The Journal of Computing Sciences in Colleges

Papers of the 13th Annual CCSC Southwestern Conference

March 20-21, 2020
California State University San Marcos
San Marcos, CA

Baochuan Lu, Editor
Southwest Baptist University

Mariam Salloum, Regional Editor
UC Riverside

Volume 35, Number 10 **April 2020**

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:

Jeff Lehman, President (2020), (260)359-4209, jlehman@huntington.edu, Mathematics and Computer Science Department, Huntington University, 2303 College Avenue, Huntington, IN 46750.

Karina Assiter, Vice President (2020), (802)387-7112, karinaassiter@landmark.edu.

Baochuan Lu, Publications Chair (2021), (417)328-1676, blu@sbuniv.edu, Southwest Baptist University - Department of Computer and Information Sciences, 1600 University Ave., Bolivar, MO 65613.

Brian Hare, Treasurer (2020), (816)235-2362, hareb@umkc.edu, University of Missouri-Kansas City, School of Computing & Engineering, 450E Flarsheim Hall, 5110 Rockhill Rd., Kansas City MO 64110.

Judy Mullins, Central Plains Representative (2020), Associate Treasurer, (816)390-4386, mullinsj@umkc.edu, School of Computing and Engineering, 5110 Rockhill Road, 546 Flarsheim Hall, University of Missouri - Kansas City, Kansas City, MO 64110.

John Wright, Eastern Representative (2020), (814)641-3592, wrightj@juniata.edu, Juniata College, 1700 Moore Street, Brumbaugh Academic Center, Huntingdon, PA 16652.

David R. Naugler, Midsouth Representative(2022), (317) 456-2125, dnaugler@semo.edu, 5293 Green Hills Drive, Brownsburg IN 46112.

Lawrence D'Antonio, Northeastern Representative (2022), (201)684-7714, ldant@ramapo.edu, Computer Science Department, Ramapo College of New Jersey, Mahwah, NJ 07430.

Cathy Bareiss, Midwest Representative (2020), cbareiss@olivet.edu, Olivet Nazarene University, Bourbonnais, IL 60914.

Brent Wilson, Northwestern Representative (2021), (503)554-2722, bwilson@georgefox.edu, George Fox University, 414 N. Meridian St, Newberg, OR 97132.

Mohamed Lotfy, Rocky Mountain Representative (2022), Information Technology Department, College of Computer & Information Sciences, Regis University, Denver, CO 80221.

Tina Johnson, South Central Representative (2021), (940)397-6201, tina.johnson@mwsu.edu, Dept. of Computer Science, Midwestern State University, 3410 Taft Boulevard, Wichita Falls, TX 76308-2099.

Kevin Treu, Southeastern Representative (2021), (864)294-3220, kevin.treu@furman.edu, Furman University, Dept of Computer Science, Greenville, SC 29613.
Bryan Dixon, Southwestern Representative (2020), (530)898-4864, bcdixon@csuchico.edu, Computer Science Department, California State University, Chico, Chico, CA 95929-0410.

Serving the CCSC: These members are serving in positions as indicated:
Brian Snider, Membership Secretary, (503)554-2778, bsnider@georgefox.edu, George Fox University, 414 N. Meridian St, Newberg, OR 97132.
Will Mitchell, Associate Treasurer, (317)392-3038, willmitchell@acm.org, 1455 S. Greenview Ct, Shelbyville, IN 46176-9248.
John Meinke, Associate Editor, meinkej@acm.org, UMUC Europe Ret, German Post: Werderstr 8, D-68723 Oftersheim, Germany, ph 011-49-6202-5777916.
Shereen Khoja, Comptroller, (503)352-2008, shereen@pacificu.edu, MSC 2615, Pacific University, Forest Grove, OR 97116.
Elizabeth Adams, National Partners Chair, adamses@jmu.edu, James Madison University, 11520 Lockhart Place, Silver Spring, MD 20902.
Megan Thomas, Membership System Administrator, (209)667-3584, mthomas@cs.csustan.edu, Dept. of Computer Science, CSU Stanislaus, One University Circle, Turlock, CA 95382.
Deborah Hwang, Webmaster, (812)488-2193, hwang@evansville.edu, Electrical Engr. & Computer Science, University of Evansville, 1800 Lincoln Ave., Evansville, IN 47722.

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 Partner

Turingscraft
Google for Education
GitHub
NSF – National Science Foundation

Silver Partners

zyBooks

Bronze Partners

National Center for Women and Information Technology
Teradata
Mercury Learning and Information
Mercy College

2020 CCSC Southwestern Conference Committee

Youwen Ouyang, Conference Chair .. California State University San Marcos
Megan Thomas, Papers Chair ..... California State University, Stanislaus
Mariam Salloum, Authors Chair ..... University of California, Riverside
Adam Blank, Posters Chair ..... California Institute of Technology
Michael Shindler, Panels/Tutorials Chair .University of California, San Diego
Paul Cao, Lightning Talk Chair ..... University of California, San Diego
Michael Shindler, Partner's Chair ..... University of California, San Diego

Regional Board — 2020 CCSC Southwestern Region

Michael Doherty, Region Chair ..... University of the Pacific
Niema Moshiri, Treasurer/Registrar ..... University of California, San Diego
Bryan Dixon, Regional Representative .... California State University, Chico
Angelo Kyrilov, Webmaster ..... University of California, Merced
Colleen Lewis, Past Region Chair ..... Harvey Mudd College
Cynthia Lee, Past Conference Chair ..... Stanford University

Experience Report: Preemptive Final Exams for Computer Science Theory Classes*

Michael Shindler
University of California Irvine
mikes@uci.edu
Matt Ferland, Aaron Cote, Olivera Grujic
University of Southern California
{mferland, aaroncot, grujic}@usc.edu

Abstract

We taught classes enacting a "preemptive final exam" grading mechanism. Students have multiple chances to display knowledge of topics, each being worth a portion of the final grade. The grade earned in each topic is some number of their best attempts out of a higher number of chances.

1 Introduction and Previous Work

As class sizes grow, it is becoming imperative to find more scalable evaluation methods. Recent concerns have included rubrics for large numbers of assignments [3], automated ways to detect students who are at risk of failing [1, 5, 6], support structures [7], and course management for instructors [4].

We introduce a grading mechanism we refer to as a "preemptive final exam," where students can display knowledge before the final exam date, allowing students to skip certain topics on the final, and saving course staff from grading those questions. This policy has precedence, such as some math teachers allowing students with an 'A' on the quizzes to skip the final exam, or allowing a comprehensive final exam to count for a larger percent of the grade if it

is a better score [8]. The latter method has improved the perception of fairness without having a large impact on class averages [8]. A related popular grading mechanism was in use at MIT's 6.034 (Artificial Intelligence) course, allowing students to use quiz scores in place of corresponding sections of their final exam. This method [10] is the basis for our preemptive final exam model. Mastery based classes offer something similar, but typically allow the students more than two chances at each topic [2]. Because of this, they tend not to scale as well to large classes that require proofs and algorithm creation rather than numerical or multiple-choice answers.

Our system is as follows. Every core topic is assigned a portion of the total class weight and can be fulfilled during one or more pre-final exams during the term. The final exam then offers another chance to demonstrate this mastery. Students who demonstrate their understanding early will have "preempted" the topic from their final while students who did not can raise their grade, but only by demonstrated improved understanding of the topic. We believe this achieves a focus on feedback wherein students receive timely remarks about what they did or did not understand about the material, with both a direction to go with their study and a chance to improve that part of their grade.

Students in all of our implementations typically increased their final grade after taking the final exam, often by a significant amount. In both implementations, over two thirds of the students improved their grade by at least 5%, and over a tenth of the students improved their grade by at least 15%.

In this report, we discuss courses that enacted this grading mechanism. We cover a year's worth of teaching at the University of Southern California, using one semester of an algorithms course and one semester of a Discrete Math course.

2 Course Setup

2.1 Introduction to Algorithms

After reviewing fundamental prerequisite concepts, this course typically covers some algorithm design paradigms: dynamic programming, greedy, and divide-and-conquer, followed by a midterm covering those three topics. The midterm has students solve one problem for each paradigm by designing an algorithm using it. We then cover network flow and some computational complexity, and conclude with a final exam.

For our offering, however, the course structure changed slightly. The lecture structure remained intact, as did the placement and purpose of the fundamentals quiz and midterm. We added a quiz in the last week of class, giving a chance to try the post-midterm topics in an exam environment and for credit towards "best of two." There was a final exam with five questions, one on each of the five topics.

40% of the grade was from homework, a fundamentals quiz, and overall midterm performance. The remaining 60% of the grade was separated into 5 sections, with one for each topic. Every topic had exactly two questions used for grading: one question on the final, and one question on the midterm or week 15 exam. The total grade for each topic would be the highest score of the two attempts. Exams contained only one question from each of the 5 topics, and each section was clearly labeled. For more details, please see the online appendix, available at `https://www.ics.uci.edu/~mikes/papers/CCSC_2020_Online_Appendix.pdf`

3 Discrete Mathematics

This class typically has two mid-semester exams, some take-home problem sets, and a final exam. We renamed the two mid-semester exams to be quizzes, and added a third one. Like the algorithms course, this quiz came in the last week of classes and covered material that would be on the final but had been lectured on after the cut-off point of the previous quiz. Each quiz was worth little on its own: the first and third were 5% each, and the second was 10%. The remaining grade that came neither from a quiz sum nor a take-home assignment came from breaking down the core topics to be learned in the course. There was a final exam that provided one last opportunity to demonstrate mastery of each topic.

Unlike algorithms, the topics weren't "best of two." Rather, we gave a number of opportunities for each topic. For example, each quiz had a (non-inductive) proof for the student to write, and the final exam had two. The best three of the five counted for 15% of the students' grade. This is partly because we don't believe that a single question in all of these topics covers the full mastery of the material. By contrast, once one has demonstrated the ability to design an algorithm using, say, dynamic programming, we believe students will retain that mastery, at least at the undergraduate level, and for at least the amount of time between the midterm and final exam.

The topics and breakdowns are presented in the online appendix.

4 Observed Outcomes

4.1 Positive Outcomes

In the algorithms course, 28 students out of 203 had an A without taking the final. Subjectively, the instructor for the course believes each of these students would have also earned an 'A' had the class grading been more traditional. We believe that the reported grade accurately reflects these students' knowledge and experience with the material. While it is conceivable that they, or any student preempting a final exam question, could have done poorly under a traditional model, we believe that the learning curve is such that the

students would be able to earn at least the same score with very high probability. Furthermore, it meant that these students would not be occupying office hours during finals period, freeing course staff for students who still have yet to demonstrate the mastery that they were (hopefully) working on.

Of course, the benefits to students go beyond the fact that some may skip the final exam, as the structure of the class creates a focus on feedback and improvement. Students can see what understanding they lack in an exam and have a chance to demonstrate it again, having learned from their mistakes. This focus on improvement is the point: showing that they learned something they had not previously demonstrated mastery of, and this is reflected explicitly in their grade. Students cannot hide a lack of knowledge on one topic by repeatedly demonstrating mastery of another.

We found the topics most influenced by this policy to be those given earlier in the term, as well as those where the class as a whole performed poorly in initial attempts. This was observed in both classes.

4.2 Particulars of Improvement at the Final

For the algorithms course, 157 students took the final. Not every student needed to attempt every question. For example, the vast majority of the class was happy with their score on the divide and conquer paradigm on the midterm. The breakdown by topic is reflected in the online appendix. The final itself could have been offered in a traditional algorithms course at our university. As such, students who had fewer questions to answer generally would leave earlier and did not appear to have a time advantage over students who had more to answer. This was true even of students with extra time accommodation, where over 90% finished early in both courses. We view the improvement, particularly in dynamic programming, as very encouraging. This is a key concept in algorithm design, and seeing that students are able to incorporate their feedback and learn from it is reassuring.

Just like with dynamic programming, we were encouraged by the improvements we saw on abstract topics from early in the term. Proofs, in particular, feel to many students to be some mysterious art form, yet with sufficient resources and incentive, they are able to develop skills in this area. For the first three topics, this should provide a cause for optimism for their future as computer scientists. Proofs are structured not unlike a computer program [9] and the connection between inductive proofs and recursive thinking should be obvious. There is a similar positive view for students to grasp some fundamental graph algorithms in what is for many of them their first year of university.

In both classes, most students who took the final saw some improvement in their grade. Over two thirds of students that took the final received at least a 5% improvement in their overall grade, and 10% received at least a 15% improvement. A breakdown is available in the online appendix.

4.3 Cautions for the Approach

Each semester we made the usual promise that 90% or above of the points would be sufficient for an 'A' in the class. For the algorithms class, nearly a third of the students finished in excess of 90%. The instructor recognized most of the names and, similar to the opinion of those who preempted the full final, believes that if we were assigning letter grades based on impression of their understanding, each would likely have had an 'A' as well. We believe this is the correct evaluation of each such student. Other instructors may have different views about what should constitute an A and should exercise caution in such promises. At the other extreme, we did not notice any students "optimizing for merely passing," although it could happen. Our view is that a student so optimizing is unlikely to have it secured by the final exam, and cannot rely on the curve for this model in the same way as for a traditionally evaluated course.

We believe our approach works under the assumption that a small number of samples is sufficient to demonstrate mastery. We believe this holds for undergraduate CS Theory, but it may not hold for other subject matters.

Instructors wishing to use this grading mechanism should be cautioned that we observed more regrade requests during the year. It appears some students now perceived the stakes to be higher on exams. Instructors should also be cautioned that there will be a grading crunch at the end, as all pre-finals need to be returned before exam week.

The lower rate of improvement for topics introduced late in the class in both offerings is a cause for concern. It is not clear if students need more time to integrate the difference or if they viewed late quizzes as a nearly free shot at taking part of their final exam twice. It also isn't clear which subset of the students viewed their feedback for such artifacts and if other means of returning the exams to students might be more effective.

Lastly, we caution instructors considering this method to carefully consider the discrete nature of the topics they are teaching. For the algorithm design course, this is less of an issue; in a discrete math class, however, we may want to cross categories more often, such as asking how many Hamiltonian Paths a graph has. Counting and discrete probability also have significant overlap.

5 Summary and Conclusions

We have introduced and piloted the use of a preemptive final exam model for two different Computer Science Theory courses. We believe we have shown that this has allowed students to better and more clearly demonstrate their competency in core topics of the course. The feedback loop created aids in this goal, as does the incentive structure. The mechanism also allowed better use of end-of-term class resources while providing a new incentive for students

to demonstrate mastery early in the term. This technique should be widely applicable across the field of Computer Science.

References

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*, pages 121–130, 2015.

[2] J B. Collins, Amanda Harsy, Jarod Hart, Katie Anne Haymaker, Alyssa Marie Hoofnagle, Mike Kuyper Janssen, Jessica Stewart Kelly, Austin Tyler Mohr, and Jessica OShaughnessy. Mastery-based testing in undergraduate mathematics courses. *PRIMUS*, 29(5):441–460, 2019.

[3] John Cigas, Adrienne Decker, Crystal Furman, and Timothy Gallagher. How am i going to grade all these assignments? thinking about rubrics in the large. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 543–544, 2018.

[4] David G Kay. Large introductory computer science classes: strategies for effective course management. *ACM SIGCSE Bulletin*, 30(1):131–134, 1998.

[5] Soohyun Nam Liao, Daniel Zingaro, Christine Alvarado, William G Griswold, and Leo Porter. Exploring the value of different data sources for predicting student performance in multiple cs courses. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 112–118, 2019.

[6] Soohyun Nam Liao, Daniel Zingaro, Kevin Thai, Christine Alvarado, William G Griswold, and Leo Porter. A robust machine learning technique to predict low-performing students. *ACM Transactions on Computing Education (TOCE)*, 19(3):1–19, 2019.

[7] Mia Minnes, Christine Alvarado, and Leo Porter. Lightweight techniques to support students in large classes. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 122–127, 2018.

[8] Ben Stephenson. The impacts of providing novice computer science students with a second chance on their midterm exams. *Journal of Computing Sciences in Colleges*, 27(4):122–130, 2012.

[9] Daniel J Velleman. *How to prove it: A structured approach*. Cambridge University Press, 2019.

[10] Patrick Henry Winston. Skills, big ideas, and getting grades out of the way. `http://web.mit.edu/fnl/volume/204/winston.html`.

Simplifying Teaching Continuous Integration and Continuous Deployment with Hands-on Application in a Web Development Course

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

Abstract

Teaching web programming can lend itself as a course where students can learn to develop with version control, containers, continuous integration (CI), and continuous deployment or delivery (CD). In my web programming course, I built a starter repo that simplifies the initial setup and helps students become familiar with CI/CD and containers. Using the GitLab/GitHub APIs allows me to automate the creation of private mirrored repos across these sites for each student, with each private repo including the starter repo I developed. Students gain access to their repo by submitting a form through my website. This paper will discuss the motivation and benefit of this approach, as well as describe how I have been able to successfully implement it. The conclusions of the impact of this approach in part come from student feedback since I started doing this approach over the past 3 semesters.

1 Introduction

Over this past year, there have been numerous articles regarding the demand for people with skills related to Development Operations (DevOps) engineering [11][9]. DevOps practices are becoming much more widely utilized by developers throughout the tech industry [10]. This is not new and has been an increasingly important component of modern software development. Companies have a difficult time finding and hiring employees with the skills and training required by this work, and on-the-job training is sometimes the only option. It is important for students to be exposed to these technologies and tools via our

computer science curriculum, such that they are not caught off guard if they are asked to use a DevOps pipeline when they enter the workplace after graduation. Giving students an example of developing with DevOps is why I built a simple CI/CD pipeline and continue to provide this to my web students each semester, along with teaching them Docker and Kubernetes [2][6]. Kubernetes is a tool to manage container deployments at scale.

2 Web Course

As I teach the web programming course that is required by both of the majors in my department, it has proven to be a great opportunity for giving students further practice with version control using git, in addition to being a simple way to introduce containers and working with a DevOps pipeline. The idea to potentially start including CI/CD came from our regular Industry Advisory Board (IAB) meetings. One goal of any college or university department should be to graduate students who can get jobs, and industry leaders want to see students coming out of school with exposure, experience, and skills in these areas.

2.1 Git Organization Management

GitHub and GitLab provide ways to have a group of repositories and control access to the repositories in that group [3][4]. These are called organizations on GitHub and Groups on GitLab. GitHub and GitLab also both provide a REST API to programmatically interact with your groups and repositories. I leveraged these APIs to build a form on my website for using GitHub in my courses, and I created a private repo for each student, where they were the only one who could see their repository. As I am the owner of the organization, I can view and access all of the repos. In many of my courses, I have used these repos as an alternative submission option for turning in assignments and projects. It is practical to provide students with a repo that can be used specifically for the course and that I already have access to, in case students request help with their code, an incorrect file was accidentally submitted, or assistance is needed for any other reasons.

2.2 Starter Code

When I started teaching, I already had previous experience working with GitHub APIs and had built a form for students to create course-specific repositories through my personal website. I gave my students the initial starter code that I had developed, so that their repositories would all begin with the same foundation. Part of my primary motivation for moving in this direction was

that I was already introducing students to Docker in my courses, but it was usually near the end of the semester and students found it difficult to get started. I realized that teaching students how to use Docker should be considered an important goal of any modern web programming course, yet figuring out how to build a Docker container or get it up and running had never actually been a component of my stated learning outcomes. Thus, I began teaching the course with these goals in mind from day one. When students came to class the first day, they would already be set up with the starter repo that I had created and copied into each student's repository, so that everyone could begin from the same point. Deciding to also integrate CI/CD into the course required some additional considerations. I needed to figure out what CI/CD tool I wanted to use, and many of these platforms rely on code in the repo to tell the tool how to perform the CI/CD pipeline steps. There are many possible options available, and they all have different pros and cons. Instead of discussing all of the options and how they vary, I will focus on the tool that I included in my starter repo. The starter repo is a public GitHub repo that anyone is welcome to use for teaching a similar course or as a starting point for any other courses that may benefit from this setup [8].

2.2.1 GitLab

For my course, I decided to use GitLab, as it provides the same remote repository functionality that GitHub offers for students, but it also has CI/CD built into its platform. GitHub has recently started adding in similar features; however, GitLab is the more well-established option in this regard. Similar to GitHub, GitLab also offers any educational institution access to their top tier accounts for free [5]. The key benefit of the top tier accounts is the amount of cloud runner minutes available to your organization on GitLab. Another tool I have taken advantage of is GitLab Runner, a GitLab tool that will connect to your repositories or groups, for use by the repositories it contains, to actually run the steps in your GitLab pipeline. The cloud version of this tool can be used without any setup, and while free accounts only receive a small amount of time, education accounts receive significantly more. Alternatively, you can set up your own servers to run a GitLab Runner for no extra cost.

My website has a form for students that requires a GitHub and GitLab username. I automatically create a GitHub repository in my course organization that mirrors the starter repository code for each student. I then link/mirror this newly created GitHub repository to a GitLab repository in the corresponding group on that platform. In this way, students have the ability to access the same code on both platforms.

2.2.2 Pipeline

In the starter repo, I built and provided my students a simple GitLab pipeline, written in YAML, to offer them some experience working with a CI/CD pipeline in web development. The simple pipeline has 4 stages on the master branch and 3 stages on any other branch of the repository. I decided to treat the master branch as the production or published code branch for the repositories. The first 3 stages are the same regardless. The pipeline is also built for the framework I use when teaching the course, which is the Python-based Django web framework [1].

1. Stage One: build and publish a Docker container with the code in the repository tagged *testing*.
2. Stage Two: Run the code against the Django specific pylint linter using the published testing Docker container[7].
3. Stage Three: Run the unit tests and generate the coverage using the published testing Docker container.
4. Stage Four (only on master): build and publish a docker container with the code in the repository tagged *latest*.

Due to the nature of CI/CD pipelines, if there is a failure at any stage, the tool will not progress to the subsequent stages. This is extremely useful behavior. If a student is pulling the latest container to a Kubernetes deployment, the code will automatically update when new code is pushed into the master branch, after that code has passed the simple tests that have been set up in the pipeline. Starting with simple tests helps demonstrate how automated CI/CD can work. As more rigorous testing is utilized and extra steps are added to the pipelines that can look for potential issues with the code automatically, the code can continuously and automatically keep rolling out to users once it has passed these checks. Testing and other software engineering fundamentals are taught in our software engineering course, but it is beneficial to reinforce understanding of these ideas and provide students with opportunities to apply best practices in their work.

2.3 Use in Class

During the first week of class, after giving a brief overview of basic HTML, CSS, and JavaScript, I start introducing students to the Python language. For many students, this is their first experience with Python, as C++ is the primary language taught in our curriculum. Given that web programming is an upper division course, I mostly focus on some simple syntax differences

between Python and C++ that they should know. However, before I get them started with programming in Python, I discuss virtual environments for Python and why these are beneficial. I start with actual Python virtual environments that symlink the Python code and environment to a local folder, but I take it even further by introducing them to Docker and how they can virtualize a Python environment in Docker. By using the provided *docker-compose.yml* in the starter repo, I show them how they can run the container and get a bash shell with that environment. The *docker-compose* is configured to map ports and run the Django development server for them, so while there is a lot of setup involved upfront, this eventually makes the workload on the students much easier. This process also mounts all of the repo folder code into the Docker container when it is running in *docker-compose*. This allows the person developing the code to see real time updates in the development server, as files are edited on the host machine while it is running. Alternatively, the command line *bash* shell can be used to run Python code from that folder directly.

3 Student Outcomes

I have now taught my web course for a few semesters with this new approach, demonstrating and providing the GitLab CI/CD features and repos to my web students. I have independently gathered feedback from students every semester, in addition to receiving my official campus evaluations on the course, where students have been given the opportunity to comment on various aspects of the course. Comparing all of this feedback with responses from before I tried this strategy, I have found that students generally feel they are better prepared to do web development now than students earlier web courses I taught.

From this feedback, the most significant change I will potentially make for future courses is to create another getting started assignment that requires students to actually use the CI/CD pipeline. Students will need to write Django tests for their code, and when they submit their code, they will have to make sure it passes their tests, meets a certain code coverage percentage, and passes the full CI/CD pipeline I provided them. Additionally, while I have encouraged students to learn Docker from the outset for many semesters, I may soon begin requiring them to submit their assignments so that they run from Docker, instead of allowing containerization to be optional.

4 Conclusion

Through my experience with students over a number of semesters, adding containers and CI/CD to my web programming course has not only increased my inclusion and use of git in lecture, but also in the assignments and projects

I require my students to do with version control in the course. As I have been teaching the web programming course in this fashion over the past two semesters and this current semester, numerous students have expressed to me that they feel notably better prepared for potential jobs in web programming now, which was my primary goal. While their comments have not all been written down in an evaluation form, many of them have verbally commented to me that they appreciate being exposed to potential careers in DevOps engineering and have now seen how tools and packages such as containers could be useful for other forms of development as well. Feedback from our IAB members has been this type of experience is what they are looking for from students coming out of school. Including this kind of material in our curriculum is adding the type of functional content that will benefit our students.

References

[1] Django. `https://www.djangoproject.com`.

[2] Docker. `https://www.docker.com/`.

[3] Github. `https://www.github.com/`.

[4] Gitlab. `https://gitlab.com/`.

[5] Gitlab for education. `https://about.gitlab.com/solutions/education/`.

[6] Kubernetes. `https://kubernetes.io/`.

[7] Pylint. `https://pypi.org/project/pylint/`.

[8] Starter repo. `https://github.com/CSUChico-CINS465/starter_repo`.

[9] Devops engineer is the most recruited job on linkedin. `https://www.businessinsider.com/devops-engineer-most-recruited-job-on-linkedin-2018-11`, November 2018.

[10] Measuring ci/cd adoption rates is a problem. `https://thenewstack.io/measuring-ci-cd-adoption-rates-is-a-problem/`, August 2019.

[11] Python developer, data scientist or devops: Which tech jobs pay best? `https://www.zdnet.com/article/python-developer-data-scientist-or-devops-which-tech-jobs-pay-best/`, June 2019.

Incorporating Arduino Labs into a CS0 Course*

Ingrid Russell, Carolyn Pe Rosiene, and Aaron Gold
Department of Computing Sciences
University of Hartford
W. Hartford, CT 06117
{irussell, rosiene, aagold}@hartford.edu

Abstract

As part of our University's efforts to integrate high impact practices into the undergraduate curriculum, the Computing Sciences Department developed a model for revitalizing an introductory computer science course for non-majors. The overarching goal of our efforts is to enhance the learning experiences in the course by applying and relating fundamental computational thinking concepts of algorithmic reasoning, data representation, and computational efficiency to real-world problems in the context of an embedded system, the Arduino. The Arduino platform provides a rich opportunity to engage students by showing broad applications of computing in domains that are part of their daily lives, thus introducing computing in a way that may improve student retention rates and encourage broader participation in computer science and engineering. Using this Arduino module in the course, we address two of the five main course objectives set forth for the course. Assessment results show that the approach has been effective. We present our experiences using the curricular material, as well as assessment results.

1 Introduction

Physical computing has been used to teach computational thinking, programming, and design skills. Researchers have examined the potential of physical

computing activities for computer science and engineering majors [2]. Rapid advances in technology and the associated reduced costs are making off-the-shelf, plug and use open source hardware and software platform with several supporting online resources [1]. In this study, we focus on the CS0 course for non-majors, where preliminary results in one course offering showed the potential of the Arduino platform in enhancing student learning [4, 3]. We extend this prior work and present experiences developing easily adaptable hands-on laboratory projects and associated curricular modules based on the Arduino platform, as well as experiences using this material in seven offerings of the CS0 course, over two years, 2017 and 2018. We engage students in a contextualized project-based learning experience and introduce them to fundamental computing concepts in the context of an interactive and easy to use environment. The curricular material is hands-on based and fosters close faculty-student interaction. In such an environment, students work in teams and are engaged in active learning. They are required to make their designs in a real-world setting. If their designs do not work, students receive instant feedback. In this process, they interact with the instructor and each other to troubleshoot their hardware and software. In addition to improving student learning, we believed this project has potential to improve student retention rates and encourage broader participation in computer science and engineering.

2 Integrating the Arduino Platform into CS0

The CS0 course provides a broad introduction to the use of computers as tools for creativity, problem solving, communications, and information organization. The CS0 course objectives are to: (1) Understand the parts of a computer and how they function and operate; (2) Form a basis for problem solving skills that evolve and adapt as technology advances; (3) Examine digital representation of information including data, sound, graphics, and video; (4) Create a fluency with foundations of communication and network infrastructure; and (5) Master advanced skills in productivity software.

The integration of these hands-on labs addresses the first two of these course objectives. Our goals for integrating the Arduino platform into CS0 are to: (1) Apply and relate fundamental computational thinking concepts of algorithmic thinking, data representation, and computational efficiency to relevant applications; (2) Become engaged with and understand the development of modern computing systems which have become part of students' daily lives; (3) Foster creativity by designing creative, tangible, and interactive computing-based systems that are personally and socially relevant; (4) Use of an interactive environment to better illustrate concepts covered in class, both hardware and software; (5) Integrate high-impact practices into the course; (6) Introduce

computing in a way that may broaden participation in computing; and (7) Align the current curriculum with the CS Principles framework to design a course that engages a broader audience.

3 Arduino Labs

All six labs are designed to incrementally introduce the student to the environment, hardware, and software. Each lab includes a set of objectives, items needed to complete the lab, and guided step-by-step instructions for students to follow. It concludes with open-ended questions and a reflection component. The series of labs leads to a cumulative project. The sections below present a summary of the six labs used. The complete set of labs are available at: http://rosiene.cs.hartford.edu/ArduinoLabs/2017/.

3.1 Getting Started

In this first lab, students run a basic program on the Arduino and learn how to use the IDE environment. They learn about the setup code and how to print. They first validate that the Arduino software is installed, connect the Arduino board to the computer through a USB cable, enter a basic program into the editor, compile, upload to the Arduino, and execute the program.

3.2 Introducing Variables and Output to the Serial Monitor

In the second lab, students are introduced to the concept of variables, to the basic software cycle of the Arduino, and become more familiar with the IDE environment. They use the serial monitor as a tool to interact with the Arduino processor while the programs are running on it; they use it for output or to write text to the computer screen. Several pre-defined functions are also introduced. Students modify lab 1 code to experiment with using these functions.

3.3 The Protoboard and Basic Electronic Circuits

In lab 3, students create a basic circuit to demonstrate the use of resistors and LEDs. Specifically, students produce a basic circuit to run one LED, and then expand the circuit by adding a second LED. Pushbuttons are then added to control the LEDs.

3.4 Arduino and Output to Components

In this lab, students are guided through writing and executing code that would (1) blink one LED, (2) blink two LEDs, and (3) sequentially blink LEDs for one second each with a half-second between each. Lab 4 demonstrates the use

of the Arduino to control LEDs. We can use the output pins on the Arduino board to control electrical components.

3.5 Arduino and Input from Components

Just as we could send signals to the electrical components of the Arduino, the Arduino can receive input from certain components as well. In this lab, students use the Arduino to read input from pushbuttons and then use "latching" to make programs remember which button was pressed. This stresses the purpose of a variable to remember whether a button was pushed or not.

3.6 Putting It All Together: Using the Arduino to Build a Random Light Game

This lab uses what have been learned and developed in previous labs to (1) randomly light up LEDs, (2) build the random light game, and (3) add a buzzer when the game ends.

4 The Arduino Platform as a Means to Course Objectives

The first two of the five course objectives listed earlier are addressed by the introduction of the Arduinos into our CS0 course. We have selected this platform to incorporate into the course for the following reasons: (1) Understand the parts of a computer and how they function and operate, and (2) Form a basis for problem solving skills that evolve and adapt as technology advances.

Clearly, the hands-on, interactive nature of these labs allows the student to make a connection between input and output, and demonstrate the purpose of memory. Students are able to make a direct correlation between how the program affects the result, thus, have a tangible, small computer to work with and understand how all of its parts interrelate and cooperate to function as a whole.

5 Assessment Results

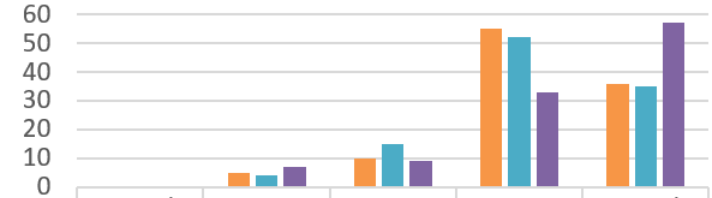
To evaluate students' reception to the Arduino component of the course, we designed a short survey given at the end of each semester. The survey includes three Likert-scale questions along with two qualitative responses.

The three Likert-scale questions included in the end of semester questionnaire are:

- Q1: Using the Arduino platform and labs enhanced my understanding of the hardware and software concepts covered in class.
- Q2: Using the Arduino microcontroller enhanced my learning experience in this course.

- Q3: The Arduino contributed to a positive learning experience in the course.

A total of 157 students enrolled in the seven sections of the course. With 106 student responses to the survey, a 68% survey participation rate, about 86% of the students agree that the Arduino labs contributed to their understanding of hardware and software concepts, over 82% indicate that the labs enhanced their learning, and about 85% say that the Arduino labs contributed to a positive learning experience. The results have been overwhelmingly positive, as can be seen in Figure 1.



	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Q1	0	5	10	55	36
Q2	0	4	15	52	35
Q3	0	7	9	33	57

Figure 1: Quantitative Survey Results

Included in the short survey are two open-ended questions giving students an opportunity to provide feedback:

- Q4: Describe what you liked most about the use of the Arduino platform and labs.
- Q5: Describe what you liked least about the Arduino microcontroller.

As would be expected, student interest and experience vary. Some thought the Arduino labs were a challenge, while others wanted to do more. Some thought the instructions were clear, some thought they were confusing. Overall, students enjoyed the interactive nature of the course and the visual environment. They enjoyed the hands-on experience, being able to apply the concepts learned in class, and being able to physically hold the objects and see the code working. While the student experience was overall positive, there were some challenges expressed in the responses to Q5. Students did not like sharing the units, 2-3 students per unit, and having the units stored in the

classroom/lab. This prevented students from taking units with them to work on after class. Based on student feedback and given the low cost of the units, at $35 per starter kit, the department decided to require students to purchase their own kits effective fall 2019. Given the fragility of some of the parts such as the resistors and wires, we plan to make sure we have enough extras on hand. While some students thought the step-by-step instructions were clear, others commented that they were, at times, not clear. Videos illustrating the step-by-step instructions appear to be a good addition to the written steps, which we plan to include in the future.

Based on the survey responses and informal interactions in class, the feedback has been very encouraging. Students enjoyed making something work and having a tangible product to prove it. They made the connection between hardware and software and saw the complexity of having to build a real-world system.

6 Conclusion and Future Work

We are using student and faculty feedback from these first offerings to help guide further development of this project and the associated curriculum modules and labs. As we develop additional material, we plan to continue to explore the use of the Arduino platform in the CS0 course, collect assessment data, and do further studies of the impact of this approach on teaching and on student learning. In addition to improving student learning, this project has potential to improve student retention rates and encourage broader participation in computer science and engineering. Beyond these courses, the proposed Arduino modules and labs have a number of different uses. An immediate consequence is that a number of systems will be set up in a lab to use as recruitment tools for computer science and engineering. These types of physical systems provide an application of engineering and computing principles that prospective students can appreciate.

References

[1] Arduino AG. Arduino homepage. `https://www.arduino.cc`.

[2] Evan Barba and Stevie Chancellor. Tangible media approaches to introductory computer science. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*.

[3] Carolyn Pe Rosiene Ingrid Russell and Aaron Gold. A cs course for non-majors based on the arduino platform. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 2020.

[4] Ingrid Russell, Karen H. Jin, and Mihaela Sabin. Make and learn: A cs principles course based on the arduino platform. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, 2016.

Using Animations to Teach String Matching Effectively*

Y. Daniel Liang[1], Lixin Li[1], Weitian Tong[2]
[1]Department of Computer Science
Georgia Southern University
{yliang, lli}@gerogiasouthern.edu
[2]Department of Computer Science
Eastern Michigan University
wtong1@emich.edu

Abstract

String matching is to find a substring in a string. The algorithms commonly used for finding a matching are the brute-force algorithm, Boyer-Moore algorithm, and Knuth-Morris-Pratt algorithm. The brute-force algorithm is intuitive. The Boyer-Moore and Knuth-Morris-Pratt algorithms are more efficient than the brute-force string matching algorithm, but they are more difficult to understand than the brute-force algorithm. We have created the animations for helping instructors to teach and students to learn these algorithms. This paper presents these animations.

Keywords: Algorithms, animation, brute-force algorithm, Boyer-Moore algorithm, data structures, KMP algorithm, string matching

1 Introduction

String matching is to find a match for a substring in a string. The string is commonly known as the text and the substring is called a pattern. String matching is a common task in computer programming. The Java String class has the **text.contains(pattern)** method to test if a pattern is in a text, the **text.indexOf(pattern)** method to return the index of the first match of the

pattern in the text, and the **text.lastIndexOf(pattern)** to return the index of the last match of the pattern in the text. The C++ string class has the **text.find(pattern)** function to return the index of the first match of the pattern in the text. In Python, you can use the in operator to test if a pattern is in a text (pattern in text) and use the **text.find(pattern)** function to return the index of the first match of the pattern in the text.

String matching is a common task in programming. A lot of research has been done to find efficient algorithms for string matching. The popular algorithms presented in many data structures and algorithm textbooks are the brute-force algorithm, the Boyer-Moore algorithm [3], and the Knuth-Morris-Pratt algorithm [6]. We have developed animations for teaching and learning these algorithms effectively. The animations show how the algorithms work step by step visually and interactively and help students to grasp the algorithms quickly. This paper presents these animations.

2 Comparing String Matching Algorithm Animations

Several string matching algorithm animations are available on the Web. The most popular ones are accessible from `https://people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html` [7] and `https://www.utdallas.edu/~besp/demo/John2010/boyer-moore.htm` [2].

The first tool [7] is an animation for the KMP algorithm. It lets the user to enter a text and a pattern and see how the KMP algorithm works in animation. It does not give an explanation for each step in the animation. Also the animation is not visually clear in the canvas. The second tool [2] is an animation for the Boyer-Moore algorithm. It lets the user to enter a text and a pattern and see how the Boyer-Moore algorithm works in animation. The user presses x to step forward and presses z to step back. It does not display how the pattern is shifted right.

Our string matching algorithm animation tools enable instructors and students to enter a text and a pattern. The user can click the Next button to see the next step in the algorithm. The animation displays an explanation for each step and reads the explanation in a computer-generated voice to guide the user through the steps in the algorithm. The tools in [2, 7] do not fit in a mobile device with a small screen. Our animation tools can fit well in a small mobile device and work with touch screens.

3 Brute-Force Algorithm Animation

The animation for the brute-force algorithm is accessible from `liveexample.pearsoncmg.com/dsanimation/StringMatch.html`, as shown in Figure 1.

The complete code implementation for all algorithms in this paper can be found at `liveexample.pearsoncmg.com/dsanimation/StringMatch.html`. The animation first prompts the user to enter a text and a pattern separated by a space. For simplicity, their sizes are limited to 20 and 7, respectively. For convenience, if no input is entered, a default text and a default pattern are provided. The default gives a worst-case example.

As shown in Figure 1, the user entered a text AAAADDAAAACDCCCDDABA and a pattern AADDAAA. Now the pattern is compared with the text starting from index 0 in the text as shown in Figure 2. You can click the Next button to see how the brute-force algorithm is used to find a match for the pattern in the text. For each step in the animation, an explanation for the step is displayed below the Next button. You can turn on or off the audio by clicking the Audio button on the upper left corner. When the audio is on, the explanation will be read by a computer-generated voice. In Figure 1, the audio icon indicates that the audio is muted. In Figure 2, the audio icon indicates that the audio is on. You can click the Reset button to restart the animation.

Figure 1.

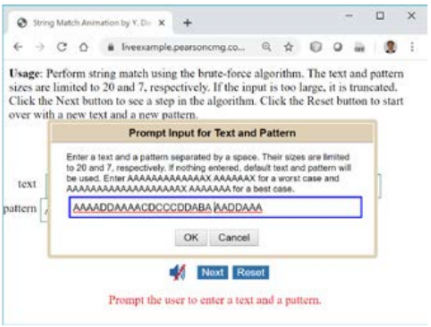
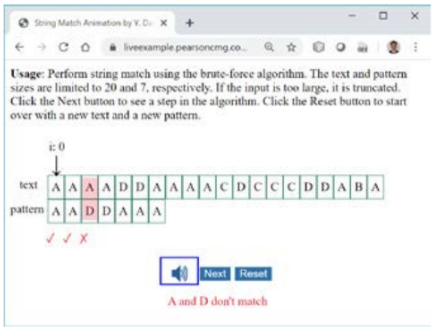


Figure 2.



4 Boyer-Moore Algorithm Animation

The brute-force algorithm is simple and intuitive, but not efficient. The algorithm searches for a match of the pattern in the text by examining all alignments. This is not necessary. The Boyer-Moore algorithm finds a match by comparing the pattern with a substring in the text from right to left. If a character in the text does not match the one in the pattern and this character is not in the remaining part of the pattern, you can slide the pattern all the way by passing this character. This algorithm can be best explored using an animation at `liveexample.pearsoncmg.com/dsanimation/StringMatchBoyerMoore.html`.

5 KMP Algorithm Animation

In the brute-force or the Boyer-Moore algorithm, once a mismatch occurs, the algorithm searches for the next possible match by shifting the pattern one position to the right for the brute-force algorithm and possibly multiple positions to the right for the Boyer-Moore algorithm. In doing so, the successful match of characters prior to the mismatch is ignored. The KMP algorithm takes consideration of the successful matches to find the maximum number of positions to shift in the pattern before continuing next search. To find the maximum number of positions to shift in the pattern, the KMP algorithm defines a failure function fail(k) as the length of the longest prefix of pattern that is a suffix in pattern[1 .. k]. The KMP algorithm is one of the most difficult subjects in the data structures and algorithms courses. An animation with various scenarios can help students to see how the algorithm works. We developed the animation accessible at `liveexample.pearsoncmg.com/dsanimation/StringMatchKMP.html`. The failure functions can be computed by comparing the pattern with itself. The animation for computing the failure functions is available at `liveexample.pearsoncmg.com/dsanimation/StringMatchKMPFail.html`.

6 Benefits

Here are the major benefits for instructors and students to use our string matching algorithm animations.

Benefit 1: In a typical lecture for introducing string matching algorithms, the instructor draws various types of texts and patterns on the board and shows the result of applying the algorithms by hand. This is a tedious and time-consuming process. The animation enables the instructor to create a text and a pattern dynamically and show the results of applying the algorithm step-by-step.

Benefit 2: Without the animations, we spent two lectures on string matching and drew various diagrams to cover all cases in the class. Now with the help of the animations, we can cover the three string matching algorithms in one lecture. The animations easily and effectively cover all cases visually in the algorithms. With the help of the animations, students can learn the subject quickly and better.

Benefit 3: A picture is worth a thousand words. An interactive animation is worth more than pictures. The interactive animation not only catches student attention in the class, it also engages the student with visual interaction. Students can use the tool to study before and after the lectures to see how an algorithm works on different texts and patterns.

Benefit 4: Our animation also serves as an example for students to write their

own programs to visualize the algorithms. This gives students the opportunity to get deeper into the algorithms and see how the algorithms work in their own animation. In our data structures and algorithms courses, we assign projects for students to write their own code for algorithm animation. Students like the algorithm animation projects. As supported in [8], students learn better when they actually implement the algorithms using animation.

7 Evaluations

Many algorithm animation tools are available. It is safe to say that algorithm animation assists instruction, but whether it helps students to learn is a mixed bag. Some experiments show positive student outcome [1, 8], while others say there are no significant difference to students whether animations are used or not [5]. An experiment conducted at George Washington University [4] showed that the students who used an interactive version of courseware spent more time and performed worse overall than those who used the non-interactive version of the courseware. The reason behind this is that the tools are ineffective and difficult to use. Our goal is to develop a simple tool that is effective and easy to use. First, our tool is free and directly accessible on the Web and can run on any device from a Web browser. There is no need to install any software. Second, our tool is intuitive and user friendly. It has only a short paragraph of instructions on how to use it. Third, our animation has an explanation for each step in the algorithm. Additionally, the explanation can be read in a computer-generated voice.

We use the animation in our data structures and algorithm course in Java and C++. The course covers recursion, Java generics, use of Java collections framework for array lists, linked lists, stacks, queues, priority queues, sets, maps, developing efficient algorithms, sorting, implementation of array lists, linked lists, stacks, queues, and priority queues, binary search trees, AVL trees, hashing, and graphs applications for unweighted graphs and weighted graphs. The string matching algorithms are covered in the context of developing efficient algorithms.

In the fall of 2015, we conducted a survey for a class of 22 students. We used a scale of 1 to 10 for answers, where 1 is poor and 10 is excellent. The result is as follows:

1. Does the brute-force string matching algorithm animation help you learn the algorithm? 9.5
2. Does the Boyer-Moore algorithm animation help you learn the algorithm? 9.1.
3. Does the KMP algorithm animation help you learn the algorithm? 8.5.

4. Does the KMP failure algorithm animation help you learn the algorithm? 8.3.

The survey strongly suggests that the tool is easy to use and helps students learn the string matching algorithms.

Over the years, we have improved and enhanced the animation based on the feedback from instructors and students. The explanation and the audio to read the explanation were the result of recent enhancement.

In the fall of 2018, we conducted a new survey with 21 students using two simple questions. With a scale of 1 (poor) to 10 (excellent). The result of the survey is as follows:

1. Are the animations intuitive to use? 9.8
2. Are the animations helpful to study the string matching algorithms? 9.2

References

[1] John Bazik. Software visualization in teaching at brown university. *Software visualization*, pages 383–398, 1998.

[2] Sergey Bereg. Boyer-moore exact pattern match. `https://www.utdallas.edu/~besp/demo/John2010/boyer-moore.htm`.

[3] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, October 1977.

[4] Duane J Jarc, Michael B Feldman, and Rachelle S Heller. Assessing the benefits of interactive prediction using web-based algorithm animation courseware. *ACM SIGCSE Bulletin*, 32(1):377–381, 2000.

[5] Michael D Byrne1 Richard Catrambone John and T Stasko. Do algorithm animations aid learning? 1996.

[6] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[7] Yves Lucet. Computer science department, university of british columbia (okanagan campus). `https://people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html`.

[8] John T Stasko. Using student-built algorithm animations as learning aids. In *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, pages 25–29, 1997.

Creating a More Equitable CS Course through Peer-Tutoring *

Adamou Fode Made[1], *Abeer Hasan*[2]
[1]*Department of Computer Science*
[2]*Department of Mathematics*
Humboldt State University
{adamou.fode, abeer.hasan}@humboldt.edu

Abstract

This paper describes the effects of a newly implemented peer-tutoring program at Humboldt State University. While the overall benefit of tutoring in students' learning is documented in research, we aim to look into the impact peer tutoring has in closing achievement gaps and creating a more inclusive learning environment. We collected and analyzed three semesters worth of students' data. Statistical methods were used to test whether tutoring improves students' success rate in one of our gateway and bottleneck courses – Computer Science Foundations 2. Our analyse suggest that the peer tutoring program has narrowed the achievement gap for Underrepresented Groups (URGs), Pell-Grant recipients, Females, and First-Generation students. Overall, tutored students had a success rate that is 17% higher than untutored ones.

Key words: CS1, CS2, Improving Student Success Rate, CS Education, URGs, Peer Tutoring.

1 Introduction

Like many institutions, our computer science program has gateway courses that impede progress for all students and in particular, URGs. Our Computer Science Foundations 2 (CS2) course has traditionally been such a gateway course. To address this, we decided to start a peer-tutoring program in spring

2018 for all our bottleneck and gateway courses. All the students in our CS2 have at least a minor in Computer Science.

Tutoring is a process in which expert and trained people help and support people who are less skilled, in an interactive, meaningful and organized way. Peer tutoring is a teaching strategy where a student tutee interacts with one or more student tutees. The most common approach to peer tutoring is when an experienced student assists one or more learners outside of class time. This is the approach which has been described in this article.

Research has shown that peer tutoring can facilitate improved competence when the peer tutor is a more advanced student than those who are tutored [4]. Peer tutoring is known to help increase students retention rate [5] as well as the overall success rate [2]. However, merely making tutoring available does not necessarily guarantee success. Success also requires mentorship for the tutors, dedication from the students requesting assistance, and possible adjustments to the tutoring format [3]. An earlier study [1] of a similar peer tutoring program showed that the program had closed the achievement gap between Underrepresented Groups, Females, First Generation, and Financial Aid recipients versus the overall students' population in a Discrete Mathematics course. In this paper, we replicate that study for our CS2 course.

Our University is classified as a Hispanic-Serving Institution, and first-generation college students are well represented in our classrooms. In 2014, URGs, undeclared and non-URGs represented respectively 30%, 11% and 59% of our students by Fall 2018, those numbers were 41%, 10% and 49%. We have seen a growing proportion of UGR students over time. As our student population is becoming more diverse, there is a greater need for programs such as peer tutoring to help close the achievement gaps.

In this paper, we report on a comparison of student performance across three sections of Computer Science Foundations 2 taught by one faculty with a combined enrollment of 86 students. We aim to answer the following questions: "Does peer tutoring improve students' success rates and help close the achievement gap for students from underrepresented groups (URG)?"

2 Methodology

We started a peer-tutoring program for our CS2 course in Spring 2018. The program ran for three consecutive semesters (Spring 2018, Fall 2018, and Spring 2019) and will continue for the foreseeable future pending on available funding. Students had the option of attending free peer tutoring services when they needed help. Tutoring was offered on a first-come-first-served basis, and was available Monday through Thursday, mostly in the evening, for a total of 20 hours per week. Two tutors were available to help students in each shift. We

kept track of tutoring attendance data. Only 31 out of 86 students did not participate in the tutoring program. Our CS program had an enrollment of about 180 students per year for the last five years.

The tutor helps only one student at a time and restricts the contact to no more than five minutes when the center is busy. A student is considered tutored if they attended two or more sessions in the semester for a total of two contact hours. The tutors were instructed to guide students to solutions through questions and similar examples without solving the homework for them. Tutors do not just review homework, but help students develop confidence in reaching their answer and becoming independent learners. Each tutor was required to participate in a 16-hour training workshop. Experts at the University Learning Center conducted the training. Tutor training covered the following topics: professionalism, ethics, learning theory, tutor cycle, creating a welcoming environment, challenging scenarios, students of concern, and supplemental instruction showcase.

Currently, the tutors are expected to help with more than three courses, including all of the 100-200 level CS department courses. All of the tutors have complete our gateway courses (CS2 and Discrete Mathematics) with a grade of A- or higher. We started with four tutors the first semesters (All males, one URG), and we now have eight (five males and three females) with a combined four Latinx students (URG) and one female (not URG). We tried to hire a diverse group of tutors, for example, 38% of the tutors were female and 50% were Latinx students.

The same instructor taught the three offerings of the CS2 course and using the same textbook, Problem Solving with C++, 9th edition by Walter Savitch. The instructor used similar teaching materials, including syllabi, equivalent homework assignments, and study guides. Further, the instructor crafted the exams with an effort to keep problems at similar difficulty levels without reusing previous exam problems. However, since students had no access to final exams from previous semesters, the final exam content was relatively the same in the three consecutive course offerings.

3 Results and Statistical Data Analysis

Table 1 shows a comparison of success rates between tutored and untutored students. We can see that students who used tutoring services had a higher success rate (81.8%) compared to those who did not (64.5%). The sample difference in the success rate between tutored and untutored students in our study is 17.3%. Wald's 90% confidence interval on the difference in success rates for tutored minus untutored students is (0.008, 0.338). Fisher's exact test on the counts in Table 1 resulted in a p-value of 0.064, which is significant

at the level 0.10 [6]. While we cannot claim that tutoring is the sole cause of this increase in the success rate, it certainly helped those who utilized it and the 17% increase in success rate is practically important.

Table 1: Student Success Rates by Tutoring Participation.

	Failed Count	Passed Count	success Rates %
Tutored	10	45	81.8
Not Tutored	11	20	64.5

Table 2: Student Success Rates Grouped by Different Factors with the p-value from Fisher's Exact Test on each Factor. Total sample size = 86

Risk Factors	Student Group	Fail	Pass	Success Rate%	P-value
First Generation College Students	First Gen.	15	36	70.6	0.537
	Not First Gen.	5	23	82.1	
	Undeclared	1	6	85.7	
Legal Sex	Female	7	18	72.0	0.781
	Male	14	47	77.0	
Financial Aid	Received	16	37	69.8	0.131
	None	5	28	84.8	
Underrepresented Groups	URG	8	26	76.5	0.892
	Not URG	4	10	71.4	
	Undeclared	9	29	76.3	

We are passionate about reducing achievement gaps for students from underrepresented minorities and creating an equitable learning environment. We examined the success rates in the last three offerings of the course and compared success rates across different groups of students. Table 2 shows a summary of student success rates versus some risk factors that are known to impede student success. Due to our small sample counts, we chose to conduct Fisher's exact test on each factor. All p-values are reasonably large. Thus there is no indication of an achievement gap for Female, Underrepresented Groups or Pell Grant recipients. We see this as a good sign to indicate that tutoring might have helped close the achievement gap for disadvantaged students.

Finally, we compared the demographics of tutored and untutored students. Our research question was to investigate if there is a significant difference in tutoring participation between different student demographics. Table 3 summarizes tutoring participation across different student demographics. Sample proportions indicate that students from URGs and first-generation college students had the lowest participation rates in the tutoring program. The combi-

nation of small sample counts in some cells in Table 3 and students with undeclared URG and first-generation status makes it harder to compare tutoring participation rates across different student groups. However, reasonably large p-values suggest that the observed sample differences may be insignificant.

Table 3: Tutoring Participation by Student Demographics

Risk Factors	Groups	Not Tutored	Tutored	Rate %	P-value
First Gen.	First Gen.	21	30	58.8	0.374
	Not First G.	9	19	67.9	
	Undeclared	1	6	85.7	
Legal Sex	Female	7	18	72.0	0.459
	Male	24	37	60.7	
Financial Aid	Received	20	33	62.3	0.818
	None	11	22	66.7	
URGs	URG	17	21	55.3	0.273
	Not URG	9	25	73.5	
	Undeclared	5	9	64.3	

4 Conclusions and Limitations

The quantitative analyses of our student data have demonstrated that peer tutoring is likely to have a positive impact in increasing success rates among all students and reducing achievement gaps for students from underrepresented groups. We believe that tutoring has helped create a sense of community, confidence, and success in our courses. Further studies are warranted to see the long-term effect of peer tutoring as student progress towards upper-division courses where tutoring is not offered.

We note that our peer tutoring study suffers from limitations common to research in small computer science programs. In particular, the data did not come from a completely randomized experiment. The self-selection process of attending tutoring makes it difficult to account for lurking factors like student motivation and perseverance. Typically, we expect struggling students to seek tutoring help. Thus, it is not fair to compare exam scores for tutored and untutored students.

We established that tutoring services had helped increase success rates across all student demographics, helped reduce the achievement gaps for females, URGs, low income (Pell grant recipients), and first-generation college students. Our data indicate that all student demographics utilized and benefited from tutoring. Student motivation may be a confounding factor that caused students to participate in tutoring and study for the course. However,

the availability of tutoring could also increase student motivation by boosting student confidence. We suspect that struggling students find it easier to ask a colleague for help than to raise questions in class or during office hours. Some students may be reluctant to seek help from their instructor. Tutoring provided such students with a helper who will not evaluate their performance.

References

[1] A. Fode Made, A. Hasan, S. Burguess, D. Tuttle, and N. Soetaert. The Effect of Peer Tutoring in Reducing Achievement Gaps: A success Story. *Journal of Computing Sciences in Colleges*, 35(1.), 2019.

[2] R. Garcia, J. C. Morales, and R. Gloribel. The Use of Peer Tutoring to Improve the Passing Rates in Mathematics Placement Exams of Engineering Students: A Success Story. *American Journal of Engineering Education*, 5(2.), 2014.

[3] G. Huang, N. Taddese, E. Walter, and S. Peng. Entry and persistence of women and minorities in college science and engineering education. *National Center for Education Statistics.*, (601), 2000.

[4] P. Kalkowski. Peer and cross-age tutoring. *School Improvement Research Series*, 1995.

[5] S. Menzel and J. Cottam, A. J.and Greenblatt. Tutoring for Retention. *SIGCSE Proceedings of the 42nd ACM technical symposium on Computer science education*, (213-218.), 2011.

[6] J.G Tshikuka, M. G. Magafu, M. Molefi, T. Masupe, R. B. Matchaba-Hove, B. Mbongwe, and R. Tapera. *Addressing the Challenge of P-value and Sample Size when the Significance is Borderline: The Test of Random Duplication of Participants as a New Approach.*, volume 5. 2016.

Curated Pathways to Innovation: Personalized CS Education to Promote Diversity*

Natalie Linnel[1], Alankrita Dayal[2], Phil Gonsalves[2], Mayank Kakodkar[3], Bruno Ribiero[3], Ariel Starr[2], Tim Urdan[1], Janice Zdankus[4],
[1]Santa Clara University, Santa Clara, CA 95053
{nlinnell, turdan}@scu.edu
[2]YWCA Silicon Valley, San Jose, CA 95112
{adayal,pgonsalves,astarr}@ywca-sv.org
[3]Purdue University, West Lafayette, IN 47907
{mkakodka, ribiero}@purdue.edu
[4]Hewlett Packard Enterprise, San Jose, CA 95002
{janice_zdankus}@hpe.com

Abstract

Curated Pathways to Innovation (CPI^{TM}) is a web tool that gathers existing online resources for computer science (CS) engagement and learning, exposing students to CS careers and content, especially targeting K-12 girls and under-represented minorities. CPI uses a machine learning recommender to customize content, and is a collaboration of academics in CS and social science, $K-12$ educators, non-profit, and industry. We have deployed this tool in a low-income, primarily Latino/a middle school with nearly 500 students for over two years, in addition to lower-touch deployments at three high schools. We have also created in-person experiences for the students, such as reverse science fairs and hackathons, also tracked in CPI. This paper focuses on our experiences deploying the system, as well as the tool itself.

1 Introduction

Motivated by the lack of engagement of underrepresented minorities (URMs) and women in STEM [8], we have launched Curated Pathways to Innovation (CPI^{TM}), a web app to help students, particularly girls and URMs, navigate their journeys to STEM (especially computing) careers.

Our system addresses limitations of many such interventions. Most engage the student only for a short time; a single experience is unlikely to lead to a career. Interventions often focus on interest in computing, without preparing kids for the technical content they'll need to master. Finally, few interventions do longitudinal tracking. The goal is for CPI to follow students from the time they start using it, preferably in elementary school, through college and beyond. As they begin discovering CS through CPI, they are awarded badges as they complete activities which largely draw content from existing resources like code.org, Khan Academy, etc, but which also include content we have generated around helping diverse students see themselves as computer scientists, and even in-person experiences like hackathons. CPI provides context and continuity to these disparate experiences. As students get older, we will add more advanced content, and will also begin to track their academic progress in CPI, as well as connecting them with mentors and networking opportunities. Toward this ambitious goal, we have assembled a diverse team. Our leader comes from industry, and academic teams develop the recommender and the website. Deployments and curriculum are overseen by YWCA-Silicon Valley ($YWCA-SV$). Finally, a social scientist designs and analyzes data collection. We discuss curriculum and the system in Sec. 3. Our main focus in this paper is discussion of deployments and lessons learned in Sec. 4.

2 Related Work

We explore our curriculum's relationship to the literature in Sec. 3.2. Here we discuss software designed to get kids, especially girls and URMs, interested in computing, such as Alice [4], code.org and Scratch, or teach programming, like Codecademy.com or W3Schools.com. What sets CPI apart from these is how we bring resources together in one place, with motivating activities, to allow students to plan and track their progress, coupled with machine learning to measure efficacy and provide recommendations to students. iRemix [7] lets educators to create a social network where students create and share media and comment on them. iRemix is not focused on CS and does not include specific activities. Other systems that aggregate materials are The National Girls' Collaborative [2], which facilitates organizations working to bring girls into STEM careers to share resources and best practices, and Engage CSEdu [3]

which allows CS educators to share materials. Both are resources for educators, not for students to interact with. LRNG [1] has 'playlists' of online and real-life educational experiences targeted at different cities, allowing students to earn badges; it does not include CS activities. None of these systems use machine learning for tracking or recommendation.

3 The System

3.1 The CPI Workflow

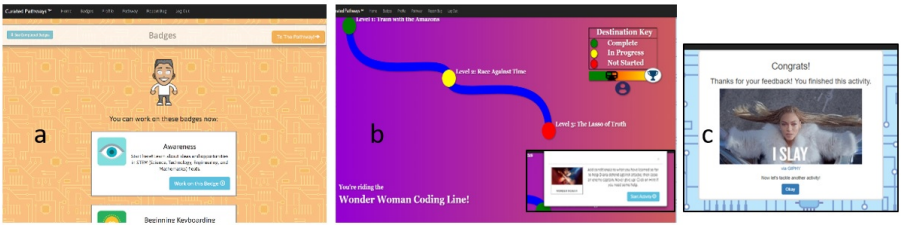


Figure 1: (a) Students choose a badge. b) Students are shown their progress through the badge's activities. b-inset) Clicking on an activity, the student is given instructions, then they are usually taken out of CPI to do the activity. After completing the activity, they return to CPI to rate the activity and take a quiz to certify completion. c) If successful, they're shown a gif related to an area they've expressed interest in (like Beyonce), and sent back to the Pathway page (Fig 1b) to continue the badge. After completing all activities, the student gets credit for the badge. Some badges have prerequisites, so completing one may unlock more.

Recently, we added avatars (Fig. 1a) and options for customizing avatars as rewards for completing badges. We also added a portal allowing teachers to track student progress (Sec. 4). The recommender customizes students' experience by placing an ordering on available badges. It uses demographic data, answers to survey questions, and usage data from CPI. The recommender team generates weekly reports to monitor any potential negative bias.

3.2 Curriculum

When choosing activities for CPI, we follow best practices for encouraging females in STEM, many of which also apply to URMs.[8] One recommendation is to expose students to diverse role models; for example, we include a coding activity centered on Wonder Woman, an activity reflecting on a trailer for

the movie Hidden Figures, and interviews with diverse programmers. Also recommended is emphasizing effort over talent, and breaking down stereotypes. We include motivational popups that directly address the attitudes of female/URM students, as well as students outside these groups. Activities using programming in hip hop, dance, and fashion, or that let them build bridges or move gears to make a machine, focus on how STEM applies to real-world applications, another recommendation. CPI pedagogy is based on a philosophy of "learn by doing", as research shows that active learning is motivating to CS learners. [6] CPI takes students through three levels: Awareness, Cultivating Interest, and Preparation. Awareness activities are focused on engagement and exposing students to STEM careers. Activities include videos of famous people, such as the founder of Facebook, talking about the importance of STEM, and infographics about opportunities and salaries for women and URMs in STEM. Cultivating Interest activities help students see themselves in STEM through games where they gain confidence and begin learning simple programming. Activities include coding Angry Birds through mazes and coding Pixar characters. Preparation activities are more challenging, and require students to engage in a series of tasks preparing them for STEM careers. They help students develop mindsets that are conducive to pursuing STEM, but also teach concrete skills like Python. CPI's activity-badge structure and Awareness-Interest-Preparation progression echoes research showing that structured learning environments help students succeed in STEM. [6] Recently, we have expanded our offerings to include content in biology, chemistry, and physics, as well as basic computer skills like keyboarding, as a lack of basic skills can form an emotional block to learning coding. [5]

4 Deployments

We are deployed in a low-income public middle school with over 470 6th-8th graders, and over 70% Latino/a students. We chose middle school as it is where career aspirations start to form and interest in STEM begins to diverge by gender.[8, 4, 5] For two and a half years, every student has spent one class period per week using CPI. Before partnering with us, the school had difficulty sourcing quality STEM curriculum. Unless noted, we discuss this deployment. We also have lower-touch deployments with three high schools, including the high school where most of our middle schoolers continue.

4.1 Teacher Engagement

Starting the deployment with the whole school had benefits and drawbacks. It was helpful for getting feedback and iterating on the system and deployment, but we were concerned about teacher buy-in, as the decision was made by the

school. We used several strategies to engage individual teachers and students. We created in-person activities (Sec. 4.2), providing tangible value to teachers, and provided incentives to students (Sec. 4.3). Prior to launch, all teachers received CPI training. A YWCA-SV staff member served as the CPI facilitator and was on-site whenever students were using CPI. We provided volunteers in almost every class session where CPI was used, helping students and teachers learn the system, and providing technical support.

While many teachers became engaged with CPI, some didn't. CPI's design for independent learning, coupled with the use of volunteers at the beginning of the deployment, led some teachers to treat CPI as an activity they weren't involved in; some even used this as prep time. In addition, due to CPI's personalized learning focus, students tend to be working on different activities at any moment, making it difficult for teachers to connect CPI to the other content they are teaching - when many of the teachers do not feel confident teaching CS in the first place. To increase teacher ownership, we decreased volunteer time, though the facilitator was still on site for trouble-shooting. We developed a teacher portal for the web app, allowing teachers to monitor student progress, and trained teachers with it; then the facilitator met with each teacher individually, in addition to classroom visits. Early in the last two school years, we gave teachers a review of the prior year's results and student progress, emphasizing the need for longitudinal study and discussing the above issues. We are also working with teachers to develop badges related to content from outside CPI, to better integrate CPI into the classroom.

4.2 In-Person Interventions

Beyond online activities, we've hosted several events; a reverse science fair where volunteers from industry set up displays at the school about their work projects, a summer math camp, and for two years we've organized hackathons at the middle school and one of the high schools. With an eye to scale and with YWCA-SV's leadership, we've begun working with partners to get our students access to existing programs, including a library's mobile maker lab, a community college's summer camps, a tech company's mentorship program, and other YWCA-SV programs.

4.3 Student Incentives

To motivate students, we've provided incentives including gift cards and sports memorabilia. While incentive decisions were made collaboratively with teachers, we have found incentives problematic. The largest problem is that they unintentionally motivated students to cheat. Students can skip the activity, going straight to the comprehension quiz. Quizzes are not long, so students

can answer the questions repeatedly to find the right answers. To combat this, we have made some changes. First, a teacher can now use their portal to see how many times a student has attempted a quiz, and are notified by email the second time a student fails a quiz. We don't allow a quiz to be started until at least half the average completion time for that activity has passed. Finally, we require students to get all questions right to receive credit. We also became concerned that incentives might reduce students' intrinsic motivation to learn CS. However, the students expect incentives, so to abruptly remove them would lead to resentment. We have decided to phase them out. Now, classes compete with each other on CPI progress, and winning classes receive a party. We also believe adding avatars,(Sec. 3) with avatar customization as an incentive for progress, is a better incentive as it ties students' motivation more closely to their progress.

4.4 Data Collection on STEM Interest/Confidence

Our social scientist designed a method where we initially surveyed students with a long baseline, followed by periodic shorter surveys. However, stopping classes to survey hundreds of students, and tracking absences to ensure make-ups, has made implementation difficult. The number of students who have taken all of our surveys is too small to reach statistically significant conclusions. Our social scientist has re-designed data collection to balance dimensionality with completeness. We've kept the baseline, but replaced the shorter surveys with questions asked alongside comprehension quizzes. With data collection integrated with the system, we expect more complete data.

5 Conclusions

Curated Pathways to Innovation is a web app that uses machine learning to customize students' experience of discovering and learning CS. We report on our experiences working with a large, diverse team spanning the academy, non-profit, and industry to deploy CPI in a low-income, primarily Latino/a middle school with almost 500 students. We discussed lessons learned around teacher engagement, student incentives, and data collection. We have re-worked our deployment and data collection models and look forward to realizing our goal of a system that allows us to use longitudinal data to evaluate and improve the recommender, the individual activities, and the system itself.

References

[1] Lrng. https://www.lrng.org/.

[2] National girls collaborative project. https://ngcproject.org/.

[3] Ncwit's engage csedu. https://www.engage-csedu.org/.

[4] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. Storytelling alice motivates middle school girls to learn computer programming. pages 1455–1464, 01 2007.

[5] J. Shah P. Morreale. M. Andujar, L. Jimenez. Evaluating visual programming environments to teach computing to minority high school students. *CCSC: Southeastern Conference*, JCSC 29,2, 2013.

[6] Pooja Sankar, Jessica Gilmartin, and Melissa Sobel. An examination of belongingness and confidence among female computer science students. *ACM SIGCAS Computers and Society*, 45:7–10, 07 2015.

[7] Anni Sapountzi and Kostas Psannis. Social networking data analysis tools challenges. *Future Generation Computer Systems*, 10 2016.

[8] Ming-Te Wang and Jessica Degol. Gender gap in science, technology, engineering, and mathematics (stem): Current knowledge, implications for practice, policy, and future directions. *Educational Psychology Review*, 29, 01 2016.

Google Tech Exchange: An Industry-Academic Partnership that Prepares Black and Latinx Undergraduates for High-Tech Careers*

April Alvarez[1], Legand Burge[2], Shameeka Emanuel[1], Ann Gates[3]
Sally Goldman[1], Jean Griffin[1], Harry Keeling[2], Mary Jo Madda[1]
Bianca Okafor[1], Alycia Onowho[2], Gloria Washington[2]
[1]Google LLC
Mountain View CA 94043
jeangriffin@google.com
[2]Howard University Computer Science Program
Washington D.C. 20059
blegand@scs.howard.edu
[3]University of Texas at El Paso Computer Science Department
El Paso, TX 79968
agates@utep.edu

Abstract

This paper describes Google Tech Exchange, an industry-academic partnership that involves several Historically Black Colleges and Hispanic Serving Institutions. Tech Exchange's mission is to unlock opportunities in the tech industry for Black and Latinx undergraduates. It is an immersive computer science experience for students and faculty. Participants spend a semester or two at Google in Silicon Valley taking or co-teaching computer science courses, including cutting-edge ones not offered at many universities. The 2018-2019 graduates especially valued the community-building, and a high percentage secured technical internships or jobs.

1 Introduction

Graduates of Computer Science (CS) and Software Engineering programs often have gaps in knowledge, experience, and enculturation when interviewing for, or starting, jobs in the software engineering industry [3, 6, 7, 14]. Approaches to closing these gaps include apprenticeships, co-ops, and internships. Tech Exchange takes an alternative approach, where undergraduates spend a semester or two in Silicon Valley taking cutting-edge CS courses co-taught by Google employees. It also addresses another gap, the underrepresentation of Black and Latinx professionals in the U.S. tech industry [10, 1, 18].

This experience report is organized as follows. The Background section discusses gaps in undergraduate preparation for jobs in the software engineering industry, and the underrepresentation of Black and Latinx technologists. The next section describes the Tech Exchange 2018-2019 Model. The Lessons Learned and Future Plans section is followed by a Discussion section.

2 Background

This section discusses gaps in undergraduate preparation for software engineering jobs, approaches to bridge the gaps, and Black/Latinx underrepresentation.

CS students fall short of industry expectations in technical interview preparation, written and oral communication skills, and project management skills [3, 4, 13, 14]. Although many qualities of a great software engineer involve soft skills, CS and Software Engineering education programs typically focus primarily on technical knowledge [8]. Several approaches to experiential learning attempt to bridge the gap between academia and the software engineering industry. These include internships, project-based courses (Fox Patterson 2012) [5], service learning courses [7, 11], and industry-academic partnerships such as co-ops and apprenticeships. Co-op participants typically spend one or more semesters (or quarters) working for an employer in partial fulfillment of degree requirements. Apprenticeships often involve a relationship that spans multiple years, such as the UK Degree Apprenticeships [2, 9].

These approaches have limitations. Co-ops and apprenticeships are immersive but are only offered by a small number of institutions. Project-based courses and service-learning courses are valuable but not immersive. Internships are immersive but there are significant hurdles to obtain them, and they do not provide a consistent set of educational experiences that prepare undergraduates for the software engineering industry. More innovative programs are needed.

Gaps in preparation are especially problematic in the U.S. for those who are Black or Latinx due to underrepresentation in the tech industry [10, 1, 18]

compounded by stereotype threat [15]. Immersive, rigorous, and welcoming educational experiences are needed to provide technical and social preparation.

3 The 2018-2019 Tech Exchange Model

Tech Exchange is an industry-academic partnership that involves Google and several Historically Black Colleges (HBCUs) and Hispanic Serving Institutions (HSIs). It is one way that Google makes a long-term investment in CS education to increase pathways to high-tech careers for underrepresented groups. Tech Exchange is an immersive program where junior CS majors spend a semester on Google's campus in Silicon Valley, take state-of-the-art CS courses co-developed and/or co-taught by Googlers and HBCU/HSI faculty, and engage in social opportunities that foster professional development. It builds on the success of prior partnerships, Howard West and Googler-In-Residence [16]. In 2018-2019, 5 HBCU/HSI faculty participated, and 65 students (41% female) from 10 HBCU/HSIs (38 both semesters; 18 Fall-only; 9 Spring-only).

Contractually, Tech Exchange is a partnership between Google and Howard University. Howard sets academic policies, hires faculty, registers students, awards credit, and has domestic exchange agreements with the other universities. Google's 2018-2019 team had a Program Lead, a Community & Operations Manager, and a part-time Creative Strategy Manager (from Google Education Equity), and 2 part-time Academic Program Managers (from Google Research).

Tech Exchange hosted weekly community meetings and 30+ social and career development events including field trips, team-building activities, and guest speakers. Over 100 Googlers volunteered as instructors, TAs, mock interviewers, mentors, and event hosts. Social media campaigns that showcased student included a CS Ed Week campaign (on LinkedIn, Instagram, and Twitter), a YouTube video *Inside Google's Tech Exchange Program*, and a Google blog post *Tech Exchange students reflect on their future careers.*

Tech Exchange's academic goals are to: 1) provide innovative and engaging courses that prepare students for careers in the software engineering industry; 2) establish collaborative working relationships between Google volunteers and the HBCU/HSI faculty to create/revise course content and pedagogies to align with current industry practices; 3) encourage HBCU/HSI faculty to bring back courses and pedagogies to their institutions. The 2018-2019 program spanned the Fall 2018 and Spring 2019 semesters. Twelve technical courses were offered:

- Algorithms, Cybersecurity, Databases, Data Science, Entrepreneurship
- Interview Prep, Machine Learning, Mobile Apps, Product Management
- Programming Languages, Software Engineering, Theory of Computation

Most courses met 2x/week and earned 3-credits. Each had a teaching team with one or more TAs. Most had both an HBCU/HSI faculty member and a Google instructor. There were 5 HBCU/HSI faculty members and 10 Google instructors. The teams met before the start of the semester and weekly throughout. Active learning was highly encouraged, using techniques such as whiteboarding, group projects, discussion slides, pair programming, and peer instruction.

Tech Exchange 2018-2019 was evaluated internally by administrators from Google and Howard. Each semester students completed mid-semester anonymous course feedback surveys, developed by Google. In the spring, all students completed a draft version of the Basic Data Structures Inventory [12], and 33 students completed a post attitudes survey about career readiness and Tech Exchange (developed by Google and Howard). The course teams completed Carl Wieman's Teaching Practices Inventory at the end of each semester as a reflection exercise [17]. At the end of the spring semester, Google held a focus group for the HBCU/HSI faculty and another one for the Google instructors and TAs. All Google volunteers had the opportunity to complete a mid-semester survey in the fall to gauge their attitudes about Tech Exchange.

4 Lessons Learned and Future Plans

This section discusses lessons learned from 2018-2019 and future plans regarding the administration, community, social media, and academics.

The administrative structure outlined above will remain the same, but better planning and communication are needed. The community-building aspects were generally successful. In the attitudes survey completed by 33 students at the end of the Spring semester, 48% reported that the sense of community was the best part of Tech Exchange. In a separate survey of Google volunteers 75% of the respondents reported that volunteering for Tech Exchange had a positive impact on their view of their employer. Students enthusiastically participated in the social media campaigns and continue to stay connected on social media.

Tech Exchange will transition to a one-semester, spring-only program. In the fall, teachers and administrators can plan, and students can take core courses at their home institutions. Regarding student selection, the intention was to recruit juniors with a high GPA in a CS-related major. In the future the prerequisites will be better clarified and the selection process will include a technical interview. The Academic Program Managers used a draft version of the Basic Data Structures Inventory (BDSI) [12] as a guideline to determine the new prerequisites. They also administered this concept inventory in the spring and analyzed performance trends. This helped to shape the student selection process for 2020.

In the future fewer courses will be offered, primarily electives where Google

can add the most value. A key finding was that students need more practice with technical interviews. Instead of the 2018-2019 1-credit, 1x/week Interview Prep course, a 3-credit, 2x/week Applied Data Structures course will be offered that includes interview prep. To alleviate burnout, exam/project deadlines are now staggered, and students will agree to take no more than 15 credits.

The HBCU/HSI faculty made recommendations during their focus group: R1) more planning time; R2) clarified expectations; R3) faculty office; R4) better coordination of guest speakers; R5) more professional development; R6) greater access to Google tech talks and events. R1&2 have been addressed by reserving fall for planning. R3 has been accomplished. R4 is the responsibility of each course team. R5 will be addressed with a more in-depth orientation. R6 is not possible because the faculty are not Google employees.

The Google instructors and TAs expressed widespread agreement, in their focus group of $\tilde{2}0$ participants, that Tech Exchange was a rewarding experience. They suggested better communication across courses, more time for course planning, an improved selection process for students and teachers, and clearer guidelines regarding roles and responsibilities. Many expressed frustration with the learning management system (Blackboard); some students and Googlers were never able to access it. In the future, Google Classroom will be used. Several thought that some students seemed underprepared and that some students had unusually heavy course loads.

Thirty-three students completed a survey after the Spring semester. Their responses were strong (median=4 on a Likert scale of 1-5) regarding their opinion of Tech Exchange overall, their likelihood to succeed in a tech career and accept a job in Silicon Valley, and feeling welcome with a sense of belonging in Silicon Valley. At least 44 of 65 obtained technical internships or jobs by summer; 32 in the tech industry (15 at Google); 12 tech-related in other industries. Some benefits were unanticipated. For example, some graduates on their own initiative started clubs at their schools to help students practice for technical interviews.

5 Discussion

This paper describes the motivation for Tech Exchange and compares it to other industry-academic partnerships. It describes the 2018-2019 program, lessons learned, and future plans. Several measures indicate that the 2018-2019 program was successful. Students reported a high degree of satisfaction and a high degree of confidence about succeeding in a tech career. Over two thirds of them obtained technical internships or full-time jobs by the summer. Despite the many challenges encountered during the 2018-2019 year, Google, Howard, and the other participating universities will continue to invest in Tech Exchange.

References

[1] Women, Minorities, and Persons with Disabilities in Science and Engineering: 2011. NSF 11-309. Technical report, National Science Foundation, Division of Science Resources Statistics, 2011.

[2] Matthew Barr and Jack Parkinson. Developing a Work-based Software Engineering Degree in Collaboration with Industry. pages 1–7, 2019.

[3] Andrew Begel and Beth Simon. Struggles of new college graduates in their first software development job. In *SIGCSE'08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education*, pages 226–230. ACM, 2008.

[4] Denae Ford, Titus Barik, Leslie Rand-Pickett, and Chris Parnin. The tech-talk balance: what technical interviewers expect from technical candidates. *Proceedings - 2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2017*, (9408):43–48, 2017.

[5] Armando Fox and David Patterson. Viewpoint: Crossing the software education chasm. *Communications of the ACM*, 55(5):44–49, 2012.

[6] Vahid Garousi, Gorkem Giray, Eray Tuzun, Cagatay Catal, and Michael Felderer. Closing the Gap Between Software Engineering Education and Industrial Needs. *IEEE Software*, 2019.

[7] Wendy A. Lawrence-Fowler, Laura M. Grabowski, and Christine F. Reilly. Bridging the divide: Strategies for college to career readiness in computer science. *Proceedings - Frontiers in Education Conference, FIE*, 2014:1–8, 2015.

[8] Paul Luo Li, Andrew J. Ko, and Jiamin Zhu. What makes a great software engineer? *Proceedings - International Conference on Software Engineering*, 1:700–710, 2015.

[9] Joseph Maguire, Steve Draper, and Quintin Cutts. What Do We Do When We Teach Software Engineering? In *First UK and Ireland Computer Science Education Research Conference (UKICER '19).*, pages 1–7. ACM, 2019.

[10] J. Margolis, R. Estrella, J. Goode, and K. Nao. *Stuck in the shallow end: Education, race, and computing.* The MIT Press, Cambridge, MA, 2008.

[11] Christian Murphy, Swapneel Sheth, and Sydney Morton. A two-course sequence of real projects for real customers. *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, pages 417–422, 2017.

[12] Leo Porter, Daniel Zingaro, Soohyun Nam Liao, Cynthia Taylor, Kevin C. Webb, Cynthia Lee, and Michael Clancy. BDSI: A Validated Concept Inventory for Basic Data Structures. *Proceedings of the 2019 ACM Conference on International Computing Education Research - ICER '19*, pages 111–119, 2019.

[13] Alex Radermacher and Gursimran Walia. Gaps between industry expectations and the abilities of graduates. *SIGCSE 2013 - Proceedings of the 44th ACM Technical Symposium on Computer Science Education*, pages 525–530, 2013.

[14] Chris B. Simmons and Lakisha L. Simmons. Gaps in the Computer Science Curriculum: An Exploratory Study of Industry Professionals. *The Journal of Computing Sciences in Colleges*, 25(5):60–65, 2010.

[15] Claude M. Steele and Joshua Aronson. Stereotype threat and the intellectual test performance of African Americans. *Journal of Personality and Social Psychology*, 69(5):797–811, 1995.

[16] A. Nicki Washington, Legand Burge, Marlon Mejias, Ketly Jean-Pierre, and Qi'anne Knox. Improving Undergraduate Student Performance in Computer Science at Historically Black Colleges and Universities (HBCUs) through Industry Partnerships. In *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 203–206. ACM, 2015.

[17] Carl Wieman and Sarah Gilbert. The Teaching Practices Inventory: A New Tool for Characterizing College and University Teaching in Mathematics and Science. *CBE Life Sciences Education*, 13(3):552–569, 2014.

[18] Stuart Zweben and Betsy Bizot. 2018 Taulbee Survey. 2018.

Plagiarism Prevention through Project Based Learning with GitLab*

Giovanni Gonzalez Araujo, Angelo Kyrilov
Department of Computer Science and Engineering
University of California, Merced
Merced, CA 95343
{ggonzalezaraujo, akyrilov}@ucmerced.edu

Abstract

In this paper we investigate the extent to which Project Based Learning (PBL) contributes to plagiarism prevention. PBL refers to the process of delivering course material to students by asking them to work on projects that make use of the concepts that need to be covered. We accomplished this in a flipped classroom environment, with the GitLab system as a platform. In addition to restructuring the class activities, we also redesigned all our programming assignments, and went from simple, automatically graded exercises, to open-ended problems on real software projects that students can relate to. To evaluate the effectiveness of our proposed approach, we looked at plagiarism rates form prior course offerings, and found that more than 70-80% of students had committed plagiarism. In our current course, there have not been any plagiarism cases. We believe this can be attributed to the interventions we designed.

1 Introduction

Plagiarism is a widespread issue in undergraduate Computer Science Education. In the typical classroom, most assessment components are practical programming exercises, which makes it easy for students to plagiarize [4]. In addition to sharing code with one another, students can also obtain complete assignment solutions from many online platforms, such as Chegg, Geeks for

Geeks, and Stack Overflow. While such platforms are useful for providing support, some students misuse and abuse these systems by essentially outsourcing their assignments to other developers, learning very little, in the process.

The rates of plagiarism in our undergraduate Computer Science courses have been steadily increasing in recent years, especially in courses that have a large programming component. Despite our efforts to combat the situation by using plagiarism detection tools and introducing harsher punishments for offenders, plagiarism rates remain high, similar to the findings in [7]. A study by [12] suggests that coursework design is highly correlated with plagiarism rates. We therefore sought to redesign several aspects of one of our courses in an attempt to prevent plagiarism.

In this paper, we investigate whether the changes introduced in the course, namely the GitLab platform, the flipped classroom, and the Project Based Learning (PBL) approach, have contributed to plagiarism prevention, and thereby improved learning.

The rest of the paper is organized as follows. Section 2 contains background and related work. Section 3 outlines the experiment, and section 4 presents the results and discussions. Section 5 is a brief summary of the main findings.

2 Background and Related Work

Contrary to traditional classroom teaching methods, the flipped classroom approach consists of the use of technology to access learning materials outside of class, while engaging in active learning activities during class time. According to [3], this approach shows very promising results in higher education. [5] studied two offerings of a CS1 class held in two different semesters, with a total of 1307 students across both, where first offering had a traditional format, while the second one followed an inverted classroom approach. The authors found that in the second offering, students performed significantly better in the final.

Live coding demonstration, defined as "the process of designing and implementing a coding project in front of class during lecture period", is another form of active learning. [8] examines the effectiveness of live-coding demonstrations in introductory CS courses. The authors show that live coding demonstrations led to significant increases in student performance on projects. Furthermore, 90% of students in the study agreed that code examples were more educational than traditional lecture slides.

Computer Science is subject to significant plagiarism rates because of the nature of the assignments given [2]. In most cases, programming assignments have very specific instructions, leaving little room for creative input from students. Students see these assignments as having low educational value, leading to a lack of motivation to work on them.[10]. In [6] the authors found auto-

mated assessment systems that generate low-quality feedback, in the form of a binary (correct/incorrect) signal, can lead to cheating. This is because binary feedback offers no guidance on how to correct a problem, and with no other viable alternatives, students oftentimes resort to dishonest practices.

Instructors have been relying on plagiarism detection software, with MOSS, described in [1], being one of the most widely used. [9] gave MOSS the ability to detect plagiarism across multiple semesters. To detect cases of assignment outsourcing, [11] proposed a method for automatic plagiarism detection, by means of computing the differences between consecutive submissions made by a student, assuming they should not differ greatly from one to the next. The authors found their method to be 82% accurate.

Despite the popularity and high accuracy of plagiarism detection tools, they do little to prevent plagiarism. We have found that even harsher punishments for plagiarism are not effective in reducing plagiarism. This is a motivating factor for our efforts in redesigning the curriculum in order to prevent plagiarism, rather than detect it and punish the offenders.

3 Research Methodology

The first step was to determine plagiarism rates from past course offerings, taught in a traditional manner. We selected Spring 2018 (59 students), and Fall 2018 (120 students). We classified each student into one of the following categories: "Start Honest - Stay Honest", "Start Honest - Cheat", or "Cheat on First Attempt". Classification was done by comparing all submissions to each other for similarity. Cases of high similarity were examined manually. Submissions were also compared to solutions posted on Chegg, flagging all students who used them. The results are seen in Figure 1.

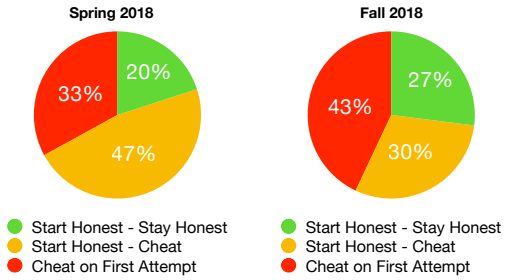


Figure 1: Spring 2018 and Fall 2018 student plagiarism classification

Overall plagiarism rates for Spring 2018 and Fall 2018 are 80% and 73% respectively, and about 25% of students who cheated downloaded complete

solutions from Chegg and turned them in unmodified. In an attempt to prevent this, we introduced the following changes to the course in the Fall 2019 offering.

We moved away from the traditional lecturing with slides to a flipped classroom environment with live coding demonstrations, and Project Based Learning (PBL). Each week, students were assigned a project of appropriate size and scope. In the first week of the course, students were introduced to RESTful API design, so all their projects were web applications. Lectures were organized as a series of 15-minute presentations by the instructor, followed by 20 minute coding sessions. As students developed their code, they pushed it to their GitLab repositories. Using the Continuous Integration and Continuous Deployment (CI/CD) features of GitLab, students' applications were automatically built, unit tested, and deployed to a live environment, accessible worldwide. When students were done with a particular project, they complete a Merge Request, which initiates a code review session between the student and the teaching assistant. As a result of this code review, some students found problems with their code, which they then went on to fix. Figure 2 shows the interface of a typical project students were assigned. In this case it is a game that generates a random set of letters and the user has to make valid English words. The game shown, still contains a bug that accepts any sequence of characters as a valid English word.

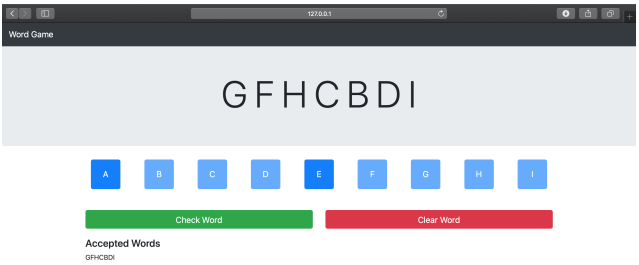


Figure 2: Game interface, with "bugs" still unresolved

Within a given week, students had two lecture periods like the one described above, and an optional laboratory session. It was optional because not all students needed it. Some students were able to finish their weekly project within the two lecture periods, while others needed more time and assistance. In addition to the laboratory time, students also had the opportunity to get help in office hours hosted by the instructor and the teaching assistant. This pattern repeated every week, with a new project being assigned, using it as a vehicle to deliver the course concepts that needed to be covered, as prescribed by a PBL philosophy.

4 Results and Discussion

In the Fall 2019 semester, when the interventions were implemented, there were no cases of plagiarism found. This does not mean that plagiarism was completely eradicated, but it is promising to see that no students were detected. In addition to this, none of the current lab assignments have been posted on Chegg, or any other similar forum, meaning that there was no outsourcing of assignments. This leads us to conclude that the changes we introduced in our course have been effective in reducing plagiarism, and therefore improved student learning.

Possible factors that caused this improvement include the fact that students are not working on command-line scripts, which they struggle to relate to. They are working on web applications that look and feel similar to the software they interact with every day, which could motivate students to put in more effort. It is possible that the structure of class activities played a part too. With a PBL approach, students were could move at their own pace, allowing more advanced students to complete the material quickly and spend their time on productive things, while students who needed help, had multiple opportunities to get it, especially in the lab, because they walk in with something that they have been working on during lectures, making the task easier to complete.

5 Conclusion

Plagiarism is a widespread problem in Computer Science Education, amplified further when lab assignments are poorly designed, reused between multiple course offerings, or automatically graded.

We redesigned one of our courses to follow the flipped classroom philosophy with a PBL framework. The results are very promising, as there have been no cases of plagiarism this semester, since the interventions were introduced. Reasons for this include the fact that students are more motivated to work on their solutions, because the problems they solve, and the platforms that they implement them on, are similar to software products they interact with on a daily basis. Another reason is that it is harder to outsource open-ended questions to websites like Chegg and Stack Overflow. If students attempt to get help on these types of questions online, they are more likely to receive guidance, rather than complete solutions, which is the way these online tools should be used. Further research is currently underway in order to more accurately determine the effects of each aspect of the interventions we introduced.

6 Acknowledgments

Many thanks to the anonymous reviewers for helping to improve the paper.

References

[1] A Aiken. *MOSS: A System for Detecting Software Plagiarism.* 2002.

[2] R. Fraser and D. Cheriton. Collaboration, Collusion and Plagiarism in Computer Science Coursework. *Informatics in Education*, 13, 09 2014.

[3] M. N. Giannakos, J. Krogstie, and N. Chrisochoides. Reviewing the Flipped Classroom Research: Reflections for Computer Science Education. In *Proceedings of CSERC*, 2014.

[4] J. K. Harris. Plagiarism in Computer Science Courses. In *Proceedings of ECA*, 1994.

[5] D. Horton and M. Craig. Drop, Fail, Pass, Continue: Persistence in CS1 and Beyond in Traditional and Inverted Delivery. In *Proceedings of SIGCSE*, 2015.

[6] A. Kyrilov and D. C. Noelle. Binary Instant Feedback on Programming Exercises Can Reduce Student Engagement and Promote Cheating. In *Proceedings of Koli Calling*, 2015.

[7] L. Nichols, K. Dewey, M. Emre, S. Chen, and B. Hardekopf. Syntax-based Improvements to Plagiarism Detectors and Their Evaluations. In *Proceedings of ITiCSE*, 2019.

[8] M. J. Rubin. The Effectiveness of Live-coding to Teach Introductory Programming. In *Proceeding of SIGCSE*, 2013.

[9] D. Sheahen and D. Joyner. TAPS: A MOSS Extension for Detecting Software Plagiarism at Scale. In *Proceedings of L@S*, 2016.

[10] J. Sheard, A. Carbone, and M. Dick. Determination of Factors Which Impact on IT Students' Propensity to Cheat. In *Proceedings of ACE*, 2003.

[11] N. Tahaei and D. C. Noelle. Automated Plagiarism Detection for Computer Programming Exercises Based on Patterns of Resubmission. In *Proceedings ICER*, 2018.

[12] P. Vamplew and J. Dermoudy. An Anti-plagiarism Editor for Software Development Courses. In *Proceedings of ACE*, 2005.