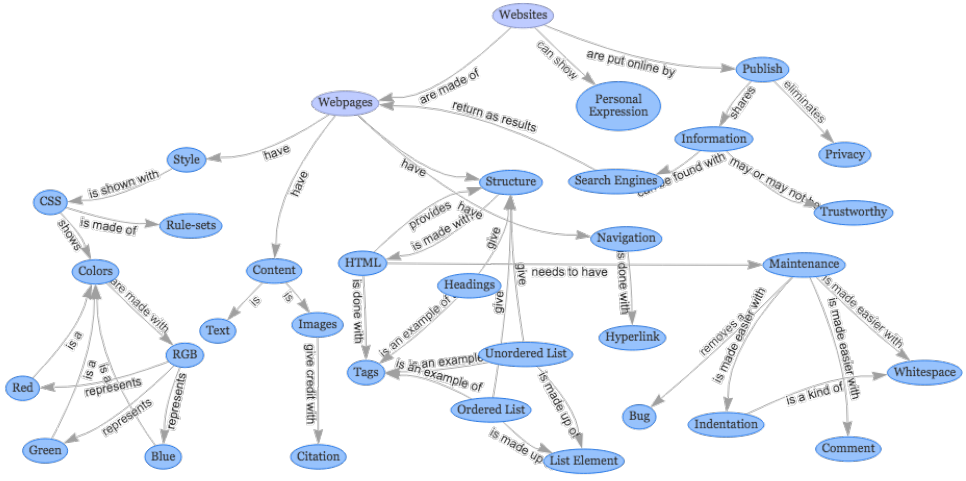


Figure 1: An example of a concept map illustrating the concepts in ellipses with lines between them forming propositions. The image is taken from the interface as the student was allowed to draw this concept map on the computer screen. At the start of the task, the concepts were given and the student was not allowed to add or remove them. The student adds, removes or edits links and rearranges the concepts as desired.

write. Concept maps have also been studied for use as an assessment tool, chiefly in science classrooms [22, 19, 31, 21, 28].

Despite these positive studies using concept maps as assessment tools in science classrooms, as [29] points out there are some barriers to their widespread adoption. Concept maps need to be shown to be valid assessments, applicable to the domain being assessed. Showing validity and applicability is made more difficult due to the variations in the concept map task and the consistent and reliable scoring of the maps. From an educator's point of view, a task typically involves writing a prompt and providing several concepts and linking words associated with that prompt.

This study responds to the barriers mentioned above. The validity and applicability are addressed by correlating concept map results with traditional assessments in the classroom. The concept map task is specific to the curriculum being used. Similar task creation and concept map scoring are handled by automating these jobs. Computer programs ease the burden on the classroom teacher and allow for consistency and reliability in scoring concept maps.

There is significant literature concerning the evaluation of concept maps [35, 19]. Scoring methods can be divided into three broad categories: holistically, with emphasis on concept map quality [7, 23], qualitatively [8, 20], and quantitatively [29, 19, 21]. Many of the quantitative scoring algorithms focus on the structure and components of the concept map itself. Several computer programs, Cmapanalysis [26], C-Tools [15], COMPASS [13] and CRESST [16], exist with some automatic assessment tools or used a tool to score the maps [2]. Other studies suggest a weighted map, [9], closeness index to some expert map [1, 20], and hybrid approaches, combining some or all of these techniques to derive a score.

Indeed, at least one study suggested that the concept maps of students can be qualitatively evaluated by the teacher and taken as a snapshot of understanding and used primarily for formative assessment [4, 34], removing the need for scoring at all. There has even been interest in some interactivity during concept map construction, either with teachers, peers or the computer. [36, 5, 18, 17]

Our approach to automated scoring is discussed in Section 3.6.

2.2 Related Work in Middle School Computer Science Assessment

Several studies have examined assessment in middle school computer science in recent years.

The research work by Boe et al. [3] introduces an automated process to gather information about Scratch programs. It does this by performing a static analysis, that is, it counts the number of instances of particular blocks of Scratch code. One good point raised in the paper is the inherent difficulty

of evaluating performance based scripts and codes. Though not limited to Scratch, it is a feature of Scratch that the programs are written in blocks and the way to evaluate the code is by running the program and exploring the possibly different paths of execution. Running each script, possibly several times, is a process that can be time consuming since the action of the script should be witnessed as opposed to the script producing text output. If a correct text output existed, then a line by line file comparison could be easily done[1]. A given Scratch project may not have such a static solution file. The main idea of the paper is to provide some automatic project evaluation to assist in Scratch script assessments. Boe et al. admit that their automated assessment framework is meant to supplement the manual scoring of projects. The weakness here is that the framework is limited to counting the blocks of code written in Scratch. Particular Scratch elements will need to be present in a script in order for it to be correct, but how it is used, that is, if it is correctly used is not determined by the tool. For example, if an assignment requires the use of conditionals then counting the if-statements would be of some limited use.

Salac et al. [30] take a closer look at how to assess Scratch code made by elementary school students. They developed an automated system to search student code and identify candidate code snippets that can be extracted to make multiple-choice, fill-in-the-blank and open-ended questions. The main thrust was to see if students did better when asked about generic code or about code they had written themselves. The authors hypothesized that knowing what the code does, the *function* of it, can be known when others' code is reused, but understanding how the code works, or its *structure*, is necessary if the student wrote the code himself. The findings were that when students answered questions about their own code in its original context, they answered correctly more often than when presented generic code. However, if their own code was used, but the question put it in a new context, the students performed worse. It seems the students were remembering what they did more than what the questions were necessarily asking. The conclusion was that students were more likely to answer explanation questions about their own code, but were less thorough with the explanation than those who had to explain generic code. The contribution of this work is the automated program that can find candidate code in the student projects that could be used in a personalized assessment. There is great value in a personalized assessment, however, it seems that most of the students simply remember what it is that they wrote rather than have an understanding of how or why it works.

The traditional methods of assessing computer science concepts and programming ability are expert created multiple-choice, fill-in, and free-response

[1]For instance, with the use of the standard Unix file comparison program diff.

style questions, interviews with students and inspection of artifacts, statically or by executing code. These are effective, but they are time intensive and rely on instructor expertise.

3 Methodology

This study uses the CS Discoveries curriculum as a basis for Computer Science concepts and measures them with a concept map assessment task. Details about CS Discoveries are given in Section 3.1. To address the concerns about validity and scoring the current work proposes to

1. create and administer unit tests in line with the CS Discoveries curriculum,
2. have students complete assessments made and scored by their classroom teacher,
3. create a concept map task that students can use to make concept maps about the units, and
4. score the concept maps with a computer program.

Finally, look for a correlation in the scores of the first two types of assessment and the concept maps.

3.1 CS Discoveries

The Code.org CS Discoveries (CSD) curriculum launched around 2017 and is geared toward students in grades 6-10. It provides excellent resources and links the learning objectives back to the Computer Science Teachers' Association and the K-12 Framework for Computer Science for guidance [11, 6, 10]. The curriculum has projects and meaningful checkpoints for informal and formative assessments, but when it was first introduced, did not provide summative assessments beyond the projects [10]. The data for this study were collected at that time. Since then, Code.org has added a summative assessment to the end of the units.

CSD Unit 2 invites students to create and share their own web pages. It also presents an excellent opportunity for students to think carefully about what information they are sharing with the world and the importance of privacy. Code.org has built a tool called Web-Lab that runs in a web-browser and allows users to type HTML and CSS tags into files in one window frame and see the results in another window frame.

CSD Unit 3 is about drawing with the computer, programming interactive animations and creating games. The Game Lab is a Code.org built program that turns a web-browser into an interactive drawing program to aid in teaching coding. The Game Lab is divided into two panels, one for the code and a second

that displays the results of the code's execution. The students do the actual coding using a subset of javascript.

3.2 Creating Unit Tests

The unit tests created by the researcher were made to be in line with the Code.org curriculum and the Computer Science Teachers' Association (CSTA) standards. Each of the questions can be traced to the curriculum lesson and also uses the materials from the class. Therefore the material should be familiar to the students and cover the relevant parts of the curriculum. There is an example question from the Unit 3 test shown in Figure 2. The content validity traces are detailed in [12].

3.3 Establishing the Concept Map Task

A researcher in Germany used concept maps to measure knowledge of incoming Computer Science undergraduates [24]. The work involved collecting large numbers of concept maps using an online tool called CoMapEd, developed by the researcher, Professor Mühling. The tool works in modern web browsers and allows an investigator to set the desired parameters of the concept map task such as the focus prompt students see and whether students may create their own concepts or must use those provided. A 6-digit hex number identifier is generated when a student begins creating a concept map that allows the map to be revisited in another session at a later time. We asked Professor Mühling for permission to use his tool to collect our data and he graciously granted us access.

Some possible concept map tasks include filling in blanks in concepts or in links on completed concept maps [31]. Other tasks involve having students choose from a bank of concepts [24] or having students segment and structure concepts [17]. In fact 739 possible concept map tasks were outlined in [33] by varying the component pieces of concept maps and what mappers must do with them.

For our study, the concept map task is made up of a focus question and a set of concepts. The concepts were taken from the curriculum materials and selected by the researchers based on their relevance to the goals of the lessons. The concept lists for Unit 2 and Unit 3 are shown in Tables 1 and 2. The students must then connect related concepts to form propositions and add a linking word to label the connection. The students are not allowed to add concepts, nor can they remove them.

For the Unit 2 concept map task, 34 concepts were selected and listed in Table 1. After reviewing Chapter 1 from CSD Unit 3, we extracted 38 concepts to be used in the concept map task as shown in Table 2.

Table 1: Complete list of 34 concepts extracted from CS Discoveries Unit 2.

Websites	Webpages	Content	HTML	Structure
Text	Tags	Headings	Personal Expression	Images
Red	CSS	Ordered List	Unordered List	List Element
Citation	Bug	Comment	Indentation	Whitespace
Maintenance	Navigation	Hyperlink	Publish	Style
Rule-sets	Green	Information	Search Engines	RGB
Trustworthy	Privacy	Blue	Colors	

Table 2: Complete list of 38 concepts extracted from Chapter 1 of CSD Unit 3.

Boredom	Problem	Computer	Entertainment	Self-expression
Shape	Rectangle	Ellipse	Height	X- Y- coordinates
Stroke	Color	Parameter	RandomNumber	Variable
Value	Sprite	Properties	Location	Animation
Grid	Width	Label	Counter Pattern	Keyboard Input
Visible	Draw Loop	Size	Debugger	Debug
True	Mouse Input	Else Clause	Boolean	Conditional
False	Fill	Boolean	Expression	Expression

3.4 Recruiting Classroom Teachers

The researchers recruited two classroom teachers to help us in the study. One was from School A, an urban school with a teacher new to computer science. The other was an experienced computer science teacher from a suburban school we are calling School B. Both teachers were new to CS Discoveries and attended Professional Development given by the researchers.

3.5 Research Study

The teacher from School A taught Unit 2 for a quarter. The final project was to create a Personal Web Page. The teacher also administered the unit test created by the researchers and the concept map task. The concept maps were provided and test scores were reported to the researchers as anonymous data, that is, without any student names or other information.

School B instead taught Unit 3 and similarly administered the unit test and concept map task. This teacher also made a vocabulary test and reported the scores for both assessments along with the students' concept maps as anonymous data.

The demographics are then treated in aggregate and in total, there were

189 students in the study, all in the 8th grade, between 13 and 15 years old, 98 girls and 91 boys.

3.6 Scoring the Concept Maps

The concept maps were scored using a computer program developed by the researcher. The scores are based only on the structure of the maps themselves, that is, the propositions found in the concept maps. The program takes as input a scoring rubric for each unit to be scored along with all the concept maps. In order to generate the scoring rubrics, the researcher evaluated all possible propositions available from the pool of concepts. Each proposition was given a rating, $\{-1, 0, 1, 2, 3\}$ where -1 means the relating the concepts is a *misconception*, a 0 means the concepts are *unrelated* and the strength of the relation increases up to 3 which is a *strong relation*.

Assigning the actual point values for the propositions in the concept map was inspired by previous work in the literature. For Unit 2 the inspiration came from Novak and Gowin's 1984 work [14]. In it, they show a scoring method that grants 1 point for valid relationships, 5 points for a valid hierarchy, and 10 points for each 'crosslink' or proposition constructed of concepts from different hierarchical branches. They also score 1 point for each example a student correctly lists. In our work, we don't allow students to add concepts, and we aren't able to track hierarchies. Therefore, we say 'inspired by' this scoring method because we grant 1 point for *weakly related* concepts, 5 points for *related* concepts, and 10 for *strongly related*, 0 points for *unrelated* concepts and -1 for *misconceptions*.

An idea of the scoring used by Rice [28] was that students should be penalized for missing what was important. This lead us to try a different points assignment for School B. Concept maps from this school scored 1 point for any positive relationship, and -1 for *unrelated* concepts or *misconceptions*.

In all cases, the linking words in the propositions are not considered for scoring.

4 Results and Discussion

The results are shown as scatterplots in Figure 3 and numerically in Table 3. There is evidence of correlation in most of the cases. For School A, the Personal Web Page project correlated with the concept map task much better than the unit test, which doesn't have much evidence of a correlation. In School B, both the unit test and the vocabulary test had good evidence of a correlation.

There is good evidence that the concept map task can offer similar outcomes to a traditional test or a class project. There could be several reasons for the unit test at School A not being as strongly correlated with the concept map

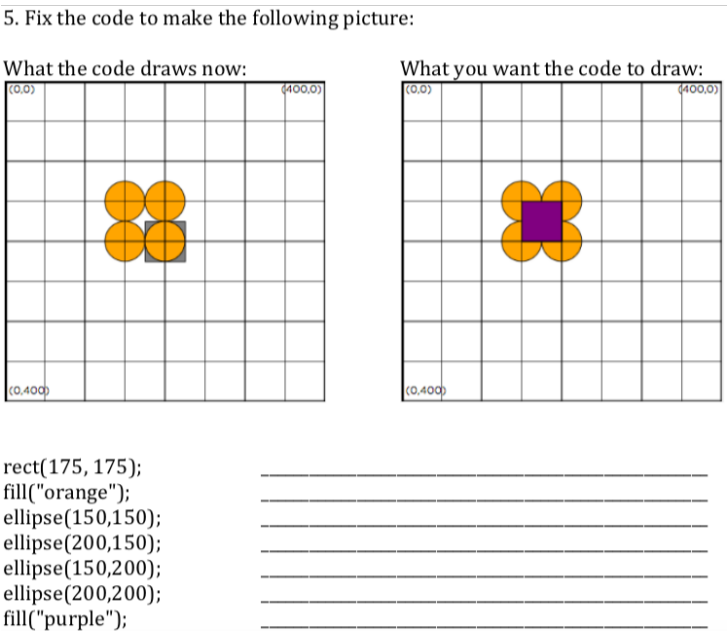


Figure 2: This is an example question from the Unit 3 Test.

score. It was the first time through a new curriculum for an inexperienced CS teacher and the unit test was made by someone outside the classroom. It is entirely possible that the students were not prepared for the kinds of questions the test posed. Asking students to provide HTML code directly from memory during the test is a more rigorous test than the one offered in the Code.org updated curriculum. The students were able to build web pages in class, so it is our conclusion that students could reference key words while building their sites, but likely didn't have such a reference available when taking the test.

Students are not currently familiar with concept maps or how to construct them. The classroom teachers in this study were shown how to use the online tool to create their maps and also instructed their students. Adding concept map creation as part of the curriculum would help students to be more comfortable making maps.

4.1 Threats to Validity

Any time a study is done with students and teachers taking measurements with assessments there are threats to validity. The questions we want to answer when addressing this topic are: 1) Did we measure what we intended to

Table 3: Summary of the Correlation Statistics for both Schools and both assessments, showing the correlation coefficient and P-Value for all cases.

School	Test	Coefficient	P-value
A	Unit Test	0.14354681	0.209911893
A	PWP	0.5181389	$7.070\,608 \times 10^{-5}$
B	Unit Test	0.3796854	$2.377\,383 \times 10^{-7}$
B	Vocab.	0.3998293	$9.166\,005 \times 10^{-9}$

measure? and 2) What does the numerical score mean? There is also the question of what to do with the results of the assessment.

What the concept map task actually measures is not precisely known. This study would be only the first of a series of studies to establish what is measured with some certainty. The positive correlation with tests and other assessments provides some evidence that the concept map task measures student knowledge and understanding about the CS Discoveries Units. There is also a case to be made that the concept map task also measures the amount of effort a student put forth in the task. The more propositions made, the higher the score in general. Since more than half of the possible propositions result in no points or negative points, this is not simply a measure of effort exclusively.

The numerical scores themselves provide an indication of the number and quality of propositions that can be stated by a student relative to his peers. In all cases in this study, the higher the score the better. It doesn't stand to reason that a certain number has any specific meaning or that a threshold could be established to claim a certain level of understanding. The study was not constructed to be able to make these kinds of statements. This is the main reason why this work shows that the concept map task would make a good supplement to other forms of assessment.

The unit tests themselves were constructed to be valid assessments. However, they would need to be tested carefully in several studies to be able to make such a claim. There is some evidence from their construction and also from the scatterplots in Figure 4 that the students performed comparably on the teachers' assessments as on the tests.

4.2 Potential Benefits to Alternative Assessments

There are several potential benefits to using the concept map tools and assessments in the classroom. Students who are learning English or who have difficulty with language can benefit from this task because required language

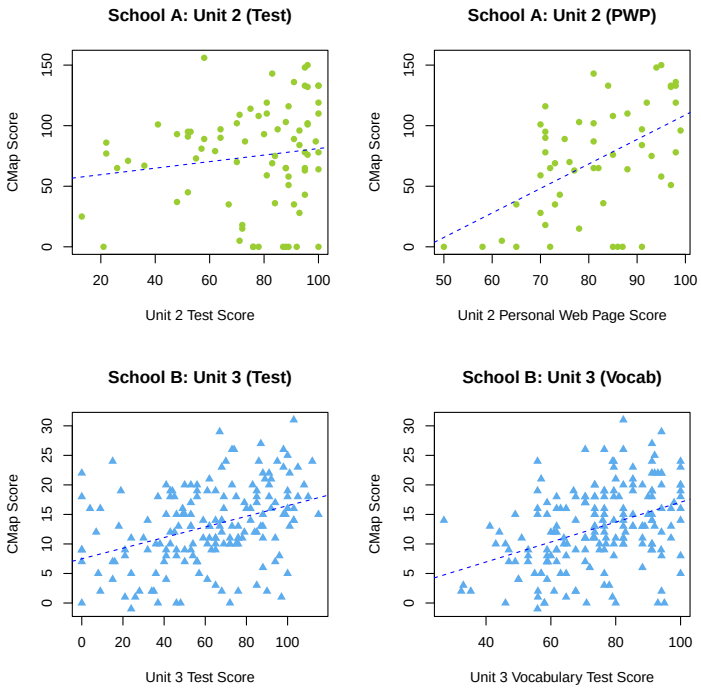


Figure 3: Scatterplots showing the scores students achieved in four different traditional assessments versus the scores they achieved on a concept map task. The concept map scores are shown along the y-axis and the different traditional test scores are across the x-axis.

skills are minimal. It is easy to subtitle or translate concepts.

The task can be administered entirely online. This makes the assessment distance learning friendly. In addition, it can be quite easy to detect students sharing answers because it is unlikely for two students to generate the same map. Dishonesty can be easily addressed by changing a few concepts for each map in the class, a process that itself can also be automated.

5 Conclusion and Future Work

This study addresses one of the chief concerns in adopting concept maps as an assessment, how to score them. We have used a computer program to score the maps and shown that a concept map task can give scores that correlate to

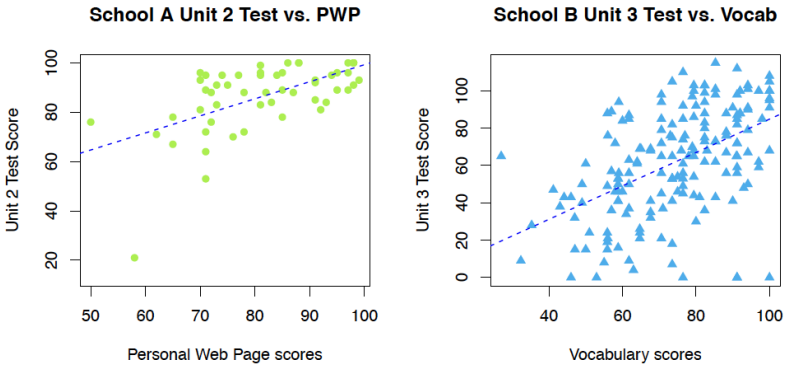


Figure 4: Scatterplots showing the scores students achieved in teacher made assessments versus the scores they achieved on the researcher made unit tests. The teacher test scores are across the x-axis and the unit test scores are shown along the y-axis.

student performance on other assessments.

It is our recommendation that these assessments be used as an alternative and a complement to other kinds of assessment. An inexperienced teacher can quickly and easily make, administer and score a concept map task. This can help ease the burden on already overworked teachers. An instructor with more experience can also use the maps themselves to find areas where students have misconceptions about how topics and concepts are related. These function essentially as a knowledge inventory and can offer insight into student understanding.

Future work includes more studies to help establish the validity of the concept map task and scoring approach. The scoring schemes could be varied to uncover how much impact the assignment of points to concepts can have on the total map scores. Additionally, nothing was done with the linking words in the propositions in this study. The links could be studied for semantics and their sentiment analyzed and incorporated into the final score. A more sophisticated scoring algorithm could also be used to award points for groups of concepts that were emphasized in the lessons.

References

[1] William H Acton, Peder J Johnson, and Timothy E Goldsmith. "Structural knowledge assessment: Comparison of referent structures." In: *Journal of Educational Psychology* 86.2 (1994), p. 303.

[2] AS Awati and Arti Dixit. "Automated evaluation framework for student learning using concept maps". In: *International Journal of Advance Research, Ideas and Innovations in Technology* 3.1 (2017), pp. 452–461.

[3] Bryce Boe et al. "Hairball: Lint-inspired static analysis of scratch projects". In: *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, pp. 215–220.

[4] AJ Cañas, JD Novak, and J Vanhear. "Concept Mapping as an Assessment Tool in Science Education". In: *Proceedings of the Fifth International Conference on Concept Mapping* (2012).

[5] Alberto J Cañas, Joseph D Novak, and Priit Reiska. "How good is my concept map? Am I a good Cmapper?" In: *Knowledge Management & E-Learning: An International Journal* 7.1 (2015), pp. 6–19.

[6] Debbie Carter. *K–12 Computer Science Framework*. Retrieved from `http://www.k12cs.org`. 2016.

[7] Rodrigo Carvajal, Alberto J Cañas, et al. "Assessing concept maps: first impressions count". In: *San José, Costa Rica: Universidad de Costa Rica*. Citeseer. 2006.

[8] Zenobia CY Chan. "A qualitative study on using concept maps in problem-based learning". In: *Nurse education in practice* 24 (2017), pp. 70–76.

[9] Kuo-En Chang et al. "A new assessment for computer-based concept mapping". In: *Journal of Educational Technology & Society* 8.3 (2005).

[10] Code.org. *CS Discoveries curriculum*. 2022. URL: `https://code.org/educate/csd/`.

[11] Computer Science Teachers Association. *CSTA K-12 Computer Science Standards, Revised 2017*. `http://www.csteachers.org/standards`. 2017.

[12] Gabriel De Pace. "Concept Maps for Middle School Computer Science Assessment". PhD thesis. University of Rhode Island, 2022.

[13] Evangelia Gouli et al. "COMPASS: an adaptive web-based concept map assessment tool". In: *Proceedings of the first international conference on concept mapping* (2004).

[14] D Bob Gowin and Joseph D Novak. "Learning how to learn". In: *USA: Cambridge University* (1984).

[15] Scott H Harrison et al. "C-TOOLS Automated Grading for Online Concept Maps Works Well with a Little Help from" WordNet". In: *Concept Maps: Theory, Methodology, Technology. Proc. of the First Int. Conference on Concept Mapping*. Vol. 2. 2004, pp. 211–214.

[16] Howard E Herl et al. “Feasibility of an On-line Concept Mapping Construction and Scoring System.” In: *Annual Meeting of the American Educational Research Association* (1997).

[17] Tsukasa Hirashima et al. “Framework of kit-build concept map for automatic diagnosis and its preliminary use”. In: *Research and Practice in Technology Enhanced Learning* 10.1 (2015), pp. 1–21.

[18] Gwo-Jen Hwang, Po-Han Wu, and Hui-Ru Ke. “An interactive concept map approach to supporting mobile learning activities for natural science courses”. In: *Computers & education* 57.4 (2011), pp. 2272–2280.

[19] Jeroen Keppens and David Hay. “Concept map assessment for teaching computer programming”. In: *Computer Science Education* 18.1 (2008), pp. 31–42. DOI: 10.1080/08993400701864880. eprint: http://dx.doi.org/10.1080/08993400701864880. URL: http://dx.doi.org/10.1080/08993400701864880.

[20] Ian M Kinchin, David B Hay, and Alan Adams. “How a qualitative approach to concept map analysis can be used to aid learning by illustrating patterns of conceptual development”. In: *Educational research* 42.1 (2000), pp. 43–57.

[21] Xiufeng Liu. “The Validity and Reliability of Concept Mapping as an Alternative Science Assessment when Item Response Theory Is Used for Scoring”. In: *Annual Meeting of the American Educational Research Association* (1994).

[22] Kimberly M. Markham, Joel J. Mintzes, and M. Gail Jones. “The concept map as a research and evaluation tool: Further evidence of validity”. In: *Journal of Research in Science Teaching* 31.1 (1994), pp. 91–101. ISSN: 1098-2736. DOI: 10.1002/tea.3660310109. URL: http://dx.doi.org/10.1002/tea.3660310109.

[23] Norma L Miller and Alberto J Cañas. “A semantic scoring rubric for concept maps: design and reliability”. In: *Proceedings of the Third International Conference on Concept Mapping* (2008).

[24] Andreas Mühling. “Aggregating concept map data to investigate the knowledge of beginning CS students”. In: *Computer Science Education* 26.2-3 (2016), pp. 176–191. DOI: 10.1080/08993408.2016.1241340. eprint: http://dx.doi.org/10.1080/08993408.2016.1241340. URL: http://dx.doi.org/10.1080/08993408.2016.1241340.

[25] J. D. Novak and A. J. Cañas. “The Theory Underlying Concept Maps and How to Construct and Use Them”. In: *Technical Report IHMC CmapTools* (2008).

[26] Joseph D Novak et al. “Cmapanalysis: an extensible concept map analysis tool”. In: *Journal for Educators, Teachers and Trainers* 4.1 (2013), pp. 36–46.

[27] Joseph D. Novak. “Results and Implications of a 12-Year Longitudinal Study of Science Concept Learning”. In: *Research in Science Education* 35.1 (2005), pp. 23–40. ISSN: 1573-1898. DOI: 10.1007/s11165-004-3431-4. URL: http://dx.doi.org/10.1007/s11165-004-3431-4.

[28] Diana C Rice, Joseph M Ryan, and Sara M Samson. "Using concept maps to assess student learning in the science classroom: Must different methods compete?" In: *Journal of Research in Science teaching* 35.10 (1998), pp. 1103–1127.

[29] Maria Araceli Ruiz-Primo and Richard J. Shavelson. "Problems and Issues in the Use of Concept Maps in Science Assessment". In: *Journal of Research in Science Teaching* 33.6 (1996), pp. 569–600. DOI: 0022-4308/96/060569-32.

[30] Jean Salac et al. "Patterns in elementary-age student responses to personalized & generic code comprehension questions". In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 2020, pp. 514–520.

[31] Candace Schau et al. "Select-and-Fill-in Concept Map Scores as a Measure of Students' Connected Understanding of Science". In: *Educational and Psychological Measurement* 61.1 (2001), pp. 136–158. URL: http://dx.doi.org/10.1177/00131640121971112.

[32] Jean Marie Schultz. "Mapping and Cognitive Development in the Teaching of Foreign Language Writing". English. In: *French Review* 64.6 (May 1991), pp. 978–88. ISSN: 0016-111X.

[33] Maija Strautmane. "Concept map-based knowledge assessment tasks and their scoring criteria: An overview". In: *Concept maps: Theory, methodology, technology. Proceedings of the fifth international conference on concept mapping*. Vol. 2. 2012, pp. 80–88.

[34] L Tomaswick and Jennifer Marcinkiewicz. "Active learning–concept maps". In: *Kent State University Center for Teaching and Learning* (2018).

[35] Mary Katherine Watson et al. "Assessing conceptual knowledge using three concept map scoring methods". In: *Journal of engineering education* 105.1 (2016), pp. 118–146.

[36] Po-Han Wu et al. "An innovative concept map approach for improving students' learning performance with an instant feedback mechanism". In: *British Journal of Educational Technology* 43.2 (2012), pp. 217–232. ISSN: 1467-8535. DOI: 10.1111/j.1467-8535.2010.01167.x. URL: http://dx.doi.org/10.1111/j.1467-8535.2010.01167.x.

[37] Aman Yadav et al. "Expanding computer science education in schools: understanding teacher experiences and challenges". In: *Computer Science Education* 26.4 (2016), pp. 235–254.

An Open-Source BinaryGame for Learning Reverse Engineering*

D'Angelo Gourdine and Suzanne J. Matthews
U.S. Military Academy
West Point, NY 10996
{dangelo.gourdine, suzanne.matthews}@westpoint.edu

Abstract

This paper introduces an open-source BinaryGame that assists students learning reverse engineering. The game consists of ten levels that increase in difficulty, help pages on GDB, and supports three flavors of assembly language. Work on the BinaryGame is ongoing; for our initial study, we used the BinaryGame to introduce students in a computer systems & organization course to Arm assembly. These students had prior knowledge of x64 assembly, but no prior knowledge of Arm assembly; our goal was to boost our students' confidence in learning unfamiliar assembly languages. Our results suggest that the BinaryGame increased student confidence in their a.) general reverse engineering abilities; b.) ability to reverse engineer programs in an unfamiliar assembly language, and c.) ability to reverse programs in Arm assembly. We believe that the BinaryGame can help students build their reverse engineering skillset.

1 Introduction

The BinaryGame is a open-source resource designed to help students learn reverse engineering at their own pace through an iterative, guided design. The notion of creating a game or other executable to teach reverse engineering has existed for nearly twenty years. The earliest known example is perhaps "Dr. Evil's Binary Bomb" (a.k.a., Bomblab) [5], initially developed by educators at

Carnegie Mellon University, and distributed as part of the seminal text, *Computer Systems: A Programmer's Perspective* [4]. The original Bomblab was an executable compiled for x86 systems and was used as part of classroom laboratory assignment at CMU. The bomb consists of six phases. Students have to enter the correct string for each phase; if the incorrect string is entered, the bomb "explodes" (e.g., prints out a string that says "Boom!"), causing a student to lose points. In the intervening years, many other bomb-like executables/games were developed [3, 13, 15] and used to teach basic reverse engineering concepts at conferences across the country, including Defcon and Black Hat, the premier conferences for hackers. The source code (and compiled binaries) to these custom re-implementations are typically not shared across institutions to reduce the likelihood that solutions "get out into the wild", either through code-sharing utilities (e.g., GitHub, Pastebin) or detailed walkthroughs on YouTube. The proliferation of online solutions prevents instructors from reusing (or sharing) otherwise well-designed assessments that took considerable time and effort to create.

A key novelty of our work is that the BinaryGame is open-source, allowing instructors to modify and reuse the source code as a teaching aid in their own courses. The BinaryGame is *not* designed to replace Bomblab-like assessments, but to *supplement* them. In their 2016 paper, Shashidhar and Cooper [15] advocate for the use of hands-on lab activities to "level the playing field" when teaching malware analysis, prior to introducing a more free-form "grand challenge" assignment. In their course, Shashidhar and Cooper use Portable Executable (PE) files to first introduce students to the principles of malware analysis in a laboratory setting. The Bomblab assignment is then discussed as a potential "grand challenge" assignment to provide a "less structured learning environment... where leadership and direction comes from the student as much as the instructor" [15]. Our creation of the BinaryGame supports the model curriculum laid out by Shashidhar and Cooper by freely providing a hands-on lab activity to introduce students to reverse engineering prior to the introduction to a larger "grand challenge" assignment.

Wile the literature strongly supports the use of simple exercises to build reverse engineering skills, there are very few readily-available resources for instructors to use in their courses. Shashidhar and Cooper [15] do not describe nor provide any of the lab activities they use. Aycock et al. [1] discuss a series of exercises they developed to teach reverse engineering, though not with sufficient data to fully reproduce all of them. While Stricklan and OConnor [16] discuss a framework for creating "diversified challenges" for a reverse engineering course, the challenges themselves are not provided. Like Aycock et al. [1], the BinaryGame contains levels that build on each other and increase in difficulty as a user advances through the game. Like the method discussed by

Stricklan and OConnor [16], the BinaryGame includes options for compiling to distinct architectures and enables the generation of multiple unique binary executables. Like Bomblab, the goal for each level is to input a string that will allow the user "pass" to the next level. Unlike prior work, the BinaryGame also includes introductory guides on GDB and different flavors of assembly, and suggests readings in the free online textbook, *Dive into Systems* [12].

Work on the BinaryGame is ongoing and community contributions are welcome. For our initial evaluation of the BinaryGame, we focus specifically on its ability to introduce students to an unfamiliar assembly language, namely Arm assembly. The Arm architecture is one of the fastest growing and widely used computer architectures today; as of 2021, over 200 billion chips using the Arm architecture are used in a variety of mobile devices [14]. Unlike Stricklan and OConner [16] who use Qemu [2] to generate Arm binary executables, we deploy the BinaryGame on a Raspberry Pi. Our decision to use the Raspberry Pi in lieu of a hardware emulator was manifold: first, prior work strongly supports the use of the Raspberry Pi to teach cybersecurity concepts [6, 17, 7, 8]. Next, researchers have also used the Raspberry Pi to teach topics like C programming [18], Arm assembly [9, 10], and parallel programming [10], all which are common topics in a computer systems course, a natural place to use the BinaryGame. Lastly, using the Raspberry Pi allowed us to hide the code to generate new binaries on the Pi's file system, enabling the user to generate novel, playable binaries of the BinaryGame at will on their Pis.

2 Overview of Game

The BinaryGame includes ten levels that increase in difficult and scope, modeled after concepts covered in each of the first nine sections of the Assembly chapters in the free online textbook, *Dive into Systems* [11, 12]. The levels are: (0) predefined C functions; (1) basic string matching; (2,3) arithmetic operations; (4,5) loops; (6) recursion; (7) arrays; (8)matrices; and (9) structs. The goal of each level is to identify a unique string that will enable the user to continue on the following level; there is no penalty for incorrect answers. In the event that students are unable to pass the level and need assistance, they have the option to receive a hint, which encourage students to refer to specific portions of *Dive into Systems* and provide other clues for solving the level.

The BinaryGame is designed to be used in conjunction with the GNU Debugger (GDB), which typically represents a steep learning curve for students. As a result, the BinaryGame includes several help pages that provide an entry point to GDB and basic assembly concepts. In addition to GDB, there are separate help pages that provide basic information on Arm assembly, IA32 assembly, x86-64 assembly and a general introduction to the game.

The BinaryGame is written in C and includes a Python wrapper that generates multiple version of the binary executable. Giving students access to this variability allows them to harden their skills through increased repetitions. For the purposes of this study, the BinaryGame was compiled directly on a Raspberry Pi 4B; we have verified that the BinaryGame works with the Raspberry Pi 3B+ and 3B. We note that a 64-bit version of the operating system is preferable to ensure the assembly matches the coverage of *Dive into Systems*. To prepare the devices for classroom use, the instructor flashed a number of Raspberry Pis with an image that had the BinaryGame source code hidden in a directory on the file system. We wrote a custom shell script (called `refesh-game`) that generates a new version of the Binary Game executable, and replaces the BinaryGame directory in the user's home directory with this new version, deleting all solution files that students may have created in that directory. This allows the instructor to "refresh" the game directory between labs without reflashing the Pis. Instructors can access the BinaryGame source at: `https://github.com/suzannejmatthews/binaryGame`.

3 Methods

The surveyed students were primarily juniors majoring in computer science or cyber security. All students were enrolled in a required computer systems & organization course at West Point, which is the first course that students are introduced to reverse engineering. The BinaryGame was deployed as a lab exercise on Arm assembly fairly late in the semester. While the lab represents students' first exposure to Arm assembly and the Raspberry Pi, all students had previously completed a 10-lesson unit modeled after the first nine sections of the x86-64 assembly chapter in *Dive into Systems*. Our goal in having an Arm assembly lab in a course that otherwise covers x86 assembly was to build our students' confidence that their reverse engineering skills can translate to an ISA that they are unfamiliar with. We concentrated on two main research questions. First, RQ1: "Does the BinaryGame improve student confidence in their reverse engineering skills?". Second, RQ2: "Does BinaryGame improve student motivation to continue learning reverse engineering?"

To analyze the effect that the BinaryGame had on student's confidence in their reverse engineering skills, we asked participants to rate their current level of confidence on a Likert scale on three topics related to reverse engineering on the pre- and post-surveys:

- C1: Confidence reverse engineering programs in some ISA (e.g. x64)
- C2: Confidence learning an unfamiliar assembly language
- C3: Confidence reverse engineering programs in Arm assembly

Here, a score of 1 represented "very unconfident", 2 represented "somewhat unconfident", 3 represented "neither confident/unconfident", 4 represented "somewhat confident", and 5 represented "very confident".

To assess the impact that BinaryGame had on students' motivation to continue learning reverse engineering, we asked students to rate their current level of motivation on Likert scale on the pre- and post-surveys. For this question, a score of "1 represented ("very unmotivated") to 5 ("very motivated") to learn reverse engineering" on the pre- and post-surveys.

Since this lab was our students first exposure to the Raspberry Pi, we spent a few minutes at the beginning of the lab period discussing the ubiquity of Arm processors in mobile and embedded computing and discussed how malware may increasingly target the architecture in the future. We also discussed how cyber professionals need to be comfortable switching between multiple instruction set architectures, and how this lab was designed to help them gain confidence that they can quickly "pick up" a new ISA (namely Arm). After giving a quick overview of the features of the BinaryGame, the students were given the rest of the lab period to play through the game on their own. At the end of the lab period, we asked them to complete the post-survey which contained identical questions to the pre-survey, plus additional questions on various aspects of the game. We use a paired sample student t-test to measure the significance of the difference of the means of each population. For each question, we reject the null hypothesis when the p-value is less than 0.05. No identifiable data was collected on students; however, the pre- and post-surveys for students were distributed together to enable us to correlate student responses.

4 Results

We surveyed students over two semesters of the computer systems & organization course, which together had an enrollment of 58 students. Of these 58 students, only 33 students (56.9%) opted to participate. Of the 33 students, one student failed to complete the pre-survey and another failed to complete the post-survey. To ensure we could conduct a paired sample t-test between the two populations, we omitted these two students from our analysis. As a result, there are 31 students ($n = 31$) in our final analysis. We used the `ttest_rel` statistic in the Python SciPy stats module for the paired difference test.

4.1 Effect on Student Confidence

For RQ1, our null hypothesis is that students would not experience a significant improvement in their level of confidence in their reverse engineering skills. We reject the null hypothesis when the p-value is below 0.05 for each confidence-related topic. Table 1 depicts the mean scores and corresponding p-values of

Table 1: Summary of Confidence Results

Question	Pre-Survey $n = 31$	Post-Survey $n = 31$	p-value
C1	2.71	3.452	4.519×10^{-06}
C2	2.581	3.29	3.181×10^{-06}
C3	1.823	3.323	8.053×10^{-09}

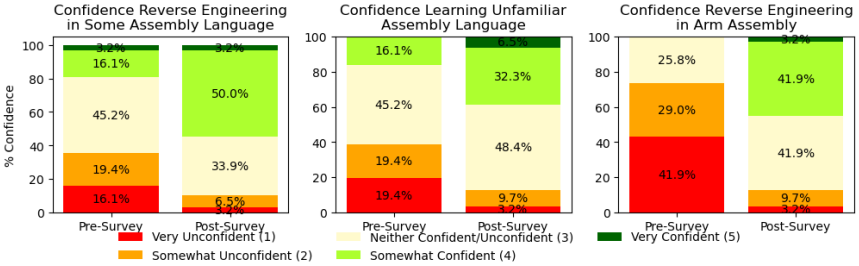


Figure 1: Distribution of Student Confidence on Pre- and Post-Surveys.

our three topics assessing student confidence related to reverse engineering. Figure 1 illustrates the distribution of scores from the pre- and post-surveys. Our data strongly suggests that our students experienced a statistically significant increase in confidence in in their ability to reverse engineer programs in some ISA, learn an unfamiliar assembly language, and reverse engineer programs in Arm assembly.

For example, only 19.3% of respondents expressed some level of confidence that they could reverse engineer in some assembly language on the pre-surveys, with a large number of students (45.2%) feeling neither confident or unconfident. In the post-surveys, the majority of students (53.2%) indicated some level of confidence in reverse engineering in some assembly language, with only 33.9% feeling neither confident or unconfident.

Similarly, students experienced a statistically significant increase in confidence in reverse engineering in an unfamiliar assembly language (C2). In the pre-survey, only 16.1% of students expressed some level of confidence that they could reverse engineering in an unfamiliar assembly language, while 38.8% expressed some level fo confidence in the post-survey. However, a large percentage of students expressed that they were neither confident or unconfident on both surveys. Perhaps unsurprisingly, students experienced the largest increase in confidence in reverse engineering in Arm assembly (C3). We do note that while a large number of students (45.1%) expressed confidence in the post-survey, a

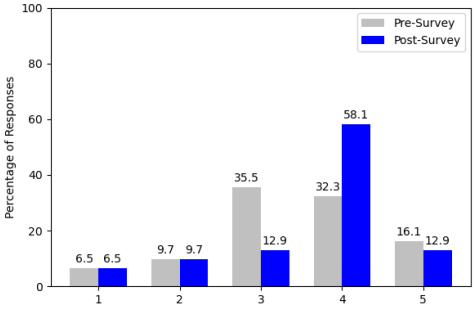


Figure 2: Distribution of Student Motivation on Pre- and Post-Surveys.

significant number (41.9%) still expressed ambivalence on the post-survey, with 12.9% expressing a lack of confidence.

While the increase in confidence across C3 is perhaps expected, the increase in confidence in "reverse engineering in some assembly language" (C1) is interesting. First, while our students had prior exposure to an in-course 10-lesson unit on reverse engineering in x86-64 prior to taking the Arm lab, nearly 35.5% of the course expressed that they felt some level of confidence about reverse engineering in some assembly language on the pre-survey. While the BinaryGame lab covered Arm reverse engineering, their overall confidence in reverse engineering in *some* assembly language increased, suggesting that process of using tools like GDB (even in the context of an unfamiliar assembly language) helped them improve their general skills in reverse engineering. Given our results, we reject the null hypothesis, and conclude that the BinaryGame had a statistically positive impact on the surveyed students' confidence levels in their reverse engineering abilities.

4.2 Effect on Student Motivation

Lastly, our pre- and post-game surveys asked students to assess their motivation for learning reverse engineering. For RQ2, our null hypothesis is that students would not experience a significant change in motivation in learning reverse engineering as a result of the BinaryGame lab. We reject the null hypothesis when the p-value is below 0.05.

Figure 2 shows the distribution of scores on the pre- and post-surveys. The y-axis indicates the percentage of students who indicated that they were: 1 ("very unmotivated"), 2 ("somewhat unmotivated"), 3 ("neither unmotivated or motivated"), 4 ("somewhat motivated") or 5 ("very motivated") to learn reverse engineering on the pre- and post-surveys. The pre-survey average was

3.419, while the post-survey average was 3.613. While 48.4% of students indicated that they were motivated to learn reverse engineering on the pre-survey, 71% of the students indicated some level of motivation on the post-survey. While students did experience a modest increase in motivation for learning reverse engineering between the pre- and post-surveys, the difference between the means was not statistically significant ($p = 0.2805$). Therefore, we accept the null hypothesis for RQ2.

We asked some additional questions of students on the post-survey to gauge the perceived usefulness of the BinaryGame to learn reverse engineering. While 81.27% of our students rated the game as being "useful" or "very useful" for learning Arm assembly, an additional 12.5% felt neutral about the game. Students who liked the game mentioned that the *"difficulty scales well"*, and that it was a *"great start"* for learning reverse engineering in Arm assembly. However, students also noted that *"the levels past level 3 felt almost the same"*, with one student complaining that problems could be easily solved by *"putting a break-point at the cmp instructions and reading out what the inputs had to be"* We feel this is valid criticism; it *is* hard to create meaningful reverse engineering exercises. However, we feel heartened that many students appreciated the level of difficulty.

5 Conclusion

This paper focused on introducing an unfamiliar (Arm) assembly language to a population of students who were already exposed to reverse engineering concepts. Our results suggest that the BinaryGame not only helps increase student confidence in using Arm assembly, but in their existing skillset as well; by seeing the same concepts applied to a novel architecture, students gained confidence in their general abilities to reverse engineer.

Future assessment will focus on the effectiveness of the BinaryGame to introduce reverse engineering concepts to students who are largely unfamiliar with any assembly language, and to include larger populations of students. The BinaryGame is still being actively developed, and we continue to refine the complexity of levels and modalities for deploying the game. Community contributions are welcome!

6 Acknowledgement

This project is supported by the National Science Foundation (DUE - 2141814). The views expressed in this article are those of the author and do not reflect the official policy or position of the Department of the Army, Department of Defense or the U.S. Government.

References

[1] John Aycock et al. "Exercises for Teaching Reverse Engineering". In: *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE 2018. Larnaca, Cyprus: Association for Computing Machinery, 2018, pp. 188–193. ISBN: 9781450357074. DOI: 10.1145/3197091.3197111. URL: https://doi.org/10.1145/3197091.3197111.

[2] Fabrice Bellard. "QEMU, a fast and portable dynamic translator." In: *USENIX annual technical conference, FREENIX Track*. Vol. 41. 46. California, USA. 2005, pp. 10–5555.

[3] Logan Brown, Gavin Hayes, and Tejas Rao. "Reinventing Bomblab". Major Qualifying Project. Department of Computer Science, Worcester, PA, USA: Worcester Polytechnic Institute, 2017. URL: https://digital.wpi.edu/downloads/s7526g02j4.

[4] Randal E Bryant, O'Hallaron David Richard, and O'Hallaron David Richard. *Computer systems: a programmer's perspective*. Vol. 2. Upper Saddle River, NJ USA: Prentice Hall, 2003.

[5] Randal E Bryant and David R O'Hallaron. "Introducing computer systems from a programmer's perspective". In: *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*. 2001, pp. 90–94.

[6] Andreea Cotoranu and Li-Chiou Chen. "Using Raspberry Pi as a Platform for Teaching Cybersecurity Concepts". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 2019, pp. 1237–1237.

[7] Sandra Gorka et al. "Tempting High School Students into Cybersecurity with a Slice of Raspberry Pi". In: *Journal of The Colloquium for Information Systems Security Education*. Vol. 8. 1. 2020, pp. 4–4.

[8] Stylianos Karagiannis et al. "PocketCTF: A fully featured approach for hosting portable attack and defense cybersecurity exercises". In: *Information* 12.8 (2021), p. 318.

[9] Jalal Kawash et al. "Undergraduate Assembly Language Instruction Sweetened with the Raspberry Pi". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE '16. Memphis, Tennessee, USA: Association for Computing Machinery, 2016, pp. 498–503. ISBN: 9781450336857. DOI: 10.1145/2839509.2844552. URL: https://doi.org/10.1145/2839509.2844552.

[10] Benjamin Levandowski, Debbie Perouli, and Dennis Brylow. "Using embedded xinu and the raspberry pi 3 to teach parallel computing in assembly programming". In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2019, pp. 334–341.

[11] Suzanne J. Matthews, Tia Newhall, and Kevin C. Webb. *Dive into Systems*. 1st ed. Free version available at: http://www.diveintosystems.org. San Francisco, CA: No Starch Press, June 2022. ISBN: 978-1718501362.

[12] Suzanne J. Matthews, Tia Newhall, and Kevin C. Webb. "Dive into Systems: A Free, Online Textbook for Introducing Computer Systems". In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. SIGCSE '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 1110–1116. ISBN: 9781450380621. DOI: 10.1145/3408877.3432514. URL: https://doi.org/10.1145/3408877.3432514.

[13] Elizabeth Patitsas and Daniel Levy. "Dr. Horrible's Fork Bomb: A Lab for Introducing Security Issues in CS2". In: *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '13. Canterbury, England, UK: Association for Computing Machinery, 2013, p. 318. ISBN: 9781450320788. DOI: 10.1145/2462476.2465577. URL: https://doi.org/10.1145/2462476.2465577.

[14] Simon Segars. *Arm Partners Have Shipped 200 Billion Chips*. Oct. 2021. URL: https://www.arm.com/blogs/blueprint/200bn-Arm-chips.

[15] Narasimha Shashidhar and Peter Cooper. "Teaching malware analysis: The design philosophy of a model curriculum". In: *2016 4th International Symposium on Digital Forensic and Security (ISDFS)*. 2016, pp. 119–125. DOI: 10.1109/ISDFS.2016.7473529.

[16] Christopher Stricklan and TJ OConnor. "Towards Binary Diversified Challenges For A Hands-On Reverse Engineering Course". In: *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE '21. Virtual Event, Germany: Association for Computing Machinery, 2021, pp. 102–107. ISBN: 9781450382144. DOI: 10.1145/3430665.3456358. URL: https://doi.org/10.1145/3430665.3456358.

[17] Adam H Villa. "Hands-on computer security with a Raspberry Pi". In: *Journal of Computing Sciences in Colleges* 31.6 (2016), pp. 4–10.

[18] Michael Wirth and Judi McCuaig. "Making Programs With The Raspberry Pi". In: *Proceedings of the Western Canadian Conference on Computing Education*. WCCCE '14. Richmond, BC, Canada: Association for Computing Machinery, 2014. ISBN: 9781450328999. DOI: 10.1145/2597959.2597970. URL: https://doi.org/10.1145/2597959.2597970.

Incorporating a Makerspace into Computer Science Courses*

Michael B. Gousie
Department of Computer Science
Wheaton College
Norton, MA 02766
mgousie@wheatoncollege.edu

Abstract

We describe the use of the Wheaton College's Makerspace through three example projects in three different courses: Computer Organization and Assembly Language, Theory of Computation, and a student-defined Independent Study. The two regular courses show how a Makerspace project can be incorporated into a traditional computer science class, while the Independent Study shows how a group of computer science students can use such resources to facilitate their own service-learning projects. In each case, the projects enhance traditional course skills by adding real-world and/or interdisciplinary components.

1 Introduction

Makerspaces are becoming ubiquitous on college campuses, with many promises of more student collaboration, building fabrication skills, experiential learning, and more. However, as Rogers asked in [18], what do we make of Makerspaces as they pertain to post-secondary computer science education? Beyond the building of digital circuits (which may occur in purpose-built labs rather than general Makerspaces), there seems to be little connection between computer science and such collaborative spaces. In this paper, we discuss how a Makerspace can be used in conjunction with three different courses: Computer

Organization and Assembly Language, Theory of Computation, and an Independent Study based on a student-defined project. In each, the use of the Makerspace is for a completely different purpose, even as some of the tools are used in common across the multiple courses.

2 Use of Makerspaces

The use of Makerspaces in various fields in higher education, such as design, engineering, physics, and others, is generally regarded as a positive development in fostering student collaboration, communication, critical thinking, and developing fabrication skills. Furthermore, students can get real-world, hands-on experience in addition to theoretical concepts learned in the classroom [16] However, there is little actual data to support the idea that Makerspaces are fulfilling this mission, although attempts have been made [3, 5]. Adler-Beléndez *et al.* provide a comprehensive literature review on assessment issues [1].

There is a dearth of literature on the use of Makerspaces as it pertains to courses in computer science specifically. The one common exception to this is the building of physical circuits, which often takes place in a lab run by the physics department or program. However, those that do not have such a lab may incorporate this functionality in a Makerspace [4]. Many have looked at how to design functional spaces, using traditional technologies [15] or developing special-purpose hardware and software, as in the Smart Makerspace [12], among others. Still other projects determine how to foster and/or test the efficacy of Makerspaces for supporting student collaboration [2, 9].

Another large area of research centers around the use of Makerspaces for K-12 education. One such example attempts to assess the learning that develops from the use of Makerspaces incorporated in STEM fields [6]. In [13], they describe a program to educate teachers to be more effective in the design space. Scheppegrell *et al.* [19] designed lessons integrating a Makerspace for second grade children to foster computational thinking; a similar idea applied to an after-school program is reported in [21]. Yet another program built a Makerspace on a college campus for use by K-12 students, especially from underserved populations [11].

3 Examples and Results of Incorporating a Makerspace into Computer Science Courses

The Makerspace at Wheaton is housed in a purpose-built space in a newly renovated science building. It contains such equipment as a small milling machine, several FDM (Fused Deposition Modeling) and SLA (StereoLithography Apparatus) 3-D printers, a larger laser cutter, large, flat work tables, and assorted

hand tools. Next door is the machine shop housing larger equipment such as 3- and 5-axis CNC mills, a lathe, and assorted drill presses and saws. Any student on campus has access to these spaces and equipment during regular open hours, supervised by a full-time director.

We now describe each of the three computer science courses and how the Makerspace was used in each. The resulting output and/or results are also discussed in each section.

3.1 Computer Organization

A required course for all computer science majors is Computer Organization and Assembly Language. Among other things, students learn about the datapath, the route that instructions take from first being read from memory to their execution. This is done at the bit level, with many components shown as "black boxes" that perform the desired operation (for example, the program counter). Thus, the students are attempting to learn the datapath in a sort of "virtual" way; they are not using actual wires and electronic circuits. Even in that more realistic realm, there is no way to "see" the zeros and ones "moving" from one component to another. One way to help students understand the datapath is to use a circuit simulator. In such a system, the student can see how wires/circuits change from 1 to 0 and vice-versa through the use of coloring of the components. To further strengthen the students' grasp of data moving through the datapath, we hold the annual *Comp Org Crazy Model Expo* [8], a somewhat whimsical but nevertheless helpful way for students to visualize what is happening at the hardware level within circuits. The goal of the *Expo* is for pairs of students to build a model of some portion of the datapath without using any electronics, the idea being that they should show how the data flows through and is changed by various components. Students do this by modeling 0's and 1's as marbles, Hot Wheels cars, water, etc. In addition, they must create a poster describing exactly how their model mimics the real circuit(s). While the project can be a bit whimsical, the poster should be of a high standard as if presenting at a conference. Finally, the students present their model to the class during the *Expo*. The class votes on the best presentation/model, and the winners receive prizes such as stickers and squishy toys. The competition can be quite fierce!

After a group's idea is approved by the instructor, they have about two weeks to complete their project and poster. There are numerous good examples of these projects running the gamut from building with cardboard and tape to 3-D printing. Students have access to the college's Makerspace whenever there are open hours. A full-time staff member is present at these times to help with student projects. The Makerspace is also available for regularly scheduled classes, some of which are geared especially to "making."

Figure 1: Students presenting a wooden binary-to-decimal converter.

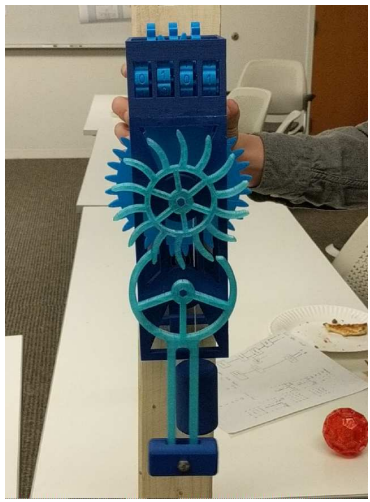


Figure 2: A working 3-D printed binary counter.

For but one example, students created a 6-bit binary-to-decimal converter. They built a wooden ramp with six tubes that held from one to 32 marbles, respectively. Each tube represents a bit position in the 6-bit value, or 2^0 to 2^5. At the start, each tube is "closed," representing zero at each bit position and thus zero for the represented value. If the "valve" for one of the tubes is opened, that represents a one at that bit position, and the appropriate number of marbles roles down into a long tube. If the fifth position is opened, for example, 16 marbles would roll out, representing 2^4. Thus, the correct number of marbles will roll out for any combination of valves (bit positions) that are opened. This is shown in Figure 1. The Makerspace was used to cut and glue the wood, as well as cut the tubing. Our Makerspace has scraps of spare wood available for free that was sufficient for a project of this size.

Another example is shown in Figure 2, a *working* 3-D printed model of a binary counter. Once the pendulum is set in motion, it meshes with the external gear which, in turn, turns an internal gear that moves the four binary counter wheels. Our Makerspace has several 3-D printers available, including free materials.

Figure 3 shows a student explaining his group's project via their poster. They built a working binary adder comprised of two half-adders. Their model was rudimentary, made of cardboard, but the thought put into it is the same no

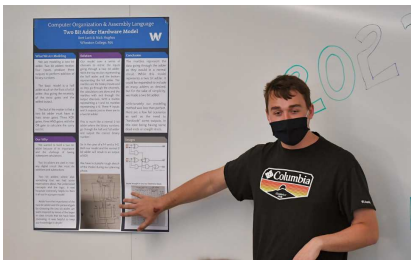


Figure 3: A student explaining a project using his group's poster.

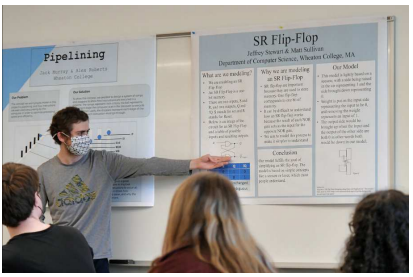


Figure 4: Another example of a poster, this time showing a flip-flop.

matter what the materials are. Even using simple materials, the Makerspace gives students a great space with large flat tables to work on their models. On the other hand, another group built a working model of an SR flip-flop out of sheet metal, using the tools available in the Makerspace. The model had a lever such that it forced one output to represent "1" and the other to represent "0" or vice-versa. Figure 4 shows the poster and one of the presenters. Other common projects involve pipelining, incrementing the program counter, and modeling the workings of multiplexers.

3.2 Theory of Computation

Students need to write programs that are more extensive and use and/or generate more realistic data in upper-level computer science courses. In this project, students use a Bracketed L-System, a type of context-free grammar (CFG), to generate two- or three-dimensional virtual trees or plants [17]. Using such a grammar produces fractal, or self-similar, plant structures, a common example being a fern. The output of the program is a file that can be displayed graphically using any number of free viewers. Furthermore, the generated files can be used to create two-dimensional cutouts or etchings using a laser cutter, a piece of equipment that is becoming more widely available in colleges with the advent of Makerspaces. This project offers these experiential learning components that are not usually associated with a computer science theory course.

The project in more detail:

- Students define a Bracketed L-System grammar that can be used to generate a plant. For example, such a grammar might have the alphabet $= [S, B, x, y, \text{“[”}, \text{“]”}, +, -]$, where

S is the start symbol,

B is a non-terminal representing a branch. A branch can be thought of as a 3-D rectangle, or rectangular parallelepiped,

x, y are scale factors in the horizontal and vertical directions, respectively,

[means push the current position onto a stack,

] means pop the stack,

+ denotes a counter-clockwise rotation by a constant number of degrees,

− denotes a clockwise rotation by a constant number of degrees.

- Using the above grammar, a production rule(s), called the input string, is created:

$$B \rightarrow BB - [-yB + xB] + [+xB - yB]$$

In other words, this sample string defines a plant that begins with two straight branches, followed by a "subtree" which is rotated clockwise and consists of a branch to the right and one to the left, followed by another subtree that is rotated counter-clockwise in its entirety. This forms the basic outline of a plant. The degree of rotation and the scaling parameters can be changed to yield differently shaped plants with the same branch structure.

- Iterating over the production rule recursively "grows" the plant, yielding a larger and more filled out tree. Iterating the above rule 15 times produces the plant shown in Figure 5.

- This output is stored in the STL file format. This format stores 3-D objects as a set of planar triangles. Thus, the rectangular branches of the plant produced above are triangulated to be stored in this format.

  A small portion of an STL file that describes two triangles is as follows:

```
facet normal 0 0 0 outer loop
vertex 0.000000 5.000000 1
vertex 1.000000 5.000000 1
vertex 1.000000 10.000000 1
endloop endfacet
facet normal 0 0 0 outer loop
vertex 1.000000 10.000000 1
vertex 0.000000 10.000000 1
vertex 0.000000 5.000000 1
endloop endfacet
```

- The STL file can then be viewed on any number of free online viewers, such as `www.viewstl.com`.
- To create a laser-cut plant, the STL file must be converted to the Scalable Vector Graphics (SVG) format. Many online viewers can export an STL file to SVG; alternatively, online applications can convert an STL file to SVG. The SVG file can then be used as input to a laser cutter. For the tree above, Figure 6 shows the final product etched onto a piece of wood.

Following the steps above, students see how one can go from a virtual description of a plant to a standard data file to a physical model. The use of the Makerspace makes the entire process more "real" for the students, as they experience how a virtual description of a physical object can become a reality.

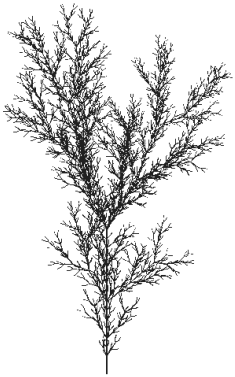


Figure 5: Tree defined by a Bracketed L-System grammar.

Figure 6: Tree etched onto a wood panel by a laser cutter.

3.3 Independent Study

The Independent Study course was taken by a group of three students. It came about as a (partial) solution to the problem of public access to art objects in possession of the college. As gallery space is severely limited, most of the art is in storage. This project endeavored to digitize 3-D art objects so as to display them via an interactive web page. As this was an interdisciplinary project, a professor in the Department of Art History was also involved in the course. While efforts to digitize art have been attempted before ([20, 7, 10, 14] to cite

just a few), most of the techniques require either expensive equipment or take an inordinate amount of time. In this project, both of those problems were mitigated.

The general steps of the implementation of the project were as follows, with notes as to what skills/equipment were needed:

1. Take video that captures all sides (except the bottom) of the object. (Smart phone, Makerspace, art handling and cataloging techniques)
2. Extract individual frames from the video file. (Programming)
3. Generate 3-D model using photogrammetry software. (Using sophisticated software)
4. Upload file to server and add link to web page. (Web design/development)

Step 1 above is often a bottleneck in the process. Here the Makerspace was used to build a turntable out of laser-cut wood and a 3-D printed housing and gear for a small motor. These pieces yielded a turntable as seen in Figure 7 such that the art object would be rotated at a constant speed while taking a video with a common smart phone. The combination of basic materials and the use of a regular phone kept the costs to approximately $60.

Figure 7: Turntable showing laser-cut wood gear and 3-D printed motor housing and gear.

While the other steps are beyond the scope of this paper, it is instructive to see how all of the interdisciplinary portions of the project use different skills. Specifically, computer science was broadly applied, from writing

a Python script for extracting frames (which reduced processing time immensely), to learning what photogrammetry is and how the software works, to writing HTML and CSS in a custom-designed web site that allows for the real-time rotation/scaling and viewing of a 3-D image, and finally to posting work on GitHub.

The results were rather extraordinary considering the minuscule cost of the project. Figure 8 shows a wood sculpture called Tree of Life. The 3-D rendering of the object is shown in Figure 9. The amount of detail is quite good for using an ordinary phone camera. This object, and others, can be seen on the web site at `http://cs.wheatoncollege.edu/whedomain/`.

Figure 8: A photo of the Tree of Life sculpture on the turntable.

Figure 9: The 3-D rendered version of the Tree of Life as shown on the web page.

4 Conclusions and Future Work

We have presented three scenarios in which a Makerspace plays a part in a computer science course. In the Computer Organization course, students use creativity and imagination to turning hardware concepts that can not be readily

"seen" into physical models. Even though these may be whimsical, students must still understand the real hardware in order to explain their model and present a professional-looking poster. In Theory of Computation, what seems like an abstract concept is turned into a model of a plant etched into a piece of wood. This shows that theoretical courses are not just that, but, through the use of real-world file formats, can create physical objects. Finally, various ideas in computer science can come together in student projects, as shown with an example of digitizing art objects. The students applied various skills in an interdisciplinary service project that helps the public access art that otherwise is not generally available to view. Not only did the students use the Makerspace, it was also true experiential learning as they had to master art handling techniques in a real-world setting. In all of these courses, the Makerspace played an essential role in turning virtual ideas into actual physical objects, or, in the case of the Independent Study, a physical object into a virtual one.

For future work, it would be instructive to determine if physical models such as created in the Computer Organization and Theory of Computation courses actually increase student understanding. As the same instructor has taught the Computer Organization course for many years, there is some data before and after the "crazy models" were introduced. Anecdotally, scores on the final exam question pertaining to the data path have improved since the inception of the *Expo*, but a more rigorous study is warranted.

References

[1] Dhyan Adler-Beléndez et al. "How Are 21st Century Skills Captured in Makerspaces? A Review of The Literature". In: *Proceedings of the FabLearn 2020 - 9th Annual Conference on Maker Education*. FabLearn '20. New York, NY, USA: Association for Computing Machinery, 2021, pp. 40–45. DOI: 10.1145/3386201.3386214.

[2] D. Andrews and D. Roberts. "Academic Makerspaces: Contexts for Research on Interdisciplinary Collaborative Communication". In: *Proceedings of the 35th ACM International Conference on the Design of Communication*. SIGDOC '17. Halifax, Nova Scotia, Canada: Association for Computing Machinery, 2017. DOI: 10.1145/3121113.3121230.

[3] Yoav Bergner et al. "What Are the Learning and Assessment Objectives in Educational Fab Labs and Makerspaces?" In: *Proceedings of FabLearn 2019*. FL2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 42–49. DOI: 10.1145/3311890.3311896.

[4] Erik Brunvand. "Extending Student Labs with SMT Circuit Implementation". In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. GLSVLSI '19. Tysons Corner, VA, USA: Association for Computing Machinery, 2019, pp. 231–236. DOI: 10.1145/3299874.3317968. URL: https://doi.org/10.1145/3299874.3317968.

[5] Katrijn Bulckens et al. "The Mind-and Makerspace: Impact Evaluation of a University Makerspace and the Development of an Impact Measurement Methodology". In: *6th FabLearn Europe / MakeEd Conference 2022*. FabLearn Europe / MakeEd 2022. Copenhagen, Denmark: Association for Computing Machinery, 2022. DOI: 10.1145/3535227.3535236.

[6] Alison Buxton, Louise Kay, and Beth Nutbrown. "Developing a Makerspace Learning and Assessment Framework". In: *6th FabLearn Europe / MakeEd Conference 2022*. FabLearn Europe / MakeEd 2022. Copenhagen, Denmark: Association for Computing Machinery, 2022. DOI: 10.1145/3535227.3535232.

[7] Bernard Frischer. "3D Data Capture, Restoration and Online Publication of Sculpture". In: *3D Recording and Modelling in Archaeology and Cultural Heritage: Theory and Best Practices*. Ed. by Fabio Remondino and Stefano Campana. Archaeopress, Oxford, 2014, pp. 137–144.

[8] Michael B. Gousie and James D. Teresco. "Helping Students Understand the Datapath with Simulators and Crazy Models". In: *Proceedings of the* 40^{th} *SIGCSE Technical Symposium on Computer Science Education*. Denver: ACM, 2013, pp. 329–334.

[9] Léonore V. Guillain and Bertrand Schneider. "Facilitators First: Building a Tool With Facilitators to Foster a More Collaborative Makerspace Community Through Movement Traces". In: *LAK21: 11th International Learning Analytics and Knowledge Conference*. LAK21. Irvine, CA, USA: Association for Computing Machinery, 2021, pp. 533–539. DOI: 10.1145/3448139.3448194.

[10] Jennifer Johnson, Derek Miller, and Kristi Palmer. *Advancing 3D Digitization for Libraries, Museums, and Archives*. Aug. 2018. URL: https://hdl.handle.net/1805/17624.

[11] Michael Johnson and Betsy DiSalvo. "Learning about Complex Adaptive Systems in Makerspaces". In: *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2*. SIGCSE 2022. Providence, RI, USA: Association for Computing Machinery, 2022, p. 1053. DOI: 10.1145/3478432.3499243.

[12] Jarrod Knibbe, Tovi Grossman, and George Fitzmaurice. "Smart Makerspace: An Immersive Instructional Space for Physical Tasks". In: *Proceedings of the 2015 International Conference on Interactive Tabletops and Surfaces*. ITS '15. Madeira, Portugal: Association for Computing Machinery, 2015, pp. 83–92. DOI: 10.1145/2817721.2817741.

[13] Steen Lembcke et al. "Makerspaces in University Colleges: A Workshop about Maker Technologies and Design Thinking in Higher Education". In: *Proceedings of the Conference on Creativity and Making in Education*. FabLearn Europe'18. Trondheim, Norway: Association for Computing Machinery, 2018, pp. 114–115. DOI: 10.1145/3213818.3213843.

[14] Stefano Marziali and Giulia Dionisio. "Photogrammetry and Macro Photography. The Experience of the MUSINT II Project in the 3D Digitization of Small Archaeological Artifacts". In: *Studies in Digital Heritage* 1.2 (2017). https://doi.org/10.14434/sdh.v1i2.23250, pp. 298–309.

[15] Owen G. McGrath. "Making a Makerspace: Designing User Services to Serve Designing Users". In: *Proceedings of the 2016 ACM SIGUCCS Annual Conference*. SIGUCCS '16. Denver, Colorado, USA: Association for Computing Machinery, 2016, pp. 95–98. DOI: 10.1145/2974927.2974949.

[16] Brian Meyer. "Makerspaces in Higher Education: An Overview". In: *ELearn* 2019.5 (July 2019). DOI: 10.1145/3329488.3331179.

[17] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990, p. 240.

[18] Michael P. Rogers and Bill Siever. "What to Make of Makerspaces". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE '19. Minneapolis, MN, USA: Association for Computing Machinery, 2019, pp. 1244–1245. DOI: 10.1145/3287324.3297796.

[19] Lindsey Scheppegrell et al. "Computational Thinking in the Making: Lessons for Second Graders in a STEM Computer Science Immersion School". In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE '19. Minneapolis, MN, USA: Association for Computing Machinery, 2019, p. 1276. DOI: 10.1145/3287324.3293833.

[20] Grazia Tucci, D. Cini, and Alessia Nobile. "Effective 3D digitization of archaeological artifacts for interactive virtual museum". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* 38 (Sept. 2012), pp. 413–420. DOI: 10.5194/isprsarchives-XXXVIII-5-W16-413-2011.

[21] Betty Wallingford et al. "Getting It to Work: Exploring Student-Driven Problem Solving in Computational Making". In: *Proceedings of the FabLearn 2020 - 9th Annual Conference on Maker Education*. FabLearn '20. New York, NY, USA: Association for Computing Machinery, 2021, pp. 110–113. DOI: 10.1145/3386201.3386216.

Spreadsheets As Hands-On Learning Tools In a Discrete Math/Structures Course*

Ali Erkan, John Barr
Computer Science Department
Ithaca College
Ithaca, NY 14850
{aerkan,barr}@ithaca.edu

Abstract

In the context of conventional Computer Science curricula, the timelessness of a Discrete Mathematics (or Discrete Structures) course and the difficulties of engaging enrolled students stem mostly from the same reason: the focus on the mathematical underpinnings of the discipline, which appears distant from the hands-on learning appreciated by our students. Consequently, CS education literature contains numerous papers outlining creative ways to address the engagement matter without diluting the contents of a fundamentally important topic. In this paper, we outline our departmental attempts to utilize spreadsheets to address this very problem.

1 Introduction

Discrete Math/Discrete Structures (referred to as DM from here on) is one of the rare constants in the continuously evolving landscape of Computer Science education. As new systems, new languages, and new paradigms are created, the mathematical underpinnings of our discipline do not get swayed. If anything, the rising focus on concurrent algorithms, machine learning, real-time systems, and so on increases the need for a strong mathematical backbone.

Yet DM remains a difficult course to teach because, unlike coding, it takes multiple semesters for students to realize the usefulness of the subject matter. As discussed in [9], even the determination of the ideal time to teach DM is not trivial when student appreciation factors are weighted in.

To increase the effectiveness of the DM course taught at our institution, we had been seizing a number of low hanging pedagogical fruits such as switching to an online textbook (with built-in engagement exercises), keeping our class-sizes small, making class time interactive, and pointing out the connections from DM topics to their applications in upper level courses. And we had been seeing reasonably good student performance as well as reasonably good course evaluations. However, during a seemingly unrelated assessment exercise in one of our senior level courses, we noticed a peculiar situation. "Algorithms + Organization = Systems" is a research-paper driven capstone course of ours to highlight the prevalence of algorithmic principles in the design of networks, operating systems, and architecture [7, 8]; it is a multifaceted context where students see the continuity between areas they may otherwise consider to be disconnected. A pre-post survey done in this course revealed that the connection between DM and any area under the umbrella term of "systems" was not sufficiently apparent to our students. In particular, as shown in figure 1, the impact/relevance/importance of DM was the only one out of a number of departmentally prioritized measures that had not changed after students spent a semester studying systems. This was initially surprising since all of the topics we had covered explicitly depended on DM topics such as summations, functions, relations, expected values, discrete probability, and modulo arithmetic. But we then realized that we were in fact witnessing a lack of "deep learning."

Deep learning is defined as relating information to previous knowledge, working to structure the content, and applying the content in new ways. Its formation is not easy even when students are seemingly effective at reading a textbook [14, 13]. Given its importance in education in general, deep-vs-shallow learning has been the focus of a multitude of studies over a range of disciplines [19, 20, 12, 6, 25].

Our efforts to address this deficiency started with a literature search revealing, as expected, that DM has been the focus of many studies: [26, 23] focus on motivation via highlighting application areas, [1, 4] consider instructional assistance, [24, 15] advocate programming to increase engagement, [2, 21, 16] outline pedagogical alternatives and innovations, and [11, 5] discuss curricular aspects. For a comprehensive survey up to 2011, please see [18].

Based on the particular problem that had motivated us and the work that had been done before us, we identified two specific goals to address deep learning: we wanted our students (i) to have a stronger sense of intuition for the associated mathematical ideas, and (ii) to test their comprehension in a lab

environment akin to the lab hour of a conventional CS1 and CS2 course. As primary constraints, (i) we did not want to take out topics to make room for experimentation, and (ii) we did not want to compromise class time by having to teach one or more complex tool(s) for experimentation. Although the idea of incorporating coding exercises initially seemed enticing, we decided to use spreadsheet-based hands-on exercises.

2 Spreadsheets in Discrete Math

The use of spreadsheets in CS education is not unprecedented, with early examples exploring the suitability of the idea in a CS-1 environment [22, 10]. Recent instances tend to focus on illuminating specific topics such as complexity [17], finite state modeling [3], and encryption [27]. Our own work led us to the realization that spreadsheets can indeed be used to cover a wide range of topics in a conventional DM course. We also observed that students are readily on board with this idea because they consider increased spreadsheet competencies to be a net gain for their professional and personal future.

We structured our four-credit course to have three lectures and a lab. Unless an exam was scheduled, students were asked to complete spreadsheet-based exercises based the topics of the most recent lectures. We exclusively used Google Sheets due to its simple and streamlined user interface and its ability to host multiple students on the same sheet (in case the exercises required group work). Even though most students already had some elementary knowledge of spreadsheets, we assumed that they were starting from scratch and identified short instructional videos through YouTube or Lynda.com; these were assigned as homework so that we never lost more than a few minutes of lab time to teach the use of the tool. With what they learned, students were able to construct most (if not all) of the content of the spreadsheets on their own.

We first designed a set of **intuition** exercises to assist (rather than assess) comprehension (section 3). For example, in the context of discrete probability, many students get confused as to when they should be multiplying vs adding measures of probability. In particular, if we tell them that we are tossing three fair coins and ask them the likelihood of getting three tails (i.e. an outcome of '`TTT`'), many do not immediately conclude that the correct answer is $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{1}{8}$. The conventional way to address this problem is to draw the underlying probability tree to enumerate all eight outcomes of equal likelihood which then leads to the correct answer. But this still happens to be analytical thinking and does not necessarily provide intuition. The spreadsheet we see in figure 2, on the other hand, tries to provide that very missing piece. Here, rows 2, 3, and 4 represent the tossing of three fair coins 32 times and observing how often they all turn out to be tails. While this particular instance has five cases

of '`TTT`', any edit action on the spreadsheet triggers the regeneration of random numbers to produce a new batch of $3 \times 32 = 96$ coin tosses. After a statistically significant number of repeats (i.e. ≥ 20), we see that the average number of `Y`'s on row `3T` tends toward 4; since $\frac{4}{32} = \frac{1}{8}$, this observation scaffolds students into seeing why multiplication (as opposed to addition) must be the correct operation for the problem. Furthermore, since the coin toss formula refers to cell '`A1`', students also explore the consequences of biasing coins. Finally, we also see that even though genuine randomness does not preclude the occurrence of long repetitions of the same side (shown in gray in figure 2), the probability of seeing three tails always converges to the mathematically predicted value.

We then designed a second group exercises to test students' comprehension of topics; we classified these as **explorations** (section 4). One specific exploration we completed after our trial semester proved to be particularly effective in showing the versatility of spreadsheets: integer addition of two fixed-length binary numbers. As a digital circuits application of truth tables, most DM books show the construction of a half-adder (a simple device that takes two binary digits to produce a sum digit and a carry digit) using basic logic operators. Figure 3a shows the realization of this logic in a spreadsheet while figure 3b confirms its correctness for all the additions it supports: $0+0$, $0+1$, $1+0$, and $1+1$. Two half-adders can then be coupled to create a full-adder which adds three binary digits (two input bits as well as the carry bit of another single digit addition); we see an instance of this in figure 3c. Adding two N bit numbers then becomes a matter of using a bank of N full-adders; figure 3d shows this in action for calculating $001 + 010 = 001$, $011 + 010 = 101$, and $011 + 110 = 001$. Note that the last case demonstrates how an incorrect result emerges as a consequence of having a fixed number of digits.

3 Intuition Exercises

We collected student feedback for the following intuition exercises:

- **Truth tables**: To construct a truth table with n Boolean variables, students create a spreadsheet with 2^n rows where the first n columns specify the n independent variables and the remaining columns calculate any desired Boolean function using '`=and()`', '`=or()`', and '`=not()`'. For example, '`=or(A1,and(B1,C1))`' means $a \vee (b \wedge c)$ were '`A1`', '`B1`', and '`C1`' are the cells that contain the a, b, c values.
- **Conversion to base 10**: Students encode base-conversion algorithms where the base in saved a particular cell. Students observe the universality of the algorithm by changing the base value and observing that the new result remains correct (figures 4 and 5).

- **Summations**: Summations of the form $\sum_{i=a}^{b} f(i)$ initially appear illusive to students as they do not readily understand the iterative nature of the calculation. By spelling out the expanded forms of particular instances such as $\sum_{i=0}^{10} i$, $\sum_{i=0}^{10} 2i$, $\sum_{i=5}^{15} 2i$ in a spreadsheet and calculating their sum with '`=sum()`', students internalize that Σ is just a multi-term addition (figure 6). This sets the stage to understand the consequences of changing the range of the i iterator, multiplying the summed term by a constant, etc, as well as initiating the proofs of the equalities of $\sum_{i=a}^{b} kf(i) = k\sum_{i=a}^{b} f(i)$, $\sum_{i=a}^{b} f(i) = \left(\sum_{i=a}^{b-1} f(i)\right) + f(b)$, etc.
- **Cartesian product**: For $A \times B$, students enumerate the members of sets A and B as row/column headers (respectively) and create the elements with '`=concatenate()`' over header combinations. '`=flatten()`' then "flattens" this table into a single row to calculate $(A \times B) \times C$.
- **Logarithms**: Many students have brittle comprehension of the novel properties of the logarithm function and thus benefit from producing "tables of confirmations" in a spreadsheet. For example, for $\log\left(\frac{a}{b}\right) \equiv \log(a) - \log(b)$, students produce two tables, one for the left side of the equivalence and one for the right. A third table of equality assertions (as in '`=A1=K1`' that produces a '`True`' iff the contents of cells '`A1`' and '`K1`' are the same) then shows that no matter what value pair students choose, they cannot create a '`False`'.
- **The "choose" function**: Students typically miss two important aspects of the choose function $\binom{N}{r}$. First, $\binom{N-1}{r} + \binom{N-1}{r-1}$ is an efficient way of calculating $\binom{N}{r}$. Second, there is symmetry in the values of $\binom{N}{r}$ as r goes from 0 to N. Students witness both in a spreadsheet (figure 8 and 7).
- **Discrete Probability**: It is not uncommon for the terms "outcome" (an element of the sample space) and "event" (a subset of the sample space) to be confused with each other but spreadsheet exercises can contrast them. Figure 9 shows different events defined over flipping three coins: (i) seeing exactly two heads, (ii) seeing anything other than exactly two heads, (iii) seeing at least two heads, and (iv) seeing at most two heads. Based on the formulaic definitions of these events, students use conditional formatting to highlight outcomes that count toward satisfying each. The trivial recalculation of all probabilities also allow students see how biasing coins influence the occurrence of events.
- **Expected Value**: By eliminating time-consuming hand calculations, spreadsheets allow students to cover a greater number of examples to internalize the purpose of this function. Spreadsheets also facilitate the convergence of expected values over infinite sample spaces.

4 Exploration Exercises

We collected student feedback for the following exploration exercises:

- The birthday problem is a classic complementary-thinking exercise where the likelihood of having at least two people having the same birthday is one minus the probability of everyone having distinct birthdays. However, due to the combinatoric aspects present in this problem, the changes of the sought probabilities as "the room gets more crowded" is not linear. We therefore have an exploratory exercise where students first try to understand these dynamics and subsequently propose an analytical solution that explains their numeric and chart results.
- Most networking books contain a classic frequency division multiplexing question: if a particular connection can simultaneously deliver N packets and if we have M (which is $> N$) virtual connections willing to use that connection each of which is p% likely to transmit at any moment in time, how many virtual connections can be admitted if we can afford to lose at most 10% of the packets? We have in the past not been able to bring our students in our DM course to solve this problem. However, when we reskinned it to a health-care/medical-person-assignment version[1], we were able to guide students to experiment with the problem in a spreadsheet.

5 Student Feedback

At the end of our trial semester, we had 43 students fill out a survey with 15 assertions each of which had to be responded using a seven-level agree/disagree scale. In order to classify responses as a function of their performance, we also had students self-report a letter grade that best represented their comprehension of the concepts covered in class. These results are shown in figure 10. If we look at the averages, we can see that students responded positively to every statement; in fact, except for two points for the A/A- cohort, all averages are above the "somewhat agree" level. We consider this to be an encouraging bottom-line for our work.

When we look at the left-most 10 stops on the x-axis, we see that the responses from the three letter-grade levels do not always correlate with each other. For example, the highest average for the C-/C/C+ cohort was the intuition exercise of coin-tossing; this also happened to be a low-point for the

[1]We are responsible for dispatching social workers; we have a probabilistic measure that indicates the likelihood of an individual needing a social worker; our task is to make sure that 90% of the time, someone who needs a social worker gets one at the time of need; assuming each social worker needs $X per years, what budget do we need to satisfy the stated constraints?

A-/A cohort. In fact, the latter group was typically not too impressed with our most straight-forward intuition exercises. On the contrary, the exercises that intrigued the A-/A cohort most were the challenging exploration exercises.

When it came to the use of spreadsheets enhancing the CONTROL, ENGAGEMENT, UNDERSTANDING, and ENJOYMENT dimensions of learning, the B-/B/B+ and C-/C/C+ cohorts responded more positively than the A/A- cohort; it would be safe to guess that A/A- students probably score high on these dimensions regardless of what enticement factors are being used in class. Interestingly, when it came to the importance of spreadsheets for their future, it was the C-/C/C+ cohort that trailed the pack.

Since the effectiveness of our interventions appears to be a function of cohorts, we also did a Pearson correlation between the three; the results are shown in figure 11. The top table shows that in terms of benefiting from the spreadsheet exercises, the greatest correlation appears to be between the A/A- and B-/B/B+ cohorts. The bottom table leads to the same conclusion by even a greater margin. This means that as we explore ways to increase the depth of learning in our DM course, even though we have evidence to suggest that a spreadsheet based lab component makes a difference, this intervention will have to be served differently to our A/B students vs our C students.

6 Conclusions

After the completion of our trial semester, more polished versions of the mentioned spreadsheet exercises (as well as a few others) are currently being tested for a second time. Therefore, our most immediate goal is to see whether the conclusions we have reached from the student feedback of the trial semester will improve. We are particularly interested in retrying the assessment exercises of the ALGORITHMS + ORGANIZATION = SYSTEMS course during its next offering; after all, a general lack of realization of the importance of DM in the context this capstone course was what had started us in the experience we reported.

We were happy to see that spreadsheets can indeed serve as a powerful and generic lab tool for a mathematical course without the need for programming or topic specific tools/animations. Furthermore, we were able to confirm that these advantages do not come at the cost of class-time. We are happy to share all instructional materials (such as list of videos on Google Sheets, lab instructions/problems, solutions) with the educational community upon request.

References

[1] Laura E. Brown, Adam Feltz, and Charles Wallace. "Lab Exercises for a Discrete Structures Course: Exploring Logic and Relational Algebra with Alloy". In: *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE 2018. Larnaca, Cyprus: ACM, 2018, pp. 135–140. ISBN: 978-1-4503-5707-4. DOI: 10.1145/3197091.3197127. URL: http://doi.acm.org/10.1145/3197091.3197127.

[2] Lijuan Cao and Audrey Rorrer. "An Active and Collaborative Approach to Teaching Discrete Structures". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE '18. Baltimore, Maryland, USA: ACM, 2018, pp. 822–827. ISBN: 978-1-4503-5103-4. DOI: 10.1145/3159450.3159582. URL: http://doi.acm.org/10.1145/3159450.3159582.

[3] Sujit Kumar Chakrabarti and Srihari Sukumaran. "Using Spreadsheets for Finite State Modelling". In: *Proceedings of the 2Nd India Software Engineering Conference*. ISEC '09. Pune, India: ACM, 2009, pp. 27–36. ISBN: 978-1-60558-426-3. DOI: 10.1145/1506216.1506223. URL: http://doi.acm.org/10.1145/1506216.1506223.

[4] Travis J. Cossairt and Joseph J. LaViola Jr. "SetPad: A Sketch-based Tool for Exploring Discrete Math Set Problems". In: *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*. SBIM '12. Annecy, France: Eurographics Association, 2012, pp. 47–56. ISBN: 978-3-905674-42-2. URL: http://dl.acm.org/citation.cfm?id=2331067.2331075.

[5] Adrienne Decker and Phil Ventura. "We Claim This Class for Computer Science: A Non-mathematician's Discrete Structures Course". In: *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '04. Norfolk, Virginia, USA: ACM, 2004, pp. 442–446. ISBN: 1-58113-798-2. DOI: 10.1145/971300.971448. URL: http://doi.acm.org/10.1145/971300.971448.

[6] Noel Entwistle. "Taking stock: teaching and learning research in higher education". In: *Review prepared for an international symposium on" Teaching and Learning Research in Higher Education", Guelph, Ontario*. 2008.

[7] Ali Erkan and John Barr. "Algorithms + Organization = Systems". In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '16. Arequipa, Peru: ACM, 2016, pp. 65–70. ISBN: 978-1-4503-4231-5. DOI: 10.1145/2899415.2899458. URL: http://doi.acm.org/10.1145/2899415.2899458.

[8] Ali Erkan et al. "Developing a Holistic Understanding of Systems and Algorithms Through Research Papers". In: *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*. ITiCSE-WGR '17. Bologna, Italy: ACM, 2017, pp. 86–104. ISBN: 978-1-4503-5627-5. DOI: 10.1145/3174781.3174786. URL: http://doi.acm.org/10.1145/3174781.3174786.

[9] David Gries et al. “Discrete Mathematics/Structures: How Do We Deal with the Late Appreciation Problem?” In: *J. Comput. Sci. Coll.* 24.6 (June 2009), pp. 110–112. ISSN: 1937-4771. URL: http://dl.acm.org/citation.cfm?id=1529995.1530017.

[10] Mary Veronica Kolesar and Vicki H. Allan. “Teaching Computer Science Concepts and Problem Solving with a Spreadsheet”. In: *SIGCSE Bull.* 27.1 (Mar. 1995), pp. 10–13. ISSN: 0097-8418. DOI: 10.1145/199691.199698. URL: http://doi.acm.org/10.1145/199691.199698.

[11] Mark D. LeBlanc and Rochelle Leibowitz. “Discrete Partnership: A Case for a Full Year of Discrete Math”. In: *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’06. Houston, Texas, USA: ACM, 2006, pp. 313–317. ISBN: 1-59593-259-3. DOI: 10.1145/1121341.1121438. URL: http://doi.acm.org/10.1145/1121341.1121438.

[12] William F Long. “Dissonance detected by cluster analysis of responses to the approaches and study skills inventory for students”. In: *Studies in Higher Education* 28.1 (2003), pp. 21–35.

[13] Ference Marton, Dai Hounsell, and Noel James Entwistle. *The experience of learning: Implications for teaching and studying in higher education*. Scottish Academic Press, 1997.

[14] Ference Marton and R Säaljö. “On qualitative differences in learning: ii - Outcome as a function of the learner’s conception of the task”. In: *british Journal of educational Psychology* 46.2 (1976), pp. 115–127.

[15] Kirby McMaster, Nicole Anderson, and Brian Rague. “Discrete Math with Programming: Better Together”. In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE ’07. Covington, Kentucky, USA: ACM, 2007, pp. 100–104. ISBN: 1-59593-361-1. DOI: 10.1145/1227310.1227348. URL: http://doi.acm.org/10.1145/1227310.1227348.

[16] Norman Neff. “Problem-directed Discrete Structures Course”. In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. SIGCSE ’10. Milwaukee, Wisconsin, USA: ACM, 2010, pp. 148–151. ISBN: 978-1-4503-0006-3. DOI: 10.1145/1734263.1734315. URL: http://doi.acm.org/10.1145/1734263.1734315.

[17] Jeff Parker. “Teaching Complexity via Spreadsheets”. In: *J. Comput. Sci. Coll.* 23.5 (May 2008), pp. 83–89. ISSN: 1937-4771. URL: http://dl.acm.org/citation.cfm?id=1352627.1352642.

[18] James F. Power, Thomas Whelan, and Susan Bergin. “Teaching Discrete Structures: A Systematic Review of the Literature”. In: *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*. SIGCSE ’11. Dallas, TX, USA: ACM, 2011, pp. 275–280. ISBN: 978-1-4503-0500-6. DOI: 10.1145/1953163.1953247. URL: http://doi.acm.org/10.1145/1953163.1953247.

[19] Paul Ramsden et al. “The context of learning in academic departments”. In: *The experience of learning* 2 (1997), pp. 198–216.

[20] John TE Richardson. *Researching student learning: Approaches to studying in campus-based and distance education*. Open University Press, 2000.

[21] Michael R. Scheessele, Hang Dinh, and Mahesh Ananth. "On Adding a Critical Thinking Module to a Discrete Structures Course". In: *J. Comput. Sci. Coll.* 30.6 (June 2015), pp. 97–103. ISSN: 1937-4771. URL: http://dl.acm.org/citation.cfm?id=2753024.2753045.

[22] Dermot Shinners-Kennedy. "Using Spreadsheets to Teach Computer Science". In: *Proceedings of the Seventeenth SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '86. Cincinnati, Ohio, USA: ACM, 1986, pp. 264–270. ISBN: 0-89791-178-4. DOI: 10.1145/5600.5905. URL: http://doi.acm.org/10.1145/5600.5905.

[23] Daniel E. Stevenson, Michael R. Wick, and Steven J. Ratering. "Steganography and Cartography: Interesting Assignments That Reinforce Machine Representation, Bit Manipulation, and Discrete Structures Concepts". In: *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '05. St. Louis, Missouri, USA: ACM, 2005, pp. 277–281. ISBN: 1-58113-997-7. DOI: 10.1145/1047344.1047443. URL: http://doi.acm.org/10.1145/1047344.1047443.

[24] Thomas VanDrunen. "The Case for Teaching Functional Programming in Discrete Math". In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. OOPSLA '11. Portland, Oregon, USA: ACM, 2011, pp. 81–86. ISBN: 978-1-4503-0942-4. DOI: 10.1145/2048147.2048180. URL: http://doi.acm.org/10.1145/2048147.2048180.

[25] Kevin Warburton. "Deep learning and education for sustainability". In: *International Journal of Sustainability in Higher Education* 4.1 (2003), pp. 44–56.

[26] Michael R. Wick and Paul J. Wagner. "Using Market Basket Analysis to Integrate and Motivate Topics in Discrete Structures". In: *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*. SIGCSE '06. Houston, Texas, USA: ACM, 2006, pp. 323–327. ISBN: 1-59593-259-3. DOI: 10.1145/1121341.1121440. URL: http://doi.acm.org/10.1145/1121341.1121440.

[27] Hong Biao Zeng. "Teaching the RSA Algorithm Using Spreadsheets". In: *J. Comput. Sci. Coll.* 28.1 (Oct. 2012), pp. 18–24. ISSN: 1937-4771. URL: http://dl.acm.org/citation.cfm?id=2379703.2379708.

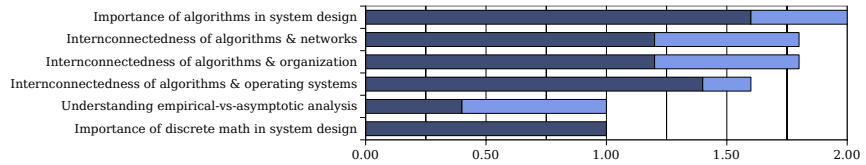
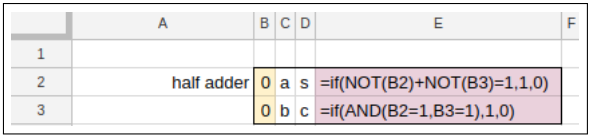


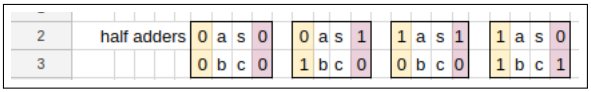
Figure 1: ***Gains in cross-cutting curricular goals as a result of taking our "Algorithms + Organization = Systems" course:*** *The dark segments on the left sides of the bars represent students' responses from the pre-test while the lighter segments represent the gains we saw from our post-tests. "Importance of discrete math in system" was the only measure where students' perception had not changed for the better.*

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG
1	**0.5**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
2	coin 1	H	T	H	H	T	T	T	H	H	H	T	H	H	T	H	T	H	T	T	T	T	T	T	T	H	T	T	H	T	T	T	T
3	coin 2	H	T	T	T	H	T	T	H	H	H	T	T	T	T	H	T	T	T	H	H	T	H	H	T	H	T	T	H	H	H	H	T
4	coin 3	H	T	T	T	T	H	T	H	T	H	H	T	H	T	T	T	H	T	H	H	H	T	H	H	H	H	H	T	H	H	H	H
5	3T		Y					Y							Y		Y		Y														

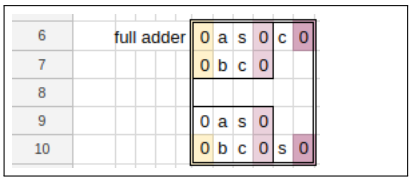
Figure 2: ***The emulation of tossing three coins 32 times:*** *The probability of getting heads is set by the value in cell* `A1`*. Rows 2, 3, and 4 are the outcomes of the coins; row 5 is the indicator for when all three joins turn up tails. The emulation of each coin toss is based on the formula* `=if(rand()<=A1,"T","H")` *while the detection of all three coins being tails is based on the formula* `=if(and(B2="T",B3="T",B4="T"),"Y","")`.



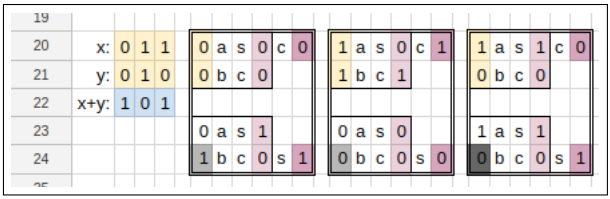
(a) *Formulas implementing a half-adder*



(b) *All four cases for a half-adder*



(c) *A full-adder from two half-adders*



(d) *Adding two three-bit binary numbers: 011+010=101*

Figure 3: ***Implementations of half-adders and full-adders in a spreadsheet****: In all figures, we use '*`a`*' and '*`b`*' to represent input and '*`c`*' and '*`s`*' to represent carry/sum. Part 3a shows the implementation of the conventional half-adder circuit using '*`AND`*' and '*`NOT`*' operators. Part 3b shows the correct operation of this half-adder for all possible input configuration. Part 3c shows the conventional construction of a full-adder from two half adders; as expected, the carry digit of the first half-adder is connected to the first input of the second half-adder. Finally, part 3d shows adding two three-digits binary numbers.*

	A	B	C	D	E	F	G	H
1	Base:	2		4th digit	3rd digit	2nd digit	1st digit	Result
2			Powers of the base:	8	4	2	1	
3			Representation of a number:	0	1	1	0	
4			Product of matching values:	0	4	2	0	6

Figure 4: ***Converting base 2 to base 10...***

	A	B	C	D	E	F	G	H
1	Base:	3		4th digit	3rd digit	2nd digit	1st digit	Result
2			Powers of the base:	27	9	3	1	
3			Representation of a number:	0	1	1	0	
4			Product of matching values:	0	9	3	0	12

Figure 5: ***Converting base 3 to base 10...***

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Sequence 1:	0	1	2	3	4	5	6	7	8	9	10	55
2	Sequence 2:	0	2	4	6	8	10	12	14	16	18	20	110
3	Sequence 3:	10	12	14	16	18	20	22	24	26	28	30	220

Figure 6: ***Expanded forms of three summations...***

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	C(N,r)	0	1	2	3	4	5	6	7	8	9	10	r
2	0	1											
3	1	1	1										
4	2	1	2	1									
5	3	1	3	3	1								
6	4	1	4	6	4	1							
7	5	1	5	10	10	5	1						
8	6	1	6	15	20	15	6	1					
9	7	1	7	21	35	35	21	7	1				
10	8	1	8	28	56	70	56	28	8	1			
11	9	1	9	36	84	126	126	84	36	9	1		
12	10	1	10	45	120	210	252	210	120	45	10	1	
13	N												

Figure 7: ***Calculation of*** $\binom{N}{r}$ ***values based on*** $\binom{N-1}{r} + \binom{N-1}{r-1}$

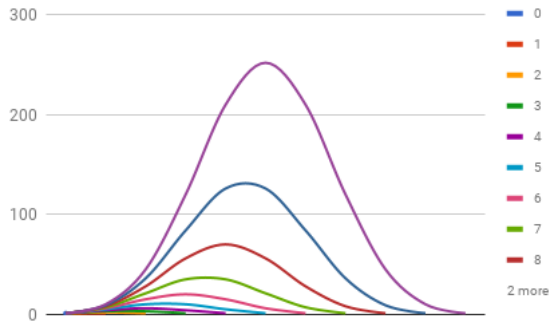


Figure 8: ***Revealing the symmetry of*** $\binom{N}{r}$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	c1	c2	c3		p(c1=H)	p(c2=H)	p(c3=H)	probability		Event 1: Exactly two H			Event 2: Not two H			Event 3: At least two H			Event 4: At most two H	
2	H	H	H		0.4	0.4	0.4	0.064		0	0.000		1	0.064		1	0.064		0	0.000
3	H	H	T		0.4	0.4	0.6	0.096		1	0.096		0	0.000		1	0.096		1	0.096
4	H	T	H		0.4	0.6	0.4	0.096		1	0.096		0	0.000		1	0.096		1	0.096
5	H	T	T		0.4	0.6	0.6	0.144		0	0.000		1	0.144		0	0.000		1	0.144
6	T	H	H		0.6	0.4	0.4	0.096		1	0.096		0	0.000		1	0.096		1	0.096
7	T	H	T		0.6	0.4	0.6	0.144		0	0.000		1	0.144		0	0.000		1	0.144
8	T	T	H		0.6	0.6	0.4	0.144		0	0.000		1	0.144		0	0.000		1	0.144
9	T	T	T		0.6	0.6	0.6	0.216		0	0.000		1	0.216		0	0.000		1	0.216
10																				
11	p(H)=	0.4						100.00%			28.80%			71.20%			35.20%			93.60%

Figure 9: ***Illustration of the sample space for tossing three coins and the definition of four events over this sample space***. *Spreadsheet formulas and conditional formatting show how a particular outcome would determine which events have occured.*

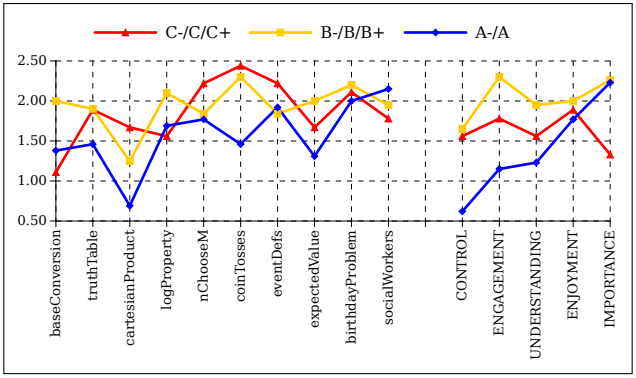


Figure 10: ***Student responses to using the spreadsheet exercises outlined in this paper****: The y-values for the leftmost 10 points on the x-axis correspond to statements of the form "Exercise <name> was beneficial to my learning". The y-values for the next four points on the x-axis focus on the impact of using spreadsheet exercises on four significant dimensions of learning:* CONTROL, ENGAGEMENT, UNDERSTANDING, *and* ENJOYMENT*. The rightmost point shows the degree to which students consider spreadsheets to be important for their future. Responses were based on a seven-point Likert scale of "Strongly disagree" (-3), "Disagree" (-2), "Somewhat disagree" (-1), "Somewhat disagree" (1), "Agree" (2), and "Strongly agree" (3). We lower-bounded the y-axis at 0.5 since none of the averages fells under this limit (i.e. all the averages were in the "agree" side). 43 students filled out the survey. Letter-grade partitioning of the students was based on the their self-reporting of their competence level in Discrete Math (i.e. the survey was anonymous).*

Exercises	C-/C/C+	B-/B/B+	A-/A
C-/C/C+	1.00	0.23	0.34
B-/B/B+		1.00	0.58
A-/A			1.00

Overall	C-/C/C+	B-/B/B+	A-/A
C-/C/C+	1.00	-0.01	-0.22
B-/B/B+		1.00	0.65
A-/A			1.00

Figure 11: ***Calculation of the Pearson correlation coefficient between the three cohorts defined on self-reported letter-grades****. The left table is on all spreadsheet exercises; the right table is on students' responses to the* CONTROL, ENGAGEMENT, UNDERSTANDING, *and* ENJOYMENT *dimensions.*

Incorporating Cybersecurity Concepts in Connecticut's High School STEM Education*

Liberty D. Page [1], Mehdi Mekni [1], Elizabeth A. Radday [2]
[1]University of New Haven, West Haven, CT 06516
[2]EdAdvance, Litchfield, CT 06759
{lpage,mmekni}@newhaven.edu, radday@edadvance.org

Abstract

This paper presents the GenCyber Teacher Academy (GTA), a unique professional development program that provides Connecticut's high school teachers across various STEM disciplines with opportunities to explore cybersecurity concepts and incorporate them in their curriculum. Participating teachers experienced inquiry-based learning, focused classroom discourse, and collaborative learning that centered on GenCyber Cybersecurity Concepts. Results indicate GTA enabled teachers to reflect on best practices in incorporating cybersecurity concepts while promoting online safety. Moreover, GTA established a sustainable GenCyber Teacher Academy Teaching Learning Community of high school teachers supported by a community of practitioners that will collectively shape the future of cybersecurity in Connecticut.

1 Introduction

The importance of cybersecurity education in Connecticut became clear when the *Board of Education* developed the *Position Statement on Computer Science Education* for all K-12 students. In 2015, a bill was passed requiring all high schools to offer Computer Science (CS) and Cybersecurity courses. However, this bill did not provide funding for professional development programs and curricula development and support [1, 2]. Therefore, cybersecurity is still

marginalized throughout education in Connecticut [3]. Additionally, Connecticut has a distributional problem with cybersecurity in high schools [4]. The demand for teachers and for increasing diversity in the teacher workforce is concentrated in school districts that are already challenged in recruiting and retaining teachers [5]. These districts, designated *Opportunity Districts*, are the 10 lowest performing districts in the state based on the accountability index [6]. They serve large urban areas with a community of students that is historically underrepresented and under-served, especially women, minorities, and students from economically disadvantaged backgrounds. There is an urgent and critical need to focus on high schools in opportunity districts and strengthen their capacity to reliably produce valued cybersecurity education outcomes for diverse groups of students, educated by prepared and supported teachers.

There is clear evidence to support the benefits of a diverse teacher workforce, including its positive impact on strengthening schools and resulting in better outcomes for students of all races/genders/ethnicities [7, 8]. However, in 2019-20, Connecticut's teacher workforce was made of 9.6% of educators of color while more than 45% of the state's students identify as people of color [4]. It is sadly acknowledged that women, students of color are underrepresented in cybersecurity learning opportunities [9, 10]. Addressing the diversity issue in the high school teacher workforce in Connecticut is a lengthy and complex process. However, training teachers on diversity helps them rapidly create a sustainable inclusive environment where all students including underrepresented minorities can thrive.

The goal of this project is to meet the rising need for highly-skilled, outside-the-box thinking cybersecurity professionals to protect the nation and support its governmental workforce. To this end, the University *of* New Haven provided Connecticut's **first** GenCyber Teacher Academy (GTA) program that trains and supports high school teachers to promote cybersecurity and online safety.

The GTA program is a learner-centered, hands-on, intensive program with a focus on **GenCyber Cybersecurity Concepts Framework** designed to train 9th-12th grade STEM high school teachers. The program activities include lectures, games, labs, lesson plan design and development, with evaluation supported by a K-12 pedagogy and curricula expert. A series of daily **Cybersecurity Seminars** featuring guest speakers from industry, government, academia, and non-profit organizations to increase awareness of post-secondary opportunities and careers in cybersecurity. The program is complemented with monthly **GenCyber Teacher Academy Learning Community** (GTALC) follow-up events in the fall offering continuous professional development, mentoring and coaching support to participating teachers.

The goals of our GTA program are: 1) Design, develop, and implement a cy-

bersecurity professional development program. Culturally responsive teaching and how it applies to teaching cybersecurity will also be addressed throughout the program; 2) Design, develop and validate cybersecurity lesson plans and associated teaching and assessment materials. 3) Build a sustainable GenCyber Teacher Academy Learning Community of high school teachers supported by a community of practitioners that will collectively shape the future of cybersecurity in Connecticut.

The remainder of this paper is organized as follows: Section 2 provides an overview of the GenCyber program. Section 3 highlights the GenCyber Teacher Academy structure. Section 4 details the curriculum design, learning outcomes, and assessment techniques. Section 5 presents the results of the GTA program. Section 6 discusses the program and future work.

2 Background

The GenCyber program provides cybersecurity experiences for students and teachers at the secondary level. The GenCyber program strives to be a part of the solution to the Nation's shortfall of skilled cybersecurity professionals. Ensuring that enough young people are inspired to utilize their talents in cybersecurity is critical to the future of our country's national and economic security as we become even more reliant on cyber-based technology in every aspect of our daily lives. To ensure a level playing field, GenCyber camps are open to all student and teacher participants at no cost. The GenCyber program is financially supported by the National Security Agency, the National Science Foundation, and other federal partners on an annual basis.

3 GenCyber Teacher Academy

3.1 Program Overview

The proposed GTA program focuses on the GenCyber Cybersecurity Concepts framework. As we target STEM 9th-12th grade teachers with no prior knowledge in computing, our GTA and associated curriculum has been designed and will be delivered to a **beginner** audience. The GTA is organized into five modules: (1) Network Fundamentals, (2) Python Programming & Scripting, (3) Cybersecurity Awareness, Ethics & Trends, (4) Cryptography, and (5) Social Engineering Attacks & Prevention. These concepts are important topics for cybersecurity training because they provide a comprehensive understanding of the cybersecurity field. Understanding computer networks is crucial for cybersecurity professionals because it enables them to understand how data is transmitted across networks and identify potential security threats, such

as network attacks and network intrusions. Python is a popular programming language that is widely used in the cybersecurity industry for writing scripts and automating various cybersecurity tasks. Knowledge of cryptography is necessary to understand how encryption algorithms work and how to secure communications and data transmission. Cybersecurity awareness training helps individuals and organizations identify and avoid potential security threats, such as phishing attacks, social engineering, and malware. Social engineering is a technique used by attackers to trick individuals into revealing sensitive information.

By studying these topics, cybersecurity professionals can gain the necessary skills and knowledge to help protect organizations and individuals from potential security threats, as well as educate others about the importance of cybersecurity. Each day, a new module is presented, and the associated activities are performed under the assistance and support of the lead instructor, the module instructor, the K-12 pedagogy expert, and the teaching assistants. Each module aligns the scheduled activities including presentations, labs, games, assessment, seminars, lesson plan development, and participant discussions and reflections. The modules have been selected in close collaboration with our K-12 pedagogy and curriculum specialist, University Computer Science (CS) and Cybersecurity faculty, and a selection of representatives made up of partnering high school teachers. The program includes series of **Cybersecurity Seminars** scheduled daily during lunch. Guest speakers leverage their knowledge and experience to share their vision and perspective on cybersecurity challenges and opportunities. The purpose of these seminars is to expose participants to cybersecurity career paths and emerging threats. To meet the GenCyber Teacher Academy program objectives, the proposed work was mapped to the tasks shared in Table 1.

Table 1: GTA Program Timeline and Phases

Phases	2021		2022				2023		
	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3
T1: Announcement & Marketing		⊗	⊗	⊗					
T2: Teacher Recruitment			⊗	⊗					
T3: Teacher Selection				⊗					
T4: Pre-Program Outreach					⊗				
T5: Summer Program Execution					⊗		⊗		
T6: Post-Program Outreach (GTALC)						⊗	⊗	⊗	⊗
T7: Program Reporting			⊗		⊗		⊗		

3.2 Pre-Program Outreach

The pre-program outreach aims to cover the following topics: (1) *participants on-boarding*, (2) *multidisciplinary groups organization*, (3) *introduction of GenCyber Cybersecurity Concepts*, (4) *cybersecurity awareness self-assessment*, (5) *interactive discussion forums*. First, participants are introduced to the detailed schedule of the GTA program and the associated detailed pre, post and summer program activities. An introduction from our GTA team members is provided. Participants are invited to introduce themselves and share their background, experience, interest in cybersecurity, and career objectives. Next, participants are organized in multidisciplinary groups that consider diversity, STEM background, and experience factors. Online presentations and learning materials are shared introducing each *GenCyber Cybersecurity Concept*. Various forms of assessments with helpful feedback from the GTA team are used. First, systematic cybersecurity concepts self-assessment quizzes. Second, a reflection assignment demonstrating the understanding of these concepts is required. A total of **eight-hour** self-paced online learning is required to complete the pre-camp outreach activities.

3.3 Post-Program Outreach

Continuous professional development is central to our GTA program. We strongly believe participant-participant and participant-instructor interactions should take place continuously, suggesting that participants should have consistent encounters after the GTA summer camp. Our GTA program is complemented with monthly *GenCyber Teacher Academy Learning Community* (GTALC) virtual follow-up sessions offering mentoring and coaching support to participating teachers during the fall. The proposed GTALC is a connection and exchange virtual space dedicated to high school teachers to share their experiences, seek advice from experts and practitioners, access resources, and gain knowledge and skills in inquiry-based pedagogy in cybersecurity that is inclusive for all students, particularly URM and women. GTALC is led by our K-12 pedagogy expert and facilitated by a community of practice which includes the rest of the GTA team members, representatives from our collaborators CSDE and CSTA as well as practitioners from K-12 educational institutions and experienced educators and professionals. A total of **twelve-hour** learning is required to complete the post-camp outreach activities.

Table 2: GenCyber Cybersecurity Concepts Framework Mapping

	Networks	Python	Cybersec.	Cryptog.	Social Eng.
Keep it Simple			✓		✓
Defense in Depth	✓			✓	
Think Like an Adversary			✓		✓
Confidentiality		✓		✓	✓
Integrity	✓	✓		✓	
Availability	✓				

4 GenCyber Teacher Academy Curriculum

4.1 Curriculum Design

Learning will take place through experiential learning modules and hands-on laboratory exercises. The complete list of Cybersecurity Concepts will be covered and the learning outcomes will mainly be limited to **knowledge** and **comprehension** levels on the Bloom's taxonomy [11]. We strongly believe teachers must have a level of knowledge beyond remembering, recall, and understanding to deliver meaningful instruction on cybersecurity. They must have a well-formed idea of what and when they teach new cybersecurity lessons in their K-12 curriculum and how they connect these new lessons to the other content being taught to students.

4.1.1 Module 1 - Network Fundamentals

This module covers the underlying principles and techniques for network and communication security. Practical examples of security problems and principles for countermeasures are presented. *Organization:* Module 1 is broken down into three units. Each unit is associated with laboratory exercises. Unit 1 introduces networking and TCP/IP protocol. Unit 2 presents computer network security. Finally, unit 3 presents WiFi networks & security. Units and laboratory exercises focus respectively on network modelling and scanning, building firewalls, configuration of intrusion detection systems (IDS) and practical work with analyzing the SSL/TLS protocol. This module promotes group work and multi-user cooperative and competitive activities that will be mainly used in laboratory exercises. *Tools:* Module 1 will use Packet Tracer which is a cross-platform visual simulation tool that allows users to create network topologies and imitate modern computer networks. It supports simulation of endpoint, router, switch, firewall and DDS systems [12]. Packet Tracer is free of charge for academic purposes making it easy and convenient for participants to teach network fundamentals and security course plans. *Learning Outcomes:* At the completion of this module, participants will be able to (1) Explain basic networking concepts, (2) Compare and categorize network media and topologies,

(3) Apply security standards to WiFi networks.

4.1.2 Module 2 - Python Programming & Scripting

This module is intended for learners with no or very little prior programming experience. It covers a range of topics, such as data types, control flow, functions, and object-oriented programming. When learners finish this module, they will be able to create Python programs for a variety of applications. The module includes a Caesar Cipher group-based project implementation. *Organization:* This module will include the following units: (1) introduction to python and data types, operators and flow control statements, (2) data structures and object-oriented programming, (3) Project implementation of Caesar Cipher. Module 2 also provides a cyber ethics overview of the ACM Code of Ethics and Professional Conduct, the IEEE Code of Ethics, and the Computer Ethics Institute's Ten Commandments. *Tools:* This module will use Replit.com, a free online editor allowing learners to code, collaborate, compile, run, share, and deploy Python from a simple web browser [13]. *Learning Outcomes:* Upon successful completion of this module, learners will be able to write Python programs involving basic variable types, common operators, and operator precedence; apply control structures and import libraries and use functions and methods; and use object-oriented programming principles to write code that is easy to read and maintain.

4.1.3 Module 3 - Cybersecurity Awareness

This module addresses the rise in reliance on digital equipment and programs to manage our daily lives, including the transmission and storage of personal information. It demonstrates how an effective cybersecurity awareness is one of the most important steps toward increasing online safety. *Organization:* Module 3 is organized in three units. Unit 1 contains interactive components that include an overview on cybersecurity and sensitive information, information storage, sanitation and disposal, breaches, incidents, and reporting. Unit 2 includes the Rules of Behavior (RoB) and highlights avoiding phishing attacks, email encryption, device locking,and secure mobile connectivity. Unit 3 presents common cybersecurity breaches, incidents and reporting. *Tools:* This module uses slides, questions bank, scenarios, videos, printable handouts, infographics, and links for open access reliable external websites and materials. *Learning Outcomes:* After completion of this module, participants can (1) discuss the unique challenges in the field of cybersecurity that differentiate it from other design and engineering efforts; (2) identify the goals and summarize the overall process of threat modeling; (3) predict and prioritize some potential threats (who might attack it and how) and the human impacts of those threats.

4.1.4 Module 4 - Cryptography

Cryptography is an indispensable tool for protecting information. In this module participants will learn the inner workings of cryptographic systems and how to correctly use them in real-world applications. The module begins with a detailed discussion of how two parties who have a shared secret key can communicate securely when a powerful adversary eavesdrops and tampers with traffic. Next, it discusses public-key techniques that let two parties generate a shared secret key. Throughout the module participants will be exposed to many exciting open problems in the field and work on fun programming projects. *Organization:* Module 4 is organized in the three units. Unit 1 introduces stream and block ciphers. Unit 2 presents message integrity and authenticated encryption. Unit 3 highlights basic key exchange and public-key encryption. *Tools:* Module 4 will use teaching materials, examples, games and assessment artefact from the popular textbook: *Introduction to Modern Cryptography* [14]. *Learning Outcomes:* After the completion of this module, participants will be able to (1) describe basic principles of cryptography and general cryptanalysis, (2) recognize the concepts of symmetric encryption and authentication, and (3) compose, build and analyze simple cryptographic solutions.

4.1.5 Module 5 - Social Engineering

This module explores the human side of cybersecurity: how social engineering attacks work and why they are important to a good threat model. It encourages participants to think about how they verify identity and truthfulness over different communication channels and how those different verification processes can be manipulated by someone who wants to run a scam. *Organization:* This modules is organized in three units. Unit 1 provides an overview on social engineering. Unit 2 introduces common phishing techniques. Unit 3 outlines malicious software. *Tools:* Module 5 will use teaching materials, videos, examples, case studies, simulated attacks. *Learning Outcomes:* Upon completion of this module, participants can (1) define social engineering and the types of attacks associated with it, (2) recognize the techniques to avoid such attacks.

4.2 Learning Outcomes Assessment

The assessment of the participating teachers' learning is an essential means of demonstrating each participant has met the goals of our GTA program and identifying areas for improvement in the proposed curriculum. Our GTA assessment plan a is a three-tier structure that includes: (1) *formative*, *interim*, and *summative* assessments. *Formative assessment* occurs in the short term with prompt feedback from instructors. Example of activities supporting formative assessment include self-assessment quizzes, essay assignments, and

discussion forums in the pre, post, and outreach phases. During the summer camp, daily warm-up and wrap up sessions are used. These sessions improve learners' retention of covered concepts and highlight their relationship with the new module. Moreover, reflection session scheduled at the end of each day of the summer camp allow for engagement with learners through discussions facilitated by the lead instructor and the K12 pedagogy expert. The *interim assessment* aims to give learners the opportunity to demonstrate understanding of material and concepts. Each module includes a set of hands-on laboratory exercises, homework assignments, and group-based projects implementation. The prompt feedback from instructors helps identify gaps in instruction and participants' learning. In addition, the participating teachers engage in course planning and design, development, and validation during the summer camp. Feedback from the lead instructor and the K12 pedagogy specialist help improve their course plan and increase the success of their implementation. *Summative assessment* is performed by the GTA team, upon the completion of the summer camp, to identify strengths and weaknesses of the proposed curriculum and potential future improvements. Examples of summative assessment include the presentation of the produced course plans elaborated by the participants during the summer camp and refined in the post outreach program supported by our GTALC events. To conclude, our integrated assessment plan aims to build participating teachers' confidence and readiness to teach cybersecurity in high schools.

5 Results

Guided by the recommendations of Creswell [15], a survey approach was used to investigate the impact of the week-long GenCyber Teacher Academy on the technology, pedagogy, and content knowledge of the participating grades 9 to 12 teachers. Survey research was the preferred method of data collection because of its economy, rapid turnaround time, and the standardization of the data [16]. Participating teachers completed pre-program, summer program, and post-program surveys. Figures 1, 2, 3, and 4 contain relevant entry and exit survey results.

5.1 Evaluation

The GenCyber Academy took place from August 8-12, 2022, on the campus of The University *of* New Haven in Connecticut. Twenty-five participants, twelve men and thirteen women, participated in the program and were selected from an applicant pool of 78 current high school teachers. The program ran daily from 8:00 AM to 5:00 PM and all participants were present each day. The

majority of each day was used for instruction and hands-on activities, with the remainder used for the participants to plan lessons for their classrooms.

All participants completed a pre-camp survey to ascertain their knowledge and experience with topics in cybersecurity, the relevant data is shown in Figures 1 and 2. The participants then completed five asynchronous modules through the learning management system site prior to the start of camp. After the five in-person days of camp, participants completed an exit survey to assess knowledge and learn about their perceptions of the camp experience.

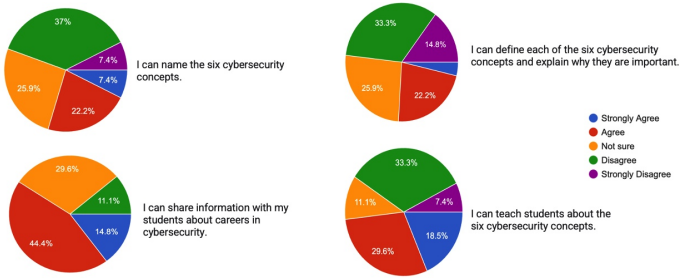


Figure 1: Entry Survey - *Mastery of Cybersecurity Concepts*

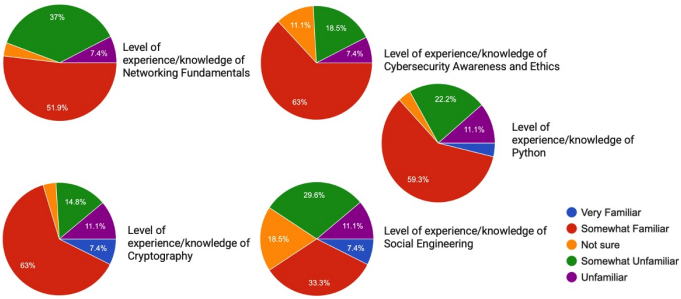


Figure 2: Entry Survey - *Mastery of GTA Curriculum*

One of the primary goals of the GenCyber program was to ensure that the selected high school educators (participants) learned the six cybersecurity concepts. The expectation was that at the conclusion of the camp the participants would be able to name the concepts, define them, and teach them to their high school students. As the entry-survey in Figure 2 shows, only seven of the twenty-five participants were "*somewhat familiar*" or "*very familiar*" with the principles and their definitions. At the conclusion of the program all twenty-five participants were confident that they could name, define and teach them

to their high school students. This was further shown throughout the week by their performance on Kahoot [17] quizzes and in conversations in which the participants named and applied the principles. Several participants also created lesson plans to help their own high school students learn the principles and recognize how they were critical to maintaining cybersecurity. Initially, eleven out of the twenty five participants were unsure or disagreed with the statement "*I can share information about careers in cybersecurity with my students.*" At the conclusion of the camp all the participants agreed or strongly agreed with that same statement.

Overall, the participants showed gains in their confidence in teaching the different concepts they learned about during the week. Figure 3 shows they reported the most confidence in teaching Cybersecurity and Ethics and Social Engineering and the least confidence in teaching Networking. The module Network Fundamental was consistently the area that the teacher participants reported being the least confident in understanding during the camp and as the most difficult concept. In the open response section several participants thought that the Networking day was difficult to follow as the material was complex, very in depth and moved too fast for novice learners. This module is being revised for 2023 to simplify the content and add a hands-on lab that requires no technology. In reviewing overall participant attitudes about their experience, the results indicate that the GTA program was successful. All but one participant agreed or strongly agreed with the statement "I learned a lot about cybersecurity."

Given this data, it is clear the participants valued the experience, felt that they had learned a lot, and would participate in more cybersecurity activities and would want others to have a similar opportunity. Participants felt that they learned enough about careers in cybersecurity to help their high school students prepare for a career in the field, could help them decide if cybersecurity is a good career path, and believe that cybersecurity is a good career option for their students. To further validate the self-reported data of gains in confidence in cybersecurity topics, the team reviewed two detailed lesson plans from each teacher to ensure content was accurate, relevant and aligned with teaching the concepts of cybersecurity. The depth of knowledge gained was evident in lesson plans as many teachers wrote lesson plans that incorporated basic Python programming, the six cybersecurity concepts, and case studies shared throughout the week. Teachers adapted content and activities from the GTA camp to use in their classrooms and demonstrated their new understanding by describing these lessons and later sharing their implementation results. Additionally, they demonstrated their new understanding by creating their own lessons for their classroom, which provided evidence of their learning. Besides the Likert scale questions, teachers were given an opportunity to answer some

open-ended questions to provide feedback to the program facilitators. Participants most enjoyed the lessons on cryptography including using the Scytale cipher and learning how to pick locks. They also enjoyed learning about social networking. A small number of participants reported that Python was their favorite activity of the week. When asked to report something they learned that they believe everyone should know the common themes were that people need to know how important cybersecurity is, how best to protect yourself (or your business/company) from cybersecurity breaches, and how common cybersecurity attacks are. Many participants also mentioned that others should understand social engineering and how it impacts humans and their behavior.

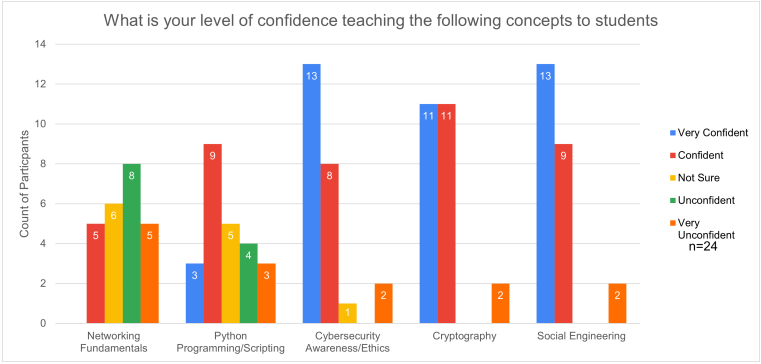


Figure 3: Exit Survey - *Participant Confidence in Teaching*

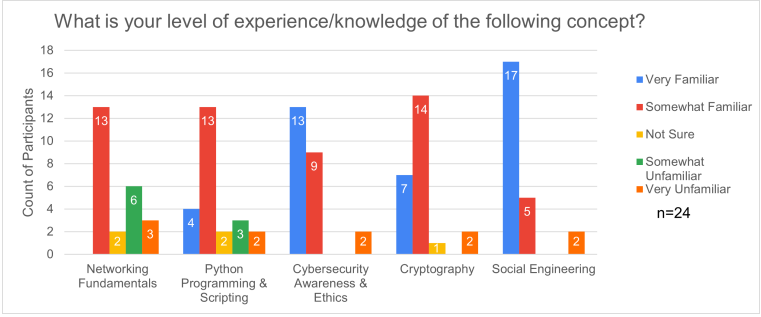


Figure 4: Exit Survey - *Participant Experience and Knowledge*

5.2 Future Improvements

Several recommendations arose throughout the week and on the surveys that can help improve the program for the future. One recommendation is that on the day that Python programming is taught it would be helpful to break the teachers into two or three groups based on their experience and knowledge of coding. Participants that were familiar with Python or other coding languages became default tutors to those with no experience and felt that they did not learn much in this session. On the other hand, those new to coding, programming, and Python felt the day was very challenging and moved too quickly. Small groups based on ability could also be used on the day networking is taught. Some felt it was overwhelming and moved too quickly, and others would have liked to go deeper and move faster. Another suggestion that can be implemented is to give students more opportunities to mix and mingle with each other. Throughout the week students sat in pods of five that stayed consistent day to day. In the future, students could change groups a few times throughout the week.

Participants had some additional ideas that would make the program even more beneficial. The access to the learning management system was helpful to have all the course materials. They recommended that a running list of resources that come up throughout the week be kept somewhere in Classroom so that they can refer to it in the future. Additionally, allowing participants to add resources that may not be mentioned but are related would be helpful. Participants also asked if they could, in the future, have access to each other's lesson plans. Since all the teacher participants wrote at least two lesson plans, they would have access to a bank of fifty lessons.

6 Conclusion and Future Work

In conclusion, the GenCyber Teacher Academy at The University *of* New Haven was successful. Participants felt that the camp was worthwhile, demonstrated learning of cybersecurity concepts, and put their learning into practice by designing lesson plans for their classes. Concrete, feasible recommendations were made to improve the program for the future.

GenCyber Teacher Academy will contribute to advance cybersecurity culturally responsive educational practices and address the critical shortage of qualified high school teachers in Connecticut and nationwide. It will establish a sustainable and scalable learning community and assess its impacts within, between and across schools to continuously improve cybersecurity education in Connecticut and the rest of the United States.

Our GenCyber Teacher Academy Learning Community (GTALC) initiative is well aligned with the Connecticut Computer Science/Cybersecurity State

Plan [18]. This plan has been defined by our partners, the *Computer Science Advisory Group* in close collaboration with the *Connecticut Council for Education Reform - ReadyCT* and *EdAdvance*. It provides a statewide vision to assist in the coherent implementation of K–12 cybersecurity instruction and opportunities for all Connecticut K–12 students to engage in high-quality cybersecurity education.

To conclude, the GTA program will continue to enhance and promote the development of cybersecurity curricula in low-performing high schools and develop a diverse, globally competitive cybersecurity and computing workforce.

7 Acknowledgements

This work was jointly supported by the National Security Agency and the National Science Foundation GenCyber Program under Grant Number 21A-CT-UNHx-UV-T1. Thanks to Jessica Berrios for assistance with graphs.

References

[1] Connecticut Examiner, "Computer science education expanding in k-12," 2019. [Online]. Available: https://ctexaminer.com/2019/12/04/computer-science-education-expanding-in-k-12/

[2] Senate and H. of Representatives in General Assembly, "Substitute senate bill no. 962, public act no. 15-94," https://www.cga.ct.gov/2015/ACT/pa/pdf/2015PA-00094-R00SB-00962-PA.pdf, 2015.

[3] K. Dell, N. Nestoriak, and J. Marlar, "Assessing the impact of new technologies on the labor market: Key constructs, gaps, and data collection strategies for the bureau of labor statistics," 2020.

[4] T. Gais, B. Backstrom, J. Frank, and A. Wagner, "The state of the connecticut teacher workforce." *Nelson A. Rockefeller Institute of Government*, 2019.

[5] Connecticut State Department Of Education, "Connecticut teacher shortage areas report 2020–2021," 2020. [Online]. Available: https://portal.ct.gov/-/media/SDE/Performance/Research-Library/ConnecticutTeacherShortage-Areas-Report-2020-21.pdf?la=en

[6] Connecticut State Department of Education, "Opportunity district," 2021. [Online]. Available: https://portal.ct.gov/-/media/SDE/Alliance-Districts/Opportunity-District.pdf?la=en

[7] "Code.org's approach to diversity and equity in computer science." [Online]. Available: https://code.org/diversity

[8] C. S. Sanger, "Inclusive pedagogy and universal design approaches for diverse learning environments," in *Diversity and Inclusion in Global Higher Education*. Palgrave Macmillan, Singapore, 2020, pp. 31–71.

[9] "Current perspectives and continuing challenges in computer science education in u.s. k-12 schools," 2020. [Online]. Available: https://csedu.gallup.com/home.aspx

[10] GOOGLE/Gallup, "Diversity gaps in computer science: Exploring the underrepresentation of girls, blacks and hispanics," 2016. [Online]. Available: http://services.google.com/fh/files/misc/diversity-gaps-in-computer-science-report.pdf

[11] B. S. Bloom, D. R. Krathwohl, and B. B. Masia, "Bloom taxonomy of educational objectives," in *Allyn and Bacon*. Pearson Education, 1984.

[12] S. R. Javid, "Role of packet tracer in learning computer networks," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 3, no. 5, pp. 6508–6511, 2014.

[13] S. Cooper, B. Clinkscale, B. Williams, and M. Lewis, "Exploring the impact of exposing cs majors to programming concepts using ide programming vs. non-ide programming in the classroom," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020, pp. 1422–1422.

[14] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC press, 2020.

[15] J. W. Creswell and C. N. Poth, *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, 2016.

[16] E. R. Babbie, *The practice of social research*. Cengage learning, 2020.

[17] Y. Basuki and Y. Hidayati, "Kahoot! or quizizz: The students' perspectives," in *Proceedings of the 3rd English Language and Literature International Conference (ELLiC)*, 2019, pp. 202–211.

[18] Connecticut State Board Of Education, "Connecticut computer science plan," 2018. [Online]. Available: https://portal.ct.gov/-/media/SDE/Computer-Science/Connecticut_Computer_Science_State_Plan_FINAL.pdf

A Liberal Arts Undergraduate Robotics Major*

Chiranjivi Lamsal[1], Michael Walters[1], Kevin McCullen[2]
[1]Physics Department
[2]Computer Science Department
State University of New York, College at Plattsburgh
Plattsburgh, NY 12901
{clams001,mwalt003,kmccu006}@plattsburgh.edu

Abstract

We describe a Liberal Arts undergraduate Robotics major. We see Robotics as an emerging and innovative field that provides excellent career opportunities, as well as the potential for graduate work. Within our state college system, a Liberal Arts Robotics program is a unique offering. Regionally, strong Middle School and High School level Robotics competition teams provide a pool of potential students. With these motivations, our institution implemented a Robotics major that is a joint effort between the Physics and Computer Science departments.

The program is built on a foundation of experiential and project work that utilizes single-board computers and microcontrollers such as the Arduino and Raspberry Pi. Consumer and maker-grade hardware is inexpensive and (often) open source or open hardware; however there are functional limitations, support is often crowdsourced, and reliability can be problematic. The utilization of custom designed 3-D printed components for robots increases the range of problems that can be solved while keeping costs low. The cornerstone of the experiential approach is the inclusion of four project courses, one per year. These project courses increase in sophistication and difficulty, focus on hardware/software interaction, while solving non-textbook and unique problems. We report on the composition of the curriculum, with special attention to the interdisciplinary nature and cross-departmental cooperation; the hurdles of

putting together the curriculum at a Liberal Arts institution, the pedagogical challenges of working with consumer and maker-grade hardware and open software, and a robotics platform to address these challenges.

1 Demand for Robotics

There is a robust and growing market for Roboticists, with over 132,000 open jobs [1]. FIRST (For Inspiration and Recognition of Science and Technology) runs five different levels of robotics competitions for ages 4 through 18, and had 679,000 student participants in 2019-2020 [3]. In 2020, it was estimated that there were 2.7 million industrial robots in use worldwide [6]. That does not include robots seen outside an industrial context. With robotic french-fry cooks, pool cleaners, vacuum cleaners, and self-driving cars, there are very few aspects of our lives that are not touched by robots.

But what is a Robot? Definition number two from Merriam-Webster [5] captures best the subject area for our major:

1. "a device that automatically performs complicated, often repetitive tasks (as in an industrial assembly line)"
2. "a mechanism guided by automatic controls"

A major in Robotics offers an opportunity to address a relatively new and growing STEM field, and one that many students find interesting and exciting. However, Robotics is typically seen as a field of Engineering, while we are at a Liberal Arts school. Our challenge was to define and implement a non-Engineering Robotics major. By "non-Engineering" we mean that we carry Middle States Accreditation, rather than ABET (Accreditation Board for Engineering and Technology). This is largely dictated by the constraints placed upon our school by our state college system, as we are not one of the designated Engineering schools in the system.

Key enablers of our major were the pre-existing 3+2 engineering program, hardware and software oriented Robotics minors, and the growth of inexpensive and widely available sensors, servos and motors, and single-board computers. We were able to utilize these stepping stones to create a Robotics major within a Liberal Arts college.

Robotics and AI are very useful in creating effective learning environments. These environments motivate students to learn the necessary skills and knowledge.

Through the completion of robotics projects, students understand “the social construction of integrated networks of authentic Science, Technology, Engineering, and Mathematics (STEM) knowledge centered around “Big Ideas” of and about STEM” [9]. Involvement of students in robotics activities also

plays an important role in bringing students from marginalized community to college, in general, and to the STEM fields in particular.

2 History

Our institution has an existing 3+2 Physics and Engineering program, conducted jointly with several neighboring engineering universities. Upon completion of three years at our school and two years at the collaborating institution, the students are awarded a Bachelor of Arts degree in Physics from our school and a Bachelor of Science degree in Engineering from the partner institution.

The Physics and Computer Science programs offer two minors in Robotics, one with an emphasis on Physics and one with an emphasis on Computer Science. The common piece of both minors is a four credit Introduction to Robotics course and a three credit course in Artificial Intelligence, both at the Junior level. Common prerequisites for the minor includes Calculus I, Calculus II or Linear Algebra, Physics I, Discrete Mathematics, and Introduction to Programming (Python).

3 Cross-Departmental Cooperation

The expertise to offer a Robotics major does not fit neatly into any existing department [11]. The Physics department had existing offerings in design (as part of the 3+2 program), as well as circuits and electromagnetics, and digital logic. They also were more experienced at developing, staffing, and funding hands-on laboratory courses. The Computer Science department had existing course offerings that were typical for a software oriented program (i.e. not a hardware focused Computer Engineering curriculum).

Both programs needed to develop new courses to complete the program. One of the key enablers was the collegial cooperation that went into this process. Some courses could conceivably have been offered by either department. Skills and available resources made it relatively easy to decide which department would offer which course. It was understood from the beginning that some faculty would cross-over to teach courses as needed.

The courses developed on the Computer Science side included an Embedded Systems course (using the Raspberry Pi), an introductory (100 level) Physical Computing course (using the Arduino), an Image Processing course, and a Distributed and Autonomous Systems course.

On the Physics side, an Introduction to Robotics course already existed. Additional courses developed included Advanced Digital Design, Sensors, Control Theory, and Robot Kinematics and Dynamics.

The cooperation between the two departments has been a key factor in our success. It helps in that there are faculty with common backgrounds, and prior experience working as engineers before moving into academia. When the university reorganized space to accommodate growth in the Computer Science department, the Physics and Computer Science departments were moved from separate buildings, together into adjacent hallways. It was also helpful that both departments had faculty who had pre-existing interest in Maker culture; such as the Raspberry Pi, Arduino, and 3-D printing; and FIRST Robotics competitions. These Maker tools form the backbone of the hands-on components of the curriculum.

4 Needs of a Robotics Curriculum

In constructing a curriculum, you have to establish the parameters of what a student needs to learn, and what skills the student needs to acquire. Some fields, such as Mathematics or Electrical Engineering, have long established basic curriculums. Compared with many STEM subjects, Robotics is a relatively new field, sitting at the intersection of multiple STEM disciplines [11].

The Robotics curriculum consists of Mathematics, Mechanical Engineering, Industrial Engineering, Electrical Engineering, and Computer Science .

The fundamentals of Robotics, in the plan-compute-implement loop, place an emphasis on sensing, computation, and actuation. Small micro-controller based kits (such as the Arduino [2]) can be purchased for under $50, and include all of the necessary pieces to sense (such as a photodiode and temperature sensor), compute (a microcontroller), and act (servos and LEDs). The ability to go from "zero to blinky lights" very quickly with these systems makes them engaging and interesting [21].

A robot senses; our curriculum needs to teach the fundamentals of sensing, the limitations of sensors, precision versus accuracy, and topics such as analog to digital conversion. A robot computes; robotic calculations span a range from simple for-loops on inexpensive micro-controllers to ROS (Robot Operating System) [17] simulations. A robot acts; the robot may be driving actuators, moving in 2-D and 3-D spaces, estimating and tracking its location (for example monitoring wheel encoders). The robot may be sensing while acting, which may be computationally expensive. The physical world looks very slow to a computer, but the hardware platform used for an inexpensive product may be "just fast enough" to keep up, requiring a different set of programming skills than building a webpage in Javascript.

A Roboticist needs to have a clear understanding of both hardware and software: the power and limitations of the hardware components (sensors, actuators, etc.); the ability to write software that interacts with the physical

world efficiently, accurately, and effectively; and skills need to be cultivated in software development, ranging from low level programming in resource constrained environments to machine learning and vision in high performance environments. The curriculum needs to address all of these factors, and it needs to provide ample opportunity to apply them in practical projects. Experience is the best teacher. A robot that continues to fail at its task because a key subsystem is using too many resources is a valuable and important lesson.

The taxonomy of Robotics that we worked with revolves around the common view that Robotics includes a loop that senses the environment, plans or computes an action, and then implements the action that changes the environment. The planning may be individual or collective (swarm); autonomous (algorithmic or through machine learning) or interactive (person-in-the-loop). The sensing may be passive or active, it may be local or remote. The action may be individual or collective, local or remote, physical or simulated.

5 Our Robotics Curriculum

Liberal Arts or Engineering? There has been, in recent years, a great deal of discussion of this topic. The argument over whether you need "liberal arts thinkers" instead of people focused on STEM has been described as a "false dichotomy" [10]. In The New York Times, Vivek Wadhwa states that "In the two companies I founded, I was involved in hiring more than 1,000 workers over the years. I never observed a correlation between the school of graduation or field of study, on one hand, and success in the workplace, on the other" [23].

The "S" and "M" portions of STEM have long been associated with Liberal Arts programs and degrees. Robotics arguably falls squarely into the "T" and "E" portions of STEM. This could be viewed as a distinction without a difference. A Liberal Arts Robotics major can be constructed that has the same depth and breadth as an Engineering Robotics major. The key is defining the contents of the curriculum and having the resources to offer that curriculum.

Shibata et al.[18] looked at the question of what courses comprise a Robotics curriculum. They looked at 19 Robotics programs in Japan, and categorized courses by title. In their Table II, they identified the top 15 common categories (those offered by 10 or more programs). Our program includes seven of them (including the last, which arguably includes the others):

- Controls
- Programming
- Electronic or Electrical Circuits
- Sensors
- Artificial Intelligence or Intelligent Systems
- Robot Vision

- Robotics

Several courses have ambiguous titles that may or may not match our curriculum. The ambiguous topics include "Mechanism", which may map to our Kinematics course; and "Technical drawing/design" which may map to our Fundamentals of Engineering Design course. Our program does not require courses such as Strength of Materials, Mechatronics, Material Processing, Fluid Mechanics, Instrumental Engineering, or Simulation Engineering.

How does a Liberal Arts Robotics program differ from an Engineering Robotics program? The clearest distinction is the General Education Foundation course requirements. As a Liberal Arts program, our students must complete a substantial set of courses that includes history, social sciences, arts, writing, and others. The science and math requirements of the major easily exceed the General Education requirements.

That means that students in the program have fewer free electives to work with. For that reason, we only offer a Bachelor of Science degree, not a Bachelor of Arts. For a student with good mathematics preparation, the General Education requirements will be approximately 30 to 34 credits. The Robotics major itself includes 67 to 76 credits (nominally 70 credits). The remainder of the 120 credits (in the range of 10 to 23 credits) are free electives. In order to keep the number of required courses (and prerequisites) to a reasonable number, some math concepts, particularly in differential equations and systems engineering, are covered within the courses that require them. Those planning to pursue graduate studies are directed by advisement to take the equivalent higher level math courses.

The curriculum contains two tracks: Hardware Application and Programming. Each year includes a **project based (experiential) course**. Each track contains *four unique courses*. These courses are annotated as such in the sample schedules shown in Tables 1 and 2). [1]

The "Advanced Electives" include the courses in the opposing track, plus:

- Computer Organization
- Artificial Intelligence
- Distributed and Autonomous Systems
- Image Processing

The longest path through the major terminates at the Robotics Senior Project course. This path requires a minimum of six semesters. As a relatively small school, limitations on course frequency leave very little slack in the schedule. Students without prior math preparation who are unable to take

[1] The C Programming course shown in the tables is one credit, all others - except Calculus I/Trigonometry - are 3-4 credits

Table 1: Sample Schedule for Hardware Application Track

	Fall		Spring	
Freshman	Calculus I/Trigonometry Physics I Introduction to UNIX/Linux **Physical Computing** General Education	16	Calculus II Physics II General Education General Education General Education	17
Sophomore	Introduction to Programming Linear Algebra Discrete Mathematics General Education General Education	16	C Programming Object Oriented Programming **Introduction to Robotics** *Fundamentals of Engineering Design* General Education General Education	16
Junior	Physics and Applications of Sensors Ethics *Electronics for Scientists* General Education Free Elective	16	**Intermediate Robotics Laboratory** Advanced Digital Design *Control Systems Engineering* Advanced Elective Free Elective	15
Senior	*Robotics Kinematics and Dynamics* Advanced Elective Free Elective Free Elective	12	**Robotics Senior Project** Free Elective Free Elective Free Elective	12

Table 2: Sample Schedule for Programming Track

	Fall		Spring	
Freshman	Calculus I/Trigonometry Physics I Introduction to UNIX/Linux **Physical Computing** General Education	16	Calculus II Physics II General Education General Education General Education	17
Sophomore	Introduction to Programming Linear Algebra Discrete Mathematics General Education General Education	16	C Programming Object Oriented Programming **Introduction to Robotics** *Data Structures and Algorithms* General Education General Education	16
Junior	Physics and Applications of Sensors Ethics *Design and Analysis of Algorithms* General Education Free Elective	16	**Intermediate Robotics Laboratory** Advanced Digital Design *Software Engineering* Advanced Elective Free Elective	15
Senior	*Embedded Systems* Advanced Elective Free Elective Free Elective	12	**Robotics Senior Project** Free Elective Free Elective Free Elective	12

Calculus I in their first semester may find it difficult to complete the program in the correct order in four years (without deviations or exceptions).

6 The Role of Single Board Computers (SBCs) and a Robotics Teaching Platform

SBCs are key to the robotics curriculum. They allow for an inexpensive, compact, wireless, and energy efficient platform that the majority of our robots are based upon. These advantages are mitigated somewhat by the disadvantages such as low computational power and the use of cheaper electronics that can cause stability issues.

To leverage this in a robot one must either use a robot platform from a vendor or develop your own. We tried vendor platforms but found each had its own limitations. Using a Pololu Zumo [14] did not give us an easy addition of computer vision. In 2015 we tried to use a BrickPi [4], which at its heart is a Raspberry Pi with a I/O motor shield, as the core of our platform. It allowed students to make their own robots using LEGO parts, sensors, and motors, but it was missing simple things like the ability to accurately use encoders. After evaluating several vendor platforms (Pololu Zumo [14] and BrickPi [4]), we decided to develop an internal platform leveraging our ability to design and 3-D print robot bodies and the parts needed to attach various motors and sensors to them. This also meant the addition of a microcontroller to replace the I/O motor shield that was a part of the BrickPi.

The teaching robot platform that was developed, KIF (Keep It Fun), Figure 1, is based on the Raspberry Pi [13]. It is constantly in revision as necessitated by hardware cycles and parts availability. KIF can be customized for each class if needed. The downside of developing a custom robotics platform is that you are your own technical support. Each revision takes time to test and vet solutions. To help mitigate this frustration, keeping the current build focused on the pedagogy to be delivered is necessary. This focus limits the scope and helps keep the pedagogical goal in the center.

The Raspberry Pi installed on KIF runs Raspberry Pi OS, which is a port of Debian, with the full GUI desktop installed. To log in wirelessly, RealVNC [16] is activated which is included with the port of Debian. The current version of the Arduino IDE, used to program the microcontroller, needs to be installed. Also required are the corresponding hardware libraries for the various motors and sensors to be included in KIF's particular build. For computer vision, OpenCV [12] is installed for the current version of Python. This used to be an arduous task involving many hours of downloading and compiling different pieces of the library, but now there is a version that is a ready-to-install Python library. For AI and machine learning, PyTorch [15] and TensorFlow [20] are

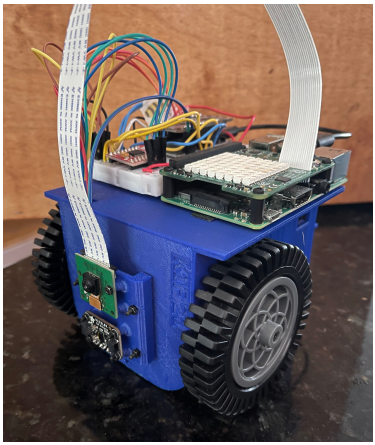


Figure 1: KIF: Keep It Fun

also installed.

These last pieces of software highlight the lack of processing power of the Raspberry Pi. Running tasks like a sprite search in an image can take seconds to a minute to finish depending on the problem's complexity. This needs to be taken into account when attempting "real time" applications. In the above circumstance, the students were allowed time to do the calculation, the robots all moved for five seconds, then the next processing phase took place. The principles are the same as with faster computers, so pedagogically they are still learning the same objectives, just the timeline must be adjusted to allow for the limitations of the SBC.

Traditionally KIF used an Arduino Mega 2560 [2] as its microcontroller. This was selected for the abundance of analog and digital I/O as well as the larger amount of memory. The problem with the Mega is that it is very slow and does not handle using interrupts on multiple channels very well. KIF, configured with motors and encoders, would tax the ability of the board. That is why KIF historically ran with stepper motors, since they do not need to use interrupts. To allow for the use of motors with encoders, KIF replaces the Arduino Mega with either an Adafruit Feather [8] or a Teensy [19]. Both these microcontrollers are fast enough to handle the multiple interrupts while doing other things like getting data from sensors or making calculations on where the robot is. The current KIF uses an Adafruit Huzzah32 Feather as its microcontroller. In 2021-2022, shortages of Raspberry Pi systems, led to more use of the Adafruit Feather for the main processing component.

7 Experiential Courses

The four experiential courses form a natural progression of increasing difficulty level.

Physical Computing focuses on the use of the Arduino with simple sensors and servos. The course content lies between the "CS0" and "CS1" content as defined by the ACM and IEEE [7]. Students begin with the basics of making an LED blink and progress through a series of hands-on projects that introduce the basics of C programming, wiring components together, working with simple electrical components, reading sensor values, and driving servos. While the course currently uses the Arduino UNO with a kit of components [22], there are a number of components that could be used. The limited memory on the UNO (32KB of Flash, 2KB of SRAM) helps prevent pedagogical scope creep.

Introduction to Robotics focuses on autonomous robotics using a basic KIF as the robot platform. Topics covered at a basic level are sensors, control theory, motion planning, machine vision, localization, deep learning, and kinematics. Each topic is explored using KIF except kinematics which uses a small robot arm. The techniques build throughout the semester. The final project consists of showing KIF a sign and having KIF move from sign to sign, moving in the correct direction based upon the signs' content until KIF reaches the appropriate end point.

Intermediate Robotics Lab is a group project-based course. Teams of students are given a challenge, and they focus on that challenge throughout the semester. Some of the previous challenges have been robots playing zombie tag, reading a map to follow the "roads" that are allowed while avoiding ones that are forbidden, and running a line maze using information from the environment to finish the task in the allotted time. These projects are picked to highlight different aspects of robotics, trying to give students a choice of following their interests. Here KIF is modified with custom pieces to help the students complete the tasks. Many times, the students will do their own CAD designs and 3-D print the needed parts.

Robotics Senior Project is where the student defines, with approval of their mentor, what they are attempting to complete for the semester. This allows the student to further refine the skills they hope to showcase on their graduate school application or resume/job interview. This is an intensive course leaving a lot of room for the students to explore what they are capable of doing. A fast try/fail cycle is encouraged. Typically, the student will not get as far as they had planned since they are usually ambitious with project scope. The grade is based on following good engineering principles and remaining focused on the task at hand.

8 Conclusions

We are a predominantly undergraduate institution, a Liberal Arts state college, and a non-Engineering school. We serve a student body that is very diverse with a high population of first-generation students. We offer an entry point to Robotics for a different population than R1 engineering schools.

The major was approved in early 2018, too late for the 2018-2019 academic year. Enrollment was 18 students in Fall of 2019, 21 students in Fall of 2020, 24 in the Fall of 2021, and 16 in Fall 2022. The numbers for 2021 and 2022 are likely impacted by reduced admissions during the two years of Covid. To date, the program has graduated 10 students, placing several in the Robotics industry and one in graduate school studying robotics.

We continue to fine-tune the curriculum structure and course content. Students have overwhelmingly chosen the hardware track, and there has been some discussion of focusing the programming track on Machine Learning and Artificial Intelligence, and renaming the track to emphasise the ML/AI focus.

Students with less prior math preparation face a challenge, and we continue to seek ways to increase the flexibility in the major.

We would like to explore ABET accreditation in the future, but not at this time.

Development continues on KIF, with a plan of making it freely available as an open educational resource.

References

[1] *132,000+ Robotics Engineer jobs in United States (2,881 new)*. en. 2022. URL: https://www.linkedin.com/jobs/robotics-engineer-jobs (visited on 08/09/2021).

[2] *Arduino - Home*. 2022. URL: https://www.arduino.cc/ (visited on 12/10/2022).

[3] *At A Glance*. en. Aug. 2022. URL: https://www.firstinspires.org/about/at-a-glance (visited on 12/09/2022).

[4] *BrickPi*. en-US. 2022. URL: https://www.dexterindustries.com/brickpi/ (visited on 12/10/2022).

[5] *Definition of ROBOT*. en. 2022. URL: https://www.merriam-webster.com/dictionary/robot (visited on 12/09/2022).

[6] *Global population of industrial robots at record high, says IFR report*. en-gb. Sept. 2020. URL: https://industryeurope.com/api/content/126fa4ea-ff08-11ea-9c47-1244d5f7c7c6/ (visited on 08/09/2022).

[7] ACM and IEEE. *Computer Science Curricula 2013*. Tech. rep. The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society, 2013, p. 518. URL: https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf (visited on 12/13/2022).

[8] Adafruit Industries. *Adafruit Feather HUZZAH with ESP8266 - Loose Headers*. en-US. 2022. URL: https://www.adafruit.com/product/2821 (visited on 12/10/2022).

[9] Myint Swe Khine, ed. *Robotics in STEM Education: Redesigning the Learning Experience*. en. Springer International Publishing, 2017. ISBN: 978-3-319-57785-2. DOI: 10.1007/978-3-319-57786-9. URL: https://www.springer.com/gp/book/9783319577852 (visited on 12/11/2022).

[10] Sergei Klebnikov. *Liberal Arts vs. STEM: The Right Degrees, The Wrong Debate*. en. Section: Education. June 2015. URL: https://www.forbes.com/sites/sergeiklebnikov/2015/06/19/liberal-arts-vs-stem-the-right-degrees-the-wrong-debate/ (visited on 12/12/2022).

[11] Rachid Manseur. "Robotics engineering program and curriculum development". en. In: *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*. Madrid, Spain: IEEE, Oct. 2014, pp. 1–5. ISBN: 978-1-4799-3922-0. DOI: 10.1109/FIE.2014.7044426. URL: http://ieeexplore.ieee.org/document/7044426/ (visited on 12/06/2022).

[12] *OpenCV Home*. en-US. 2022. URL: https://opencv.org/ (visited on 08/10/2022).

[13] Raspberry Pi. *Teach, Learn, and Make with Raspberry Pi*. 2022. URL: https://www.raspberrypi.org/ (visited on 12/10/2022).

[14] *Pololu - Zumo Robot for Arduino (Assembled with 75:1 HP Motors)*. en. 2022. URL: https://www.pololu.com/product/2510 (visited on 12/10/2022).

[15] *PyTorch*. en. 2022. URL: https://www.pytorch.org (visited on 12/10/2022).

[16] *RealVNC® - Remote access software for desktop and mobile | RealVNC*. 2022. URL: https://www.realvnc.com/en/ (visited on 12/10/2022).

[17] *ROS.org | Powering the world's robots*. en-US. 2022. URL: https://www.ros.org/ (visited on 12/12/2022).

[18] Mizuho Shibata et al. "Comparative Study of Robotics Curricula". en. In: *IEEE Transactions on Education* 64.3 (Aug. 2021), pp. 283–291. ISSN: 0018-9359, 1557-9638. DOI: 10.1109/TE.2020.3041667. URL: https://ieeexplore.ieee.org/document/9303482/ (visited on 12/06/2022).

[19] *Teensy USB Development Board*. 2022. URL: https://www.pjrc.com/teensy/ (visited on 12/10/2022).

[20] *TensorFlow*. 2022. URL: https://www.tensorflow.org/ (visited on 12/10/2022).

[21] David Townsend. *Talk Python To Me: MicroPython + CircuitPython*. 2021.

[22] Vilros.com. *Arduino Uno Ultimate Starter Kit + LCD Module*. en. 2022. URL: https://vilros.com/products/arduino-ultimate-starter-kit-lcd-module (visited on 08/13/2022).

[23] Vivek Wadhwa. "Room for Debate: Look at the Leaders of Silicon Valley". en. In: *New York Times* (Mar. 2011). URL: https://www.nytimes.com/roomfordebate/2011/03/20/career-counselor-bill-gates-or-steve-jobs/look-at-the-leaders-of-silicon-valley (visited on 10/12/2022).

Open Questions for Empathy-Building Interventions for Inclusive Software Development*

Kyle L-Messerle, Samuel Malachowsky, Daniel E. Krutz
Department of Software Engineering
Rochester Institute of Technology, Rochester, NY, USA
{klm3580, samvse, dxkvse}@rit.edu

Abstract

Research has demonstrated that much of the software being created today is not sufficiently inclusive, unbiased and equitable. This has been found to frequently result in real-world implications such as prejudice against women or people of color, and software that is inaccessible to people with disabilities. Preliminary research has found that empathy-focused experiential educational activities can be beneficial for not only creating empathy, but in advancing the participant's interest and knowledge retention over traditional non empathy-building interventions. This work will provide a foundational background on the current research in the intersection of experiential learning and empathy-building interventions in computing education. We will also present several important questions that still must be explored, thus serving as the foundation for future work in this area.

1 Introduction

Research demonstrates that we continue to be deficit in creating inclusive and equitable software [9, 16, 18, 56, 79]. Despite the prevalence and demonstrated

capabilities of experiential education [4, 38, 48, 88] and foundational demonstrated benefits of empathy-building interventions [6, 84**?**], the intersections of these topics have not been sufficiently explored. Specifically, we need to investigate and create educational empathy-building interventions to better inform and motivate students to create more inclusive and equitable software. There are several key areas that require further exploration. These include:

1. Understand the benefits and impacts of empathy-creating interventions in experiential computing education.
2. Recognize appropriate methodologies to include empathy-creating interventions in experiential computing education.
3. Understand if experiential empathy creating interventions can help to reduce bias.

Improved knowledge regarding empathy-creating interventions can directly benefit computing education while exponentially benefiting society through the creation of more fair, unbiased, and inclusive software used by the general population [64, 72, 82]. Potential benefits of empathy-building experiential education modules can contribute to the foundational understanding of experiential education from a theoretical and practical perspective, benefiting a variety of topics in computing education (*e.g.,* general computing, accessibility, artificial intelligent/machine learning, autonomy, software engineering, HCI, etc.).

The rest of the paper is organized as follows: Section 2 provides the motivation and guiding theory, while Section 3 presents related works. Section 4 discusses several important crucial questions to be addressed and Section 5 provides a conclusion.

2 Motivation and Guiding Theory

Motivations from Education: Experiential empathy-creating interventions have been explored in various non-computing domains such as in medicine [33, 46, 58, 86], and for creating tolerance in social situations [21]. Unfortunately, the application of these benefits in experiential computing education is inhibited by a lack of understanding regarding: I) A proper implementation framework [78], II) Their specific pedagogical advantages, and III) The most appropriate pedagogical and technical methods for integrating these into computing curriculum [59]. The potential benefits of empathy-building interventions in experiential computing education has been demonstrated in foundational, preliminary research . Despite these encouraging results, there is currently a lack of knowledge that inhibits the implementation and benefits of empathy-creating interventions at institutions across the United States [40, 64].

The application of empathy-building interventions in experiential computing education has been inhibited by both pedagogical and technical limitations . The hypothesis that creating empathy can increase student interest is supported by the PI's preliminary work in this area. Deeper pedagogical questions also exist, such as appropriate intervention inclusion methods and their impacts on empathy's subprocesses ('mentalizing', 'experience sharing', 'empathic concern') [6, 27, 42, 84, 90]. Additionally, technical obstacles must be overcome such as how to properly create an empathy-building experience and how to sufficiently emulate the experiences of other users (*e.g.,* accessibility challenges, racial bias, etc.).

Motivations from the Community: A lack of empathy among software developers has been attributed to the creation of biased, inequitable software [9, 26, 59]. This necessitates the creation of high-quality empathy-creating educational interventions to support the next generation of software developers in creating more equitable software for society. Research has demonstrated that increasing empathy can lead to software that is developed in a more accessible, inclusive and equitable manner [10, 87**?**]. This prior work provides confidence that improving empathy in computing students can yield similar benefits and help them to understand the necessity of creating inclusive software. Unfortunately, there is a lack of an understanding of how to most effectively teach students empathy-related concepts to construct inclusive software. While bias may be created due to unconscious developer actions or by non-human factors (*e.g.,* incoming data in AI/ML [26, 76]), an objective should be to better understand how participants can more appropriately become cognizant of, and properly address biases when developing software.

Recent US government legislation has called for software that is more inclusive and unbiased [1, 2, 3, 26, 76]. There is also a stated educational demand for easily adoptable interventions that will support the creation of more equitable software, such as software that is created with a greater amount of empathy [9, 20, 62]. Increased empathy is expected to result in the creation of software that is more inclusive, equitable and unbiased [59], while also having a positive impact on the developer's career [44]. Empathy is being seen as a greater necessity due to the increasingly globalized nature of society [59]. The demand for software with these attributes will continue to grow as more interactions and tasks are performed online [15, 68]. Preliminary observations have demonstrated that the proposed work has the capability to directly contribute to accomplishing these goals.

Guiding Theory: Research demonstrates that we continue to be deficit in creating inclusive and equitable software [9, 16, 18, 37, 41, 56, 79]. Prior work has demonstrated that increased empathy can lead to software that is developed in a more accessible, inclusive and equitable manner [10, 87**?**]. Empathy

can be developed, frequently through experiential activities [6, 23, 47, 84, 85]; however, there are no known efforts to examine the integration of experiential learning to create empathy in computing education [**?**]. Existing works have demonstrated both the capabilities of experiential learning [4, 38, 48, 88] and in empathy creation [6, 23, 47, 84, 85]. It is surmised that this increased empathy will increase the student's ambition to create more equitable and inclusive software.

General Scientific Barriers: A key challenge is how to accurately create experiential empathy-creating interventions for both instruction and evaluation in a variety of computing courses, ranging from foundational to more specialized courses. While initial work demonstrates the foundational capability of empathy-creating interventions in several offerings of a CS2 course, it has not been widely attempted in other computing curriculum. Although there are various proposed empathy measuring evaluations [34, 35], there do not appear to have been any significant efforts for measuring empathy in computing education, representing another challenge that must be addressed. Ensuring that interventions create empathy and not pity for specific users is another challenge that must be considered.

Preliminary Efforts: Foundational work has demonstrated the potential benefits of experiential empathy-creating interventions. Using a pre-and post-lab survey analysis involving 276 Computer Science 2 (CS2) students, dependent t-tests indicated that empathy-creating interventions increased student feelings that developing accessible software is important. While far from a definitive study, this observation demonstrates the foundational capability of empathy-creating interventions in experiential computing education. Existing works have demonstrated both the capabilities of experiential learning [4, 38, 48, 88] and in empathy creation [6, 23, 47, 84, 85]. However, there are no known significant efforts to examine empathy-creating interventions in experiential computing education.

3 Related Work

3.1 Experiential Education

Experiential learning is commonly used in many educational topics [4, 38, 48, 88] and has routinely demonstrated its benefits [12, 50, 51]. Experiential learning provides a complete learning experience for the student, one where they both understand the concept behind an idea and interactively learn about it [13]. Compared to alternative teaching approaches such as lectures, experiential learning has been demonstrated to be more engaging for students [55], and supports student retention of information [43, 77]. The four stages of

Kolb's Experiential Learning Cycle [52] include 'Concrete Experience,' 'Reflective Observation,' 'Abstract Conceptualization,' and 'Active Experimentation.'

3.2 Empathy-Building Interventions

Research demonstrates that people frequently fail to empathize with a particular target group because they are unwilling to empathize [73, 89]. Fortunately, research suggests that empathy can be developed, frequently through experiential activities [6, 23, 47, 84, 85]. An identified challenge in driving people to empathize are 'avoidance motives' which make empathizing more of a difficulty [31, 49, 53, 60]. An example of an avoidance motive is when people believe that addressing empathy-created concerns will be too costly [19, 66, 74] or painful [28]. Therefore, when striving to create empathy, it is imperative to demonstrate how empathy will align with, and not obstruct the project's goals [39, 74]. There are generally at least three related, but distinct subprocesses that comprise empathy [84]. 'Mentalizing' is the ability to draw inferences about a target's feelings and thoughts. 'Experience sharing' is when a person vicariously experiences another person's emotional state [42]. 'Empathic concern' focuses on a perceiver's desire to alleviate the target's distress [5]. There are several forms of empathy, including *cognitive*, *emotional*, *affective*, and *somatic* [22, 45, 61, 75]. This work will primarily focus on cognitive empathy since it is the form that is most amiable to a computing-oriented experiential environment.

There are two primary forms of empathy interventions, *Experience-based* and *Expression-based* interventions. Experience-based interventions often allow the perceiver to encounter a scenario through the target's perspective using either a hands-on or theoretical activity. This form of intervention has been traditionally used to build empathy through a deeper understanding of the target's thoughts and feelings [84]. Examples of such interventions involve medical students staying in a hospital overnight to experience a hospitalization from a patient's perspective [86], or asking participants to imagine life and feelings of a member of a stigmatized group [7]. Expression-based interventions teach participants to recognize the internal states of the participant and respond appropriately. These interventions are frequently implemented in scenarios where it is difficult to identity distress in others, or when a perceiver is impaired in conveying empathy for a target [84]. Expression-based interventions have been used in a variety of areas, such as in medical students identifying when a patient is in pain [8, 70], and helping autistic adolescents improve their affective empathy by recognizing emotional traits in others [25, 36].

4 Open Questions

There are several key questions that should be addressed in order to better understand the intersection of experiential learning for building empathy.

1. Understand the benefits and impacts of empathy-creating interventions in experiential computing education: Although there have been a large amount of existing research that demonstrates the benefits of experiential learning [4, 38, 48, 88] and empathy-building interventions [6, 23, 47, 84, 85], there is far less work that examines the intersection of these two important topics, especially in computing education. We hypothesize that empathy-creating interventions in experiential computing education will increase student interest, motivation and information retention, which are crucial for retention and encouraging students to pursue STEM careers [57, 83]. We also hypothesize that increasing empathy for diverse users will support students in understanding the need to create more equitable software. An additional question to be explored are the potential benefits of experiential vs expression-based interventions.

A better understanding of the potential benefits of empathy-creating interventions in experiential computing education can be attained using short interventions and t-tests. Measured variables may include motivation, interest and knowledge retention. A primary consideration is to ensure that a properly diverse group of students (*e.g.,* demographics, experience levels, etc.) are included in any such evaluation.

2. Recognize appropriate methodologies to include empathy-creating interventions in experiential computing education: There are several potential methodologies that may be taken to both evaluate and include experiential empathy-building interventions in computing education. We argue for small, self-contained and easily adoptable modules and interventions that can be utilized at institutions across the United States. We hypothesize that these short interventions will support the inclusion and subsequent evaluation of these topics, as short self-contained interventions have demonstrated their effectiveness in numerous other computing educational areas [65, 71, 80]. We believe in reasonably brief (*i.e.,* $\approx$ 30-60 minute interventions) since foundational computing courses are typically already packed with topics and that many institutions (especially those that are resource constrained) will not have the ability to develop entire courses focusing on this area.

3. Understand if experiential empathy creating interventions can help to reduce bias: Forms of bias include prejudice, stereotypes, affective reactions, and discrimination [32]. Bias comes in many shapes and forms ranging from overtly bias human beings, to algorithms that unintentionally contain bias [26, 76]. The adverse impacts of bias continue to be detrimental, despite

the cause.

There has been a substantial amount of work to address bias and prejudice [17, 24, 54, 69], and studies that demonstrate the potential benefits of experiential-based interventions in addressing bias [14, 63, 67, 81]. However, there are no known significant efforts to evaluate or demonstrate the impact of empathy-creating interventions in computing education in addressing prejudice.

A primary challenge will be how to effectively measure bias since it occurs both unconsciously and intentionally [11, 29]. Additionally, even if a student does recognize their own bias, they may be unlikely to truthfully admit any notions of this on a survey instrument. To address this challenge, a participant's bias could be implicitly measured, using mechanisms such as understanding affective reactions using Likert-scales to measure the range of experienced emotions [30], along with evaluating the produced artifact (*e.g.,* source code, algorithm, etc.) for aspects of bias or prejudice. These measurements could be evaluated in settings such as conventional classrooms, outreach events or small group activities using instruments such as pre-post test measures.

Despite the challenges of measuring bias, we believe that this continues to be an important and worthwhile area that warrants further exploration. This is due to the potential benefits that can be provided from the knowledge produced from further understanding how to reduce bias.

5 Conclusion

Experiential education has demonstrated its benefits in a wide variety of application areas. Additionally, empathy-building interventions have demonstrated their foundational capabilities in preliminary research. Unfortunately, there is a significant amount of research in important areas that intersect these topics that still need to be explored.

References

[1] The ada and section 508. `https://508compliantdocumentconversion.com/americans-with-disabilities-act/`.

[2] Assistive technology act law. `http://www.ataporg.org/ATLaw`.

[3] Twenty-first century communications and video accessibility act.

[4] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in neural information processing systems*, pages 5074–5082, 2016.

[5] C. D. Batson. These things called empathy: eight related but distinct phenomena. 2009.

[6] C. D. Batson, J. Chang, R. Orr, and J. Rowland. Empathy, attitudes, and action: Can feeling for a member of a stigmatized group motivate one to help the group? *Personality and Social Psychology Bulletin*, 28(12):1656–1666, 2002.

[7] C. D. Batson, M. P. Polycarpou, E. Harmon-Jones, H. J. Imhoff, E. C. Mitchener, L. L. Bednar, T. R. Klein, and L. Highberger. Empathy and attitudes: Can feeling for a member of a stigmatized group improve feelings toward the group? *Journal of personality and social psychology*, 72(1):105, 1997.

[8] J. B. Bavelas, A. Black, C. R. Lemery, and J. Mullett. " i show how you feel": Motor mimicry as a communicative act. *Journal of personality and social psychology*, 50(2):322, 1986.

[9] R. K. Bellamy, K. Dey, M. Hind, S. C. Hoffman, S. Houde, K. Kannan, P. Lohia, S. Mehta, A. Mojsilovic, and S. Nagar. Think your artificial intelligence software is fair? think again. *IEEE Software*, 36(4):76–80, 2019.

[10] C. L. Bennett and D. K. Rosner. The promise of empathy: Design, disability, and knowing the" other". In *Proceedings of the 2019 CHI conference on human factors in computing systems*, pages 1–13, 2019.

[11] I. V. Blair. Implicit stereotypes and prejudice. In *Cognitive social psychology: The Princeton symposium on the legacy and future of social cognition*, pages 359–374, 2001.

[12] W. T. Botelho, M. d. G. B. Marietto, J. C. d. M. Ferreira, and E. P. Pimentel. Kolb's experiential learning theory and belhot's learning cycle guiding the use of computer simulation in engineering education: A pedagogical proposal to shift toward an experiential pedagogy. *Computer Applications in Engineering Education*, 24(1):79–88, 2016.

[13] D. Boud, R. Keogh, and D. Walker. *Reflection: Turning experience into learning*. Routledge, 2013.

[14] C. R. Brown and G. J. Mazza. Peer training strategies for welcoming diversity. *New directions for student services*, 56:39–51, 1991.

[15] D. Brownlee. *Twitter, Square Announce Work From Home Forever Option: What Are The Risks?*, May 2020 (accessed July 3, 2020).

[16] S. Burgstahler. Designing software that is accessible to individuals with disabilities.

[17] M. M. Byrne. Instructional bias—awareness and reduction in perioperative education. *Aorn Journal*, 75(4):808–816, 2002.

[18] R. Calvo, F. Seyedarabi, and A. Savva. Beyond web content accessibility guidelines: Expert accessibility reviews. In *Proceedings of the 7th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*, DSAI 2016, pages 77–84, New York, NY, USA, 2016. ACM.

[19] C. D. Cameron and B. K. Payne. Escaping affect: how motivated emotion regulation creates insensitivity to mass suffering. *Journal of personality and social psychology*, 100(1):1, 2011.

[20] N. Carroll. Key success factors for smart and connected health software solutions. *Computer*, 49(11):22–28, 2016.

[21] G. L. Clore and K. M. Jeffery. Emotional role playing, attitude change, and attraction toward a disabled person. *Journal of personality and social psychology*, 23(1):105, 1972.

[22] D. Cohen and J. Strayer. Empathy in conduct-disordered and comparison youth. *Developmental psychology*, 32(6):988, 1996.

[23] P. Condon, G. Desbordes, W. B. Miller, and D. DeSteno. Meditation increases compassionate responses to suffering. *Psychological science*, 24(10):2125–2127, 2013.

[24] E. D. Cramer and K. D. Bennett. Implementing culturally responsive positive behavior interventions and supports in middle school classrooms: Narrating the experience of a young classroom teacher who collaborates with an experienced special education teacher to reduce subtle assumptions filled with cultural bias, this article reveals important implications for managing student behavior in more productive and culturally sensitive ways. *Middle School Journal*, 46(3):18–24, 2015.

[25] M. R. Dadds, A. J. Cauchi, S. Wimalaweera, D. J. Hawes, and J. Brennan. Outcomes, moderators, and mediators of empathic-emotion recognition training for complex conduct problems in childhood. *Psychiatry research*, 199(3):201–207, 2012.

[26] J. Dastin. Amazon scraps secret ai recruiting tool that showed bias against women. `https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G`, Oct. 2018.

[27] M. H. Davis. Measuring individual differences in empathy: Evidence for a multidimensional approach. *Journal of personality and social psychology*, 44(1):113, 1983.

[28] M. H. Davis, K. V. Mitchell, J. A. Hall, J. Lothert, T. Snapp, and M. Meyer. Empathy, expectations, and situational preferences: Personality influences on the decision to participate in volunteer helping behaviors. *Journal of personality*, 67(3):469–503, 1999.

[29] P. G. Devine. Stereotypes and prejudice: Their automatic and controlled components. *Journal of personality and social psychology*, 56(1):5, 1989.

[30] J. F. Dovidio, S. L. Gaertner, T. L. Stewart, V. M. Esses, M. ten Vergert, and G. Hodson. From intervention to outcome: Processes in the reduction of bias. *Education programs for improving intergroup relations: Theory, research, and practice*, pages 243–265, 2004.

[31] C. S. Dweck and E. L. Leggett. A social-cognitive approach to motivation and personality. *Psychological review*, 95(2):256, 1988.

[32] M. E. Engberg. Improving intergroup relations in higher education: A critical examination of the influence of educational interventions on racial bias. *Review of educational research*, 74(4):473–524, 2004.

[33] J. M. Frank, L. B. Granruth, H. Girvin, and A. VanBuskirk. Bridging the gap together: Utilizing experiential pedagogy to teach poverty and empathy. *Journal of Social Work Education*, pages 1–14, 2019.

[34] K. E. Gerdes, C. A. Lietz, and E. A. Segal. Measuring empathy in the 21st century: Development of an empathy index rooted in social cognitive neuroscience and social justice. *Social Work Research*, 35(2):83–93, 2011.

[35] K. E. Gerdes, E. A. Segal, and C. A. Lietz. Conceptualising and measuring empathy. *British Journal of Social Work*, 40(7):2326–2343, 2010.

[36] O. Golan and S. Baron-Cohen. Systemizing empathy: Teaching adults with asperger syndrome or high-functioning autism to recognize complex emotions using interactive multimedia. *Development and psychopathology*, 18(2):591–617, 2006.

[37] R. Gonçalves, J. Martins, J. Pereira, M. A.-Y. Oliveira, and J. J. P. Ferreira. Enterprise web accessibility levels amongst the forbes 250: Where art thou o virtuous leader? *Journal of Business Ethics*, 113(2):363–375, Mar 2013.

[38] S. Grace, E. Innes, N. Patton, and L. Stockhausen. Ethical experiential learning in medical, nursing and allied health education: A narrative review. *Nurse education today*, 51:23–33, 2017.

[39] A. M. Grant and D. A. Hofmann. It's not all about me: motivating hand hygiene among health care professionals by focusing on patients. *Psychological science*, 22(12):1494–1499, 2011.

[40] H. Guerrero and V. Vega. Usability analysis: Is our software inclusive? In *International Conference on Software Process Improvement*, pages 221–230. Springer, 2017.

[41] V. L. Hanson and J. T. Richards. Progress on website accessibility? *ACM Trans. Web*, 7(1):2:1–2:30, Mar. 2013.

[42] E. Hatfield, J. T. Cacioppo, and R. L. Rapson. Emotional contagion. *Current directions in psychological science*, 2(3):96–100, 1993.

[43] K. Hawtrey. Using experiential learning techniques. *The Journal of Economic Education*, 38(2):143–152, 2007.

[44] P. A. Hecker. Successful consulting engineering: a lifetime of learning. *Journal of management in engineering*, 13(6):62–65, 1997.

[45] R. Hogan. Development of an empathy scale. *Journal of consulting and clinical psychology*, 33(3):307, 1969.

[46] J. D. Holden. Improving nursing student empathy with experiential learning. 2018.

[47] H. Jazaieri, K. McGonigal, T. Jinpa, J. R. Doty, J. J. Gross, and P. R. Goldin. A randomized controlled trial of compassion cultivation training: Effects on mindfulness, affect, and emotion regulation. *Motivation and Emotion*, 38(1):23–35, 2014.

[48] S. Jose, P. G. Patrick, and C. Moseley. Experiential learning theory: the importance of outdoor classrooms in environmental education. *International Journal of Science Education, Part B*, 7(3):269–284, 2017.

[49] D. Kahneman. Prospect theory: An analysis of decisions under risk. *Econometrica*, 47:278, 1979.

[50] K. Kiili. Digital game-based learning: Towards an experiential gaming model. *The Internet and higher education*, 8(1):13–24, 2005.

[51] A. Y. Kolb and D. A. Kolb. Learning styles and learning spaces: Enhancing experiential learning in higher education. *Academy of management learning & education*, 4(2):193–212, 2005.

[52] D. A. Kolb. *Experiential learning: Experience as the source of learning and development*. FT press, 2014.

[53] Z. Kunda. The case for motivated reasoning. *Psychological bulletin*, 108(3):480, 1990.

[54] S. Lahiri and S. Self. Gender bias in education: the role of inter-household externality, dowry and other social institutions. *Review of Development Economics*, 11(4):591–606, 2007.

[55] L. Laird and Y. Yang. Engaging software estimation education using legos: A case study. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pages 511–517, New York, NY, USA, 2016. ACM.

[56] N. T. Lee. Detecting racial bias in algorithms and machine learning. *Journal of Information, Communication and Ethics in Society*, 2018.

[57] J. León, J. L. Núñez, and J. Liew. Self-determination and stem education: Effects of autonomy, motivation, and self-regulated learning on high school math achievement. *Learning and Individual Differences*, 43:156–163, 2015.

[58] T. Levett-Jones, R. Cant, and S. Lapkin. A systematic review of the effectiveness of empathy education for undergraduate nursing students. *Nurse education today*, 2019.

[59] M. Levy and I. Hadar. The importance of empathy for analyzing privacy requirements. In *2018 IEEE 5th International Workshop on Evolving Security & Privacy Requirements Engineering (ESPRE)*, pages 9–13. IEEE, 2018.

[60] K. Lewin. Group decision and social change. *Readings in social psychology*, 3(1):197–211, 1947.

[61] A. Mehrabian and N. Epstein. A measure of emotional empathy. *Journal of personality*, 1972.

[62] C. Mouza, A. Marzocchi, Y.-C. Pan, and L. Pollock. Development, implementation, and outcomes of an equitable computer science after-school program: Findings from middle-school students. *Journal of Research on Technology in Education*, 48(2):84–104, 2016.

[63] L. J. Nelson. Effects of participation in an intergroup communication program: An assessment of shippensburg university's building bridges program. 1994.

[64] I. Niculescu, H. M. Hu, C. Gee, C. Chong, S. Dubey, and P. L. Li. Towards inclusive software engineering through a/b testing: A case-study at windows. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 180–187. IEEE, 2021.

[65] S. Odeh, S. Abu Shanab, and M. Anabtawi. Augmented reality internet labs versus its traditional and virtual equivalence. *International Journal of Emerging Technologies in Learning*, 10(3), 2015.

[66] S. M. Pancer, L. M. McMullen, R. A. Kabatoff, K. G. Johnson, and C. A. Pond. Conflict and avoidance in the helping situation. *Journal of Personality and Social Psychology*, 37(8):1406, 1979.

[67] D. J. Pence and J. A. Fields. Teaching about race and ethnicity: Trying to uncover white privilege for a white audience. *Teaching Sociology*, 27(2):150–158, 1999.

[68] N. Pettijohn. *Can We Just Work From Home Forever?*, May 2020 (accessed July 3, 2020).

[69] L. Redpath. Confronting the bias against on-line learning in management education. *Academy of Management Learning & Education*, 11(1):125–140, 2012.

[70] H. Riess, J. M. Kelley, R. W. Bailey, E. J. Dunn, and M. Phillips. Empathy training for resident physicians: a randomized controlled trial of a neuroscience-informed curriculum. *Journal of general internal medicine*, 27(10):1280–1286, 2012.

[71] E. Sancristobal, M. Castro, S. Martin, M. Tawkif, A. Pesquera, R. Gil, G. Díaz, and J. Peire. Remote labs as learning services in the educational arena. In *2011 IEEE Global Engineering Education Conference (EDUCON)*, pages 1189–1194. IEEE, 2011.

[72] A. Savidis and C. Stephanidis. Inclusive development: Software engineering requirements for universally accessible interactions. *Interacting with Computers*, 18(1):71–116, 2006.

[73] K. Schumann, J. Zaki, and C. S. Dweck. Addressing the empathy deficit: Beliefs about the malleability of empathy predict effortful responses when empathy is challenging. *Journal of personality and social psychology*, 107(3):475, 2014.

[74] L. L. Shaw, C. D. Batson, and R. M. Todd. Empathy avoidance: Forestalling feeling for another in order to escape the motivational consequences. *Journal of Personality and Social Psychology*, 67(5):879, 1994.

[75] A. Smith. Cognitive empathy and emotional empathy in human behavior and evolution. *The Psychological Record*, 56(1):3–21, 2006.

[76] J. Snow. Amazon's face recognition falsely matched 28 members of congress with mugshots. `https://www.aclu.org/blog/privacy-technology/surveillance-technologies/amazons-face-recognition-falsely-matched-28`, July 2018.

[77] L. B. Specht and P. K. Sandlin. The differential effects of experiential learning activities and traditional lecture classes in accounting. *Simulation & Gaming*, 22(2):196–210, 1991.

[78] J. Strobel, J. Hess, R. Pan, and C. A. Wachter Morris. Empathy and care within engineering: Qualitative perspectives from engineering faculty and practicing engineers. *Engineering Studies*, 5(2):137–159, 2013.

[79] SwaugerShea. Software that monitors students during tests perpetuates inequality and violates their privacy. `https://www.technologyreview.com/2020/08/07/1006132/software-algorithms-proctoring-online-tests-ai-ethics/`, Aug 2020.

[80] R. F. Tinker. *Microcomputer-based labs: educational research and standards*, volume 156. Springer Science & Business Media, 2012.

[81] S. Vohra, E. Rodolfa, A. de la Cruz, and C. Vincent. A cross-cultural training format for peer counselors. *Journal of College Student Development*, 1991.

[82] M. Vorvoreanu, L. Zhang, Y.-H. Huang, C. Hilderbrand, Z. Steine-Hanson, and M. Burnett. From gender biases to gender-inclusive design: An empirical investigation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2019.

[83] X. Wang. Why students choose stem majors: Motivation, high school learning, and postsecondary context of support. *American Educational Research Journal*, 50(5):1081–1121, 2013.

[84] E. Weisz and J. Zaki. Empathy building interventions: A review of existing work and suggestions for future directions. *The Oxford handbook of compassion science*, pages 205–217, 2017.

[85] H. Y. Weng, A. S. Fox, A. J. Shackman, D. E. Stodola, J. Z. Caldwell, M. C. Olson, G. M. Rogers, and R. J. Davidson. Compassion training alters altruism and neural responses to suffering. *Psychological science*, 24(7):1171–1180, 2013.

[86] M. Wilkes, E. Milgrom, and J. R. Hoffman. Towards more empathic medical students: a medical student hospitalization experience. *Medical education*, 36(6):528–533, 2002.

[87] P. Wright and J. McCarthy. Empathy and experience in hci. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 637–646, 2008.

[88] S. Wurdinger and P. Allison. Faculty perceptions and use of experiential learning in higher education. *Journal of e-learning and Knowledge Society*, 13(1), 2017.

[89] J. Zaki and M. Cikara. Addressing empathic failures. *Current Directions in Psychological Science*, 24(6):471–476, 2015.

[90] J. Zaki and K. N. Ochsner. The neuroscience of empathy: progress, pitfalls and promise. *Nature neuroscience*, 15(5):675–680, 2012.

A Reformer Chatbot to Support Beginners in Learning Python Programming*

Poster Abstract

FNU Kaleemunnisa, Begimai Zhumakova
Pooja Patel, Krishna Mohan Bathula

Faculty Advisor
Christelle Scharff

Computer Science
Seidenberg School of CSIS
Pace University, New York, NY
cscharff@pace.edu

This research aims at designing and evaluating the use of a chatbot dedicated to supporting students in their learning of the Python programming language. By delivering on-demand guidance and tailored feedback, AI-based education chatbots have the potential to support students in their learning. They also have the possibility to be scalable and serve numerous students at once, making them accessible and productive learning tools. Python is a popular language in many disciplines, from computer science to data science and digital humanities. Learning Python is currently an excellent way to increase employability and career opportunities.

The chatbot we propose is dedicated uniquely to supporting students in learning Python. It is developed using the reformer methodology, which is based on the reversible transformer. We have experimented with GPT, BERT, and sequential models to develop this special-purpose chatbot. The sequential model, too simple, did not provide good results. The transformer model took more time and memory during training. The chosen reformer model can fit up to 1 million tokens on a 16GB GPU and handle context windows of up to 1 million words. It combines two techniques to solve the problems of attention

and memory allocation. Reformer uses locality-sensitive hashing to reduce the complexity of attending over long sequences and reversible residual layers to efficiently use memory.

We built our own dataset of more than 800 questions. The dataset consists of intents from the Python topics which include: Generalities, Arithmetic Operations, Data Types, Variables, Input and Output functions, Operators, Control Structures, File I/O Operations, Functions, and Modules. We crowdsourced intents from students and wrote the answers, including using expert knowledge from the Python.org open platform. The comprehensive documentation from Python.org is reusable for educational purposes.

Having high accuracy is not enough for a model, we still need to have it evaluated by users. Students from the undergraduate course CIS101 (Introduction to Computer) and graduate course CS661 (Python Programming) evaluated the chatbots in terms of the overall experience, usability, learning satisfaction, and engagement, with approval from Pace University Institutional Review Board (IRB). An experiment was set up for students to attempt Python problems and use the chatbot for help. We collected all questions the students asked the chatbot during the two-hour session and administered a survey to obtain students' feedback. The survey contains questions about students' background in Python, first impression of the chatbot, learning experience, user interface, and improvement needs.

The poster presents the design of the chatbot, the experiment, and the evaluation of the chatbot. It also presents the potential and limitations of such a special-purpose chatbot in the context of the popular ChatGPT general-purpose chatbot.

On Teaching Recursion in Lower Level Courses*

Poster Abstract

Jingnan Xie
Department of Computer Science
Millersville University of Pennsylvania
Millersville, PA 17551
jingnan.xie@millersville.edu

Recursion is one of the central ideas of computer science but also one of the most challenging concepts for students to understand and use. So teaching recursion is an important and challenging task. In this poster, we propose a hybrid approach for teaching recursion and compare it with some traditional methods, such as tracing recursion. The results have shown that traditional tools for teaching recursion like tracing recursion are mechanical to some degree. Almost 40% of the class could not reasonably analyze a new recursive problem. Compared with tracing recursion, a better percentage (more than 75%) of students could solve recursive problems independently using our proposed approach. Furthermore, the proposed approach provides a way to visualize recursion, which could be helpful to the students.

Prove It with Proof Buddy: The Programmer's Sidekick For Learning Proof Writing*

Poster Abstract

Steve Earth, Jeremy Johnson, and Bruce Char
Department of Computer Science
Drexel University
{se435,jjohnson,charbw}@drexel.edu

Proof Buddy is an online browser-based tool designed to teach proof writing to beginning computer science students. It has been designed from the ground up with educational purposes in mind and has been used successfully with hundreds of students since January 2022 at Drexel University. The tool helps students build and check proofs in a variety of systems. It is capable of doing Natural Deduction, both Boolean logic and first order logic, and is being extended to handle Equational Reasoning. Important instructor-centered features include that the teacher can create assignments of proof problems, which the software can auto score and have results uploaded into their school's LMS. Additionally, the tool allows proofs to be saved and exported and used as new rules. When this feature is used by the instructor, it permits a customization of the allowable rules. When this feature is used by the students, it allows them to create their own lemmas which reduces the cognitive load of a more intricate proof. Logic plays an important role in computer science, but use of formal logic has often proven to be a stumbling block for students. Formal proofs have become more important to computer science students with the resurgence of program verification and proof assistant suites. Professional grade proof assistant and theorem proving tools which handle this subject matter (such as Microsoft's LEAN, Lamport's TLA+, or Coq) are too complex for beginning programmers. Proof Buddy addresses this need for a more pedagogically minded tool. This poster showcases some of the most well-used features of Proof Buddy, as well as presents some tentative answers to research questions now tractable via the tool, including the ultimate goal of establishing a connection between mathematical proof and programming. Proof Buddy is especially

effective in discrete math and cs logic courses. Instructors teaching an introductory proof class within a computer science context would be particularly interested in Proof Buddy, especially if their course content includes Boolean Algebra or Logic. While we have been using the tool at the sophomore/junior college level, it is also appropriate for introducing the mathematical foundations of computer science to advanced high school students. Proof Buddy is a boon for teachers who are searching for a way to reach beginner programmers who are struggling with their first forays into proofs. Moreover, our tool goes beyond what others offer by providing an instructor interface which allows assignment creation and automated scoring. The axiom system is also extensible by instructor and student created lemmas. Unlike other proof assistants, Proof Buddy allows the teacher to create missing parts of proofs, with the choice of either the code/mathematical expressions or the formal rule justifying that line to be filled in by the student. Additionally, the proof creation mode is flexible with the order of construction: the tool supports students working from the top-down (i.e. the formal order of going from assumptionstotheconclusion)or fromthebottom-up(working"backwards"fromtheconclusion).Ithas been designed with a framework that allows extension to different theories and permits data collection of student work which can be analyzed for common mistakes. The beta version of Proof Buddy has already been successfully used by hundreds of students at Drexel University over the past year and received very positive reviews in surveys. The poster will include some of the preliminary results we have obtained from analyzing data collected via the tool, as well as a QR code for the viewer to try Proof Buddy out for themselves and adopt it for their own classroom!

Code That Communicates*

Poster Abstract

Devon Cook[1], Hal Smith[2]
[1]English
[2]Information Sciences and Technology
Penn State University
New Kensington Campus
New Kensington, PA 15068
{dpc5300,hhs10}@psu.edu

The purpose of this study is to explore the efficacy of using technical communication principles to teach beginner programmers how to improve the readability of their source code. Technical communication principles are typically used to help students in technical fields improve their writing, but our hypothesis is that these same principles could be applied to the process of writing readable programs. This study is designed as a case study that will explore pedagogical approaches and outcomes associated with our hypothesis. The study utilizes an existing technical writing course that is required by some majors and is one of a set of options for others. Most of the traditional technical writing activities are replaced with ones that focus on improving code readability from three perspectives: the algorithm, the module, and the system. Each perspective explores technical communication principles as they relate to coding standards, approaches to commenting, and visualization techniques. To enroll in the course, students ideally would have completed at least one three-credit programming course. As the study is being done at a small campus, to ensure there would be sufficient enrollment for the course to execute, we also permitted students who were concurrently taking a three-credit programming course to participate. As a result, this course focuses on improving existing code rather than being concerned with creating it. Further, given the limited experience of the students (n=6, five of which are concurrently enrolled in a CS1 course), we focus primarily on the readability of the code and limit the discussion of code efficiency.

A Web App for Writing With Mathematical Logic*

Poster Abstract

Bruce Char, Steve Earth, and Jeremy Johnson
Drexel University
Philadelphia, PA 19104
{charbw,se435,johnsojr}@drexel.edu

Logicwriter Actual is a web app (`https://www.cs.drexel.edu/~bchar/logicwriter/standardConfig/web/index.html`) designed for freeform entry and linear display in Unicode of text combined with symbolic logic characters such as $\equiv$, $\exists$, $\wedge$, $\Longrightarrow$, λ, and Ω. It is designed for the writing done by students or instructors in foundational-level (second year) courses introducing mathematical reasoning: elementary formal or informal proofs often involving commonplace situations or computer science contexts such as program behavior. Rather than being a scaffolded practice harness [1], or an automated reasoning tool/proof checker [2, 5], the goal of Logicwriter Actual is just to make it easier for students to practice more mathematical writing. The WYSIWYG result can be copy/pasted into most document processors (for submitted or shared work), Discord or Slack channels (for chat conversations), email, code editors, etc. It is designed to be immediately usable by browser- and laptop-savvy students, so more convenient to use in a foundational course than available alternatives (word processors, LaTeX, LyX, keyboard entry of Unicode indices, Math Jax plugins, etc. [3, 4, 6]) It is designed to need minimal computer resources (runs in browser, can be delivered from any web page server), and instructor time (for student training, or tech support). Because it is just a writing tool, it is compatible with most instructional approaches that ask students to write their own proofs and explanations. Assessment is underway through student survey of usage experience and effects, and by instructor survey/interview.to see if there are perceived benefits to its approach to text entry and style of implementation as a web app.

References

[1] Maria L Blanton, Despina A Stylianou, and M Manuela David. "Understanding Instructional Scaffolding in Classroom Discourse on Proof 1". In: *Teaching and learning proof across the grades*. Routledge, 2010, pp. 290–306.

[2] Jianting Chen et al. "ORC2A: A proof assistant for undergraduate education". In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 2017, pp. 757–758.

[3] Stephen Cummins et al. "Equality: A tool for free-form equation editing". In: *2015 IEEE 15th International Conference on Advanced Learning Technologies*. IEEE. 2015, pp. 270–274.

[4] Peter Jipsen. *ASCII Math*. `http://asciimath.org`.

[5] Wolfram Kahl. "CalcCheck: a proof checker for teaching the "logical approach to discrete math"". In: *Interactive Theorem Proving: 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings 9*. Springer. 2018, pp. 324–341.

[6] Murray Sargent III. "Unicode nearly plain-text encoding of mathematics version 3". In: *Unicode Technical Note* 28 (2010), pp. 1–49.

A Study of the Perception of Mathematics as a Learning Tool for Computer Science Undergraduates*

Poster Abstract

Bruce Char[1], John P. Dougherty[2]
[1]Drexel University, Philidelphia, PA 19104
charbw@drexel.edu
[2]Haverford College, Haverford, PA 19041
jd@cs.haverford.edu

Most programs in computer science undergraduate programs have evolved from a mathematics program, an engineering program, or some other combination. In computer science education, we will often exploit problems from mathematics with computational solutions to introduce and develop a deep understanding of computational concepts and skills related to program development and analysis. However, it needs to be clarified the degree to which students in undergraduate computer science programs see this connection between what they are learning and the role that mathematics can play in increasing the understanding of that learning. Do they view the mathematics they encounter as one of the unavoidable burdens that the major makes them endure to get a degree but will not need much after graduation? Or do they see mathematical learning and inventive thinking as something that they will need to do on a continuing basis even after graduation? The authors developed an instrument that could be used to gather evidence to understand better how our students look at the role of mathematics in their computer science courses and projects. Beginning in the fall of 2021 a survey instrument was developed and administered to students primarily at Haverford College. Students who completed the initial prototype survey were enrolled in an accelerated CS1-CS2 course where they had experience with programming before attending college. It is also assumed that they have been exposed to mathematics throughout their education. It should also be clear that in this context mathematics is

defined expansively to include algebraic and other numerical manipulations of data, as well as problem-solving techniques to divide and conquer complicated problems, appreciating and understanding the use of abstraction [2], reasoning about the state of various parts of computation through the out the execution process, as well as this distinction between verification by proof versus testing. At Drexel University, students in computer science are required to complete courses in calculus. Meanwhile, at Haverford College students are required to complete a course in discrete mathematics as well as one of two choices between analysis of algorithms and theory of computation. At each school, other courses either directly or tangentially related to mathematics are taken by students to increase their understanding and prepare them for courses that apply to computing. The obvious fields are computer science and mathematics, but also include machine learning, data science, scientific computing, and computational linguistics. Perhaps we should have expected, responses to the prototype to have indicated that students have a variety of impressions about the role of mathematics in their studies for computer science. Some of them may impact the development of persistent robust self-regulated mathematical thinking and learning even after entry into the workforce. We suspect that students in computer science do not perceive how mathematics is used in the field and thus either delay or avoid mathematics or leave the field of computer science. Perhaps students find mathematics irrelevant, unengaging, we're simply too difficult, our goal with this project is to identify first if this disconnect between math and computer science exists we're not. Our intention is to identify obstacles and suggests approaches to help computer science educators provide this mathematical foundation for computer science students [3]. We plan to continue this study to get a clearer picture of how students look at the role of mathematics. We believe that a sound foundation in math contributes positively to those in computer science [1, 4].

References

[1] John Konvalina, Stanley A Wileman, and Larry J Stephens. "Math proficiency: A key to success for computer science students". In: *Communications of the ACM* 26.5 (1983), pp. 377–382.

[2] Claudio Mirolo et al. "Abstraction in Computer Science Education: An Overview". In: *Informatics in Education* 20.4 (2022), pp. 615–639.

[3] Tom Prickett et al. "Resilience and effective learning in first-year undergraduate computer science". In: *Proceedings of the 2020 acm conference on innovation and technology in computer science education.* 2020, pp. 19–25.

[4] Keith Quille and Susan Bergin. "Programming: predicting student success early in CS1. a re-validation and replication study". In: *Proceedings of the 23rd annual ACM conference on innovation and technology in computer science education.* 2018, pp. 15–20.

Reviewers — 2023 CCSC Northeastern Conference

Chris Alvin ..... Furman University
Kailash Chandra ..... Pittsburg State University
Lawrence D'Antonio ..... Ramapo College
Joan DeBello ..... St. John's University
Dan DiTursi ..... Siena College
Peter Drexel ..... Plymouth State University
Alfreda Dudley ..... Towson University
Matthew Ferland ..... University of Southern California
Alice Fischer ..... University of New Haven
Timothy Fossum ..... Rochester Institute of Technology
Martin Gagne ..... Wheaton College
Michael Gousie ..... Wheaton College
Lama Hamandi ..... Northeastern University
Nadeem Hamid ..... Berry College
Delbert Hart ..... SUNY Plattsburgh
Zach Kissel ..... Merrimack College
Daniel Krutz ..... Rochester Institute of Technology
John MacCormick ..... Dickinson College
Sriharsha Mallapuram ..... Plymouth State University
Stephanos Matsumoto ..... Olin College of Engineering
Kevin McCullen ..... SUNY Plattsburgh
Sunjae Park ..... Wentworth Institute of Technology
Sofya Poger ..... Felician University
Raghav Sampangi ..... Dalhousie University
Unnati Shah ..... SVNIT
Hal Smith ..... Penn State New Kensington
Ashley Suchy ..... SUNY New Paltz
William Tarimo ..... Connecticut College
James Teresco ..... Siena College
Scott Valcourt ..... Northeastern University
Marc Waldman ..... Manhattan College
Michael Walters ..... SUNY Plattsburgh
Yang Wang ..... La Salle University