The Journal of Computing Sciences in Colleges

Papers of the 34th Annual CCSC Southeastern Conference

January 22nd-23rd, 2021
University of North Carolina Asheville
Asheville, NC

Baochuan Lu, Editor
Southwest Baptist University

John Hunt, Regional Editor
Covenant College

Volume 36, Number 5 **January 2021**

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:

Karina Assiter, President (2022), (802)387-7112, karinaassiter@landmark.edu.

Chris Healy, Vice President (2022), chris.healy@furman.edu, Computer Science Department, 3300 Poinsett Highway Greenville, SC 29613.

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.

Cathy Bareiss, Membership Secretary (2022), cathy.bareiss@betheluniversity.edu, Department of Mathematical Engineering Sciences, 1001 Bethel Circle, Mishawaka, IN 46545.

Judy Mullins, Central Plains Representative (2023), 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.

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

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

Grace Mirsky, Midwest Representative(2023), gmirsky@ben.edu, Mathematical and Computational Sciences, 5700 College Rd. Lisle, IL 60532.

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

Shereen Khoja, Northwestern Representative(2021), shereen@pacificu.edu, Computer Science, 2043 College Way, Forest Grove, OR 97116.

Mohamed Lotfy, Rocky Mountain Representative (2022), Information Systems & Technology Department, College of Engineering & Technology, Utah Valley University, Orem, UT 84058.

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.

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 (2023), (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:

Bin "Crystal" Peng, Associate Editor, (816) 584-6884, crystal.peng@park.edu, Park University - Department of Computer Science and Information Systems, 8700 NW River Park Drive, Parkville, MO 64152.

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

Welcome to the 2021 CCSC Southeastern Conference

Welcome to the 34th Southeastern Regional Conference of the Consortium for Computing Sciences in Colleges. The CCSC:SE Regional Board welcomes you –virtually, at least! –to Asheville, NC, the home of UNC-Asheville. The conference is designed to promote a productive exchange of information among college personnel concerned with computer science education in the academic environment. It is intended for faculty as well as administrators of academic computing facilities, and it is also intended to be welcoming to student participants in a variety of special activities. We hope that you will find something to challenge and engage you at the conference!

The conference program is highlighted with a variety of sessions, such as engaging guest speakers, workshops, panels, student posters, faculty posters, a nifty assignment session and several sessions for high quality refereed papers. We received 38 papers this year of which 21 were accepted to be presented at the conference and included in the proceedings –an acceptance rate of 55%.

Two exciting activities are designed specifically for students –a research contest and an undergraduate programming competition, with prizes for the top finishers in each.

We especially would like to thank the faculty, staff, and students of UNC-Asheville for their help in organizing this conference, especially under the challenging circumstances caused by the pandemic. Many thanks also to the CCSC Board, the CCSC:SE Regional Board, and to a wonderful Conference Committee, led by Conference Chair Dr. Marietta Cameron. Thank you all so much for your time and energy.

We also need to send our deepest appreciation to our partners, sponsors, and vendors. Please take the time to go up to them and thank them for their contributions and support for computing sciences education – CCSC National Partners: Turing's Craft, Google for Education, GitHub, National Science Foundation, Codio, zyBooks, National Center for Women and Information Technology, Teradata University Network, Mercury Learning and Information, Mercy College. Sponsoring Organizations: CCSC, ACM-SIGCSE, Upsilon Pi Epsilon.

We could not have done this without many excellent submissions from authors, many insightful comments from reviewers, and the support from our editor Baochuan Lu. Thanks to all of you for helping to create such a great program.

We hope you enjoy the conference and your virtual visit to UNC-Asheville.

Kevin Treu, CCSC:SE Regional Board Chair
Furman University
John Hunt, Program Chair
Covenant College

2021 CCSC Southeastern Conference Steering Committee

Marietta Cameron ..... Local Arrangements Chair
Brian Drawert ..... Local Publicity Chair
Marietta Cameron ..... Speakers Chair
Brian Drawert ..... Vendors Chair
Marietta Cameron ..... Local Sponsors Chair
Andy Digh ..... Programming Contest Co-Director
Chris Healy ..... Programming Contest Co-Director
Chris Healy ..... Student Research Contest Director
Nadeem Abdul Hamid ..... Nifty Assignments Co-Chair
Steven Benzel ..... Nifty Assignments Co-Chair

Regional Board — 2021 CCSC Southeastern Region

Marietta Cameron ..... 2020 Site Chair
Kevin Treu ..... Regional Board Chair
Kevin Treu ..... CCSC:SE Regional Representative
John Hunt ..... Treasurer
John Hunt ..... Program Co-Chair
Stephen Carl ..... Publicity Chair
Jean French ..... Local Registrar
Richard Chapman ..... 2019 Site Chair

Reviewers — 2021 CCSC Southeastern Conference

Ahmed, Syed ..... York Technical College
Ali, Farha ..... Lander University
Alvin, Chris ..... Furman University
Banik, Shankar ..... Citadel
Bell, Chip ..... Mercer Engineering Research Center
Bennett, Brian ..... East Tennessee State University
Besmer, Andrew ..... Winthrop University
Bonyadi, Cyrus ..... University of Maryland, Baltimore College
Bowe, Lonnie ..... Concord University
Chakraborty, Sujoy ..... Stockton University
Christ, Beau ..... Wofford College
Dannelly, Stephen ..... Winthrop University
Dogan, Gulustan ..... University of North Carolina Wilmington
Dumas, Joe ..... University of Tennessee at Chattanooga
Fain, Brandon ..... Duke University
Galanti, Mattia ..... Lander University
Galanti, Mattia ..... Lander University
Gaspar, Alessio ..... University of South Florida
Gesick, Richard ..... Southern Polytechnic State University
Glass, Michael ..... Valparaiso University
Goddard, Wayne ..... Clemson University
Gunay, Cengiz ..... Georgia Gwinnett College
Hamid, Nadeem ..... Berry College
Healy, Chris ..... Furman University
Heinz, Adrian ..... Georgia Gwinnett College
Holliday, Mark ..... Western Carolina University
Hong, Gongbing ..... Georgia College and State University
Kane, Ilish ..... Athens state University
Knisely, Jim ..... Bob Jones University
Lee Gilliean ..... Lander University
Lee, Ingyu ..... Troy University
Li, Rao ..... University of South Carolina Aiken
Lindoo, Ed ..... Nova Southeastern University
Liu, Qian ..... Rhode Island College
Liu, Yi ..... Georgia College State University
McDaniel, Melinda ..... Georgia Institute of Technology
McGee, Ethan ..... Bob Jones University
North, Sarah ..... Kennesaw State University

Patterson, Brian ..... Oglethorpe University
Payne, Bryson ..... University of North Georgia
Pearson, Bryan ..... University of Central Florida
Perez Quinones, Manuel ..... UNC Charlotte
Phelps, Gita ..... Georgia College State University
Plank, James ..... University of Tennessee
Pounds, Andrew ..... Mercer University
Pournaghshband, Hassan ..... Kennesaw State University
Rao, M. Padmaja ..... Francis Marion University
Redder, Daniel ..... Georgia Gwinnett College
Robertson, Cindy ..... Georgia Gwinnett College
Rutherfoord, Rebecca ..... Southern Polytechnic State University
Sanders, Ian ..... University of South Africa
Sanft, Kevin ..... University of North Carolina Asheville
Shim, Leem ..... University of Mount Olive
Spoehel, Elizabeth ..... The Citadel
Spurlock, Scott ..... Elon University
Sriskanda, Nesan ..... Claflin University
Sudarshan, Sanjana ..... Jetstream at Indiana University
Terwilliger, Mark ..... University of North Alabama
Vines, Paul ..... BAE Systems/FAST Labs
West, Paul ..... Charleston Southern University
Wijesundara, Isuru ..... Methodist University
Works, Karen ..... Florida State University
Xie, Mengjun ..... The University of Tennessee at Chattanooga
Yarn, Taylor ..... Roanoke College
Zhong, Bill ..... Troy University

Improving Machine Learning Fairness with Sampling and Adversarial Learning*

Jack J Amend and Scott Spurlock
Computer Science
Elon University
Elon, NC 27302
{jamend,sspurlock}@elon.edu

Abstract

Machine learning approaches learn models based on the statistical properties of training data. Learned models may be unfair due to bias inherent in the training data or because of spurious correlations based on sensitive attributes such as race or sex. This type of bias can lead to detrimental outcomes in important applications, including prison sentencing, credit scoring, and loan approvals. In this work, we perform a comparative study of techniques to increase the fairness of machine learning based classification with respect to a sensitive attribute. We assess the effectiveness of several data sampling strategies as well as of a variety of neural network architectures, including conventional and adversarial networks. Results are evaluated in terms of metrics measuring both classification accuracy and fairness. We find that model architecture and sampling strategy can both greatly affect metrics of fairness. We also find that there is no single best combination that should be used; the particular problem domain should drive the selection of neural network architecture and sampling strategy.

1 Introduction

Machine learning is becoming increasingly common in everyday life. Models are used to select ads to show users, recommend movies, and predict patient

outcomes. Advances in computing power, machine learning algorithms, and availability of training data have enabled the creation of models that can exhibit high accuracy, but are typically difficult to audit because of their complexity. Recently, many such models have been found to perpetuate societal biases. For example, a study by ProPublica found that a system that predicted recidivism scores was racially biased, predicting a higher likelihood of recidivism for Black individuals even when other factors were similar [1]. The frequency and disproportionate impact of this type of systemic bias motivate the necessity of finding ways to measure and mitigate bias in machine learning models.

In this paper, our focus is on *classification*, the most common application of machine learning, in which, given a data set of training examples and corresponding desired (ground truth) labels, a training process learns a model that can accurately predict the correct labels for given data examples. The training process seeks statistical patterns in the data that may be difficult or impossible for humans to identify. The resulting model is typically optimized to yield high levels of accuracy. We focus on artificial neural networks as the learning method most commonly employed in recent machine learning research and compare differing network architectures, particularly a vanilla "basic" network, and an adversarial network that optimizes competing objectives during training.

The training process is vulnerable to learning spurious correlations between attributes, particularly when the amount of data is limited. Sometimes these spurious correlations are harmless, e.g., learning that thin people always wear hats [9]. Learning is further vulnerable to codifying bias already present in the training data. These factors can result in models that are potentially detrimental, e.g., the association of Black individuals with higher rates of recidivism. Such factors as race and sex are of particular relevance to bias in models, and are often described as *protected* (or *sensitive*) attributes. Prior research has sought to quantify bias using several different criteria. Below we give definitions for two commonly used metrics. Following the notation of Zhang et al. [11], we use X, Y, and Z to indicate the input data, true label, and protected attribute, respectively. The model prediction for a given example is given by $\hat{Y} = f(X)$, where the function f is the learned model represented by a trained neural network. We indicate a particular value of the output variable, Y, by y, and of the protected variable, Z, by z.

- **Demographic Parity** measures that the predicted outcome is independent of the value of a protected attribute. $P(\hat{Y} = \hat{y}) = P(\hat{Y} = \hat{y}|Z = z)$

- **Equality of Opportunity** measures that the predicted outcome is conditionally independent of the value of a protected attribute for *one par-*

ticular value of the outcome, which we indicate as 1.
$P(\hat{Y} = \hat{y}|Y = 1) = P(\hat{Y} = \hat{y}|Z = z, Y = 1)$

Our research evaluates strategies to reduce bias in learned models due to spurious correlations and biased training data that may have an adverse impact based on protected attributes. We conduct several experiments using the UCI Adult Data set [3] with the goal of predicting whether individuals belong to the high or low income group. We evaluate prediction accuracy with respect to the target variable and several fairness metrics with respect to the sex (male or female) attribute. Our experiments vary data sampling strategies as well as neural network architectures with the goal of addressing four research questions:

- **RQ1** Can a basic neural network achieve boosts in fairness metrics?
  In particular, we investigate whether basic network architectures can be effective when paired with an appropriate data sampling strategy.

- **RQ2** Which network architecture outputs the least biased predictions?
  We compare the results of training models with simple architectures as well as more complex, recently developed adversarial architectures.

- **RQ3** What is the best data sampling strategy to increase fairness?
  We compare several different strategies, including resampling to ensure the number of training examples is balanced over possible values of the sensitive attribute, of the desired label, and of both.

- **RQ4** Can we combine good architecture and data sampling to achieve better results?
  We evaluate the results of pairwise combinations of several sampling strategies and network architectures.

In the next section, we review recent work in the area of machine learning fairness. In Section 3, we describe our methodology, including data sampling strategies and neural network architecture choices, followed by a review of our experiments and results in Section 4. We conclude in Section 5 and offer some thoughts on potential directions for future work.

2 Related Work

Fairness in machine learning is becoming an active area of research. A recent survey focuses on the role that unbalanced training data can play in contributing to this issue, and groups work in the area into approaches that focus

on pre-processing the data and approaches that address the issue algorithmically [7].

Wang et al. evaluates several bias-reduction techniques in a computer vision context [10]. The authors propose training an ensemble of domain-independent classifiers (i.e., one classifier per possible value of a protected attribute). Interestingly, while this approach often outperforms a variety of alternatives, in some experiments, sampling with replacement to manually balance a training data set performs better. Oversampling techniques appear to be particularly effective when the data set is large (mitigating overfitting) and the amount of bias inherent in the data is smaller.

2.1 Adversarial Networks

A particular emphasis in much of the recent work has been on using specialized neural network architectures to help reduce bias in learned models. Adversarial learning seeks to optimize multiple neural networks with competing objectives. Typically, one network optimizes classification accuracy for a given model, while another network optimizes the ability to guess the value of a protected attribute given the classifier's output. By alternately training both networks, the goal is to learn a model that can predict with high accuracy while also exhibiting low levels of bias.

Several recent approaches [2, 4, 8, 11] propose to learn a mapping from input data to a new representation that removes bias from the source data. This learned representation is then suitable for learning unbiased classification models. The approach leverages an adversarial network seeking to predict a protected attribute based on the representation. Some work also finds that having balanced data sets in terms of the distribution of examples over the protected attribute is helpful in producing a fair model and that an adversarial approach allows for smaller numbers of training examples [2].

2.2 Fairness Metrics

There are many different metrics to measure fairness in a learned model, with new metrics being regularly proposed in the literature. Unfortunately, there is no consensus as to a single best approach to quantifying fairness, and there is generally a trade-off between model accuracy and various different fairness metrics. One recent study, which conducted a survey of the human perception of fairness of competing models in a hypothetical scenario, found that, while participants showed a slight preference for equalizing fairness over accuracy, they disagreed on how to measure it [6].

One measure to quantify fairness is equality of opportunity, introduced in recent work to remove bias from learned models [5]. Other common metrics include demographic parity and equality of odds [11].

3 Methodology

In this section, we review our study's neural network architectures and data sampling strategies, as well as the data set used for evaluation.

3.1 Architecture

We experiment with four network architectures:

The **Basic** model is implemented as a simple fully connected 4-layer neural network that takes in the data, X, and outputs the predicted label $\hat{Y}$. This model serves as a baseline for comparison to the others.

The **Split** approach trains a separate Basic model for each of the possible protected attribute values, allowing each network to model a separate distribution. For a protected attribute like sex with two possible values, we train two independent basic models. At test time, each example is classified by the appropriate model.

The **CAN** (Classifier-Adversarial Network) architecture follows an adversarial learning approach, similar to several recent methods [2, 4, 11]. Adversarial learning works by pitting two competing neural networks against each other. The first, f, is the classifier, based on the Basic architecture described above, which attempts to predict the label, Y. The second network, g, uses the output from the classifier, $\hat{Y}$, to predict the protected attribute, Z. Training proceeds iteratively, alternately optimizing each network. After training, the networks reach an equilibrium, with the goal that the classifier performs with a high level of accuracy and the adversary performs poorly, near the level of random guessing in its ability to predict the protected attribute, thus limiting the correlation between the output of the classifier and the sensitive attribute.

The **CANE** model (CAN with Embedding), similar to CAN, trains competing classification and adversary networks. However, for CANE, the input to the adversary is augmented to include, in addition to $\hat{Y}$, the prediction from the classifier, the features from the penultimate layer of the classifier network. These features constitute an embedded, or lower-dimensional, representation of each input, X. They provide the adversary with more information, with the goal of helping to learn a less biased model. This variant of the CAN approach is actually more common in recent literature [2, 4, 11].

3.2 Data Sampling

Several recent approaches have focused on the impact of data sampling on fairness [5, 10]. To see how data affects the overall fairness of the model, we compare 4 sampling approaches. No Sampling, **NS**, uses the data without modification, serving as a baseline. Sensitive Sampling, **SS**, resamples the

Income	Female	Male
<=50K	9,592	15,128
>50K	1,179	6,662

Table 1: Counts of observations across income and sex in the highly unbalanced UCI Adult data set. Of 32,561 examples, there are many more low-income male observations (46.5%), while only 1,179 (3.6%) are high-income females.

training data so that the number of examples from each possible value of the sensitive attribute is the same (e.g., the same number of men and women). Label Sampling, **LS**, resamples the training data in a similar fashion with respect to the target variable, while Sensitive Label Sampling, **SLS**, equalizes the number of examples across each combination of sensitive attribute and target variable value.

3.3 Data

For our study, we selected the UCI Adult data set [3], which contains 14 continuous and categorical features including age, education, race, sex, and marital status, as well as an associated target variable indicating whether or not each individual's income is above or below $50K. This data set is well suited to our experiments because it is unbalanced in terms of the number of examples across both the sensitive attribute of sex as well as the target label. It has been shown to contain bias based on sex, and has been used in a variety of recent work on bias mitigation [2, 8]. As Table 1 shows, counts of observations across sex and income in the UCI Adult Data set are heavily skewed. Nearly two thirds of the observations are male and nearly three quarters of the observations are low-income. These disparities become even more apparent when looking at the counts for each combination of sex and income. Observations falling into both the high-income and female bins make up less than 4% of the entire data set.

4 Results and Discussion

In this section, we present our results and discuss findings for each of the four research questions.

Implementation The neural networks were implemented in Python using TensorFlow. For each architecture and sampling combination, models were trained for 100 epochs using the ADAM optimizer and a learning rate of 2e-4.

Model	Sampling	Acc. Overall	Acc. Female	Acc. Male	Parity Gap	Equality Gap -	Equality Gap +
Basic	NS	0.8479	0.9214	0.8116	0.1870	0.0808	0.1102
Basic	SS	0.8759	0.9399	<u>0.8119</u>	0.1732	0.0841	0.0446
Basic	LS	0.8744	0.9391	0.8097	0.1780	0.0880	0.0245
Basic	SLS	**0.8766**	0.9416	0.8116	0.1866	0.0936	<u>0.0069</u>
CAN	NS	0.8481	0.9192	**0.8129**	0.1613	0.0632	0.0257
CAN	SS	0.8742	0.9399	0.8085	0.1610	0.0776	0.0670
CAN	LS	<u>0.8764</u>	0.9413	0.8115	0.1631	0.0770	0.0513
CAN	SLS	0.8737	0.9402	0.8073	0.1458	0.0673	0.0899
CANE	NS	0.8444	0.9208	0.8066	<u>0.1375</u>	**0.0496**	**0.0030**
CANE	SS	0.8752	0.9394	0.8110	0.1647	0.0764	0.0218
CANE	LS	0.8732	0.9386	0.8078	0.1673	0.0820	0.0545
CANE	SLS	0.8582	0.9301	0.7862	**0.1301**	<u>0.0618</u>	0.0365
Split	NS	0.8428	0.9133	0.8080	0.1737	0.0712	0.0845
Split	SS	0.8532	**0.9478**	0.8064	0.1695	0.0896	0.0919
Split	LS	0.8539	<u>0.9468</u>	0.8080	0.1684	0.0876	0.0946
Split	SLS	0.8529	0.9447	0.8075	0.1748	0.0916	0.0816

Table 2: Experimental results with best value for each column bolded, second best underlined. For accuracies, higher is better; for gap metrics, lower is better. There tends to be a trade-off between better accuracy and gap metrics.

Metrics Table 2 lists the results for experiments with each of the models and sampling strategies. Results are averaged across multiple trials using 5-fold cross validation. For fair comparison, the same training-validation splits are used for each variant. Metrics include classification accuracy (overall and broken out for male and female) as well as fairness [2], based on the concepts of demographic parity and equality of opportunity defined in Section 1. The parity gap is calculated as the difference between probabilities of the model predicting high-income for the two sexes. The equality gap is the difference in probability of predicting each class, given the sex. This metric can be calculated for each of the target values, i.e., one for low and another for high income (Equality Gap - and +, respectively).

Question 1: Can a basic neural network achieve boosts in fairness metrics? Our results show that, for the basic network architecture, compared with not sampling, the other sampling strategies improve accuracy (SLS sampling yielded the highest overall accuracy of 87.66% across all experiments), but do not greatly improve fairness, with the exception of the high-income equality gap, which shows some of the lowest scores across all tests. Interestingly, the accuracy increase for SLS sampling comes primarily from more accurate classification of female examples, suggesting that this strategy improves the model's

ability to generalize for this underrepresented set. This outcome is supported by a closer look at classification error. Figure 1 breaks out errors in terms of false positive rate (FP), when the model incorrectly predicts high-income, and false negative rate (FN), when the model incorrectly predicts low-income. FP and FN are shown separately for overall, female, and male examples. For female examples, switching from no sampling to SLS causes the false positive rate to drop from 0.052 to 0.038, and the false negative rate from 0.027 to 0.021, while for male examples the error rate increases.

Question 2: Which network architecture outputs the least biased predictions? Compared to the basic model, all other model types result in some improvement in parity gap. This pattern was somewhat visible for the low-income equality gap, and less so for the positive equality gap. Overall, the adversarial architectures (CAN and CANE) produce models with better fairness metrics, and the CANE architecture unquestionably shows the best improvements to fairness metrics compared with the basic model. With no sampling, CANE results in the lowest equality gap (0.0496 and 0.0030) and second lowest parity gap (0.1375) across all experiments. The split model generally results in fewer improvements to fairness metrics, although combined with sampling strategies, does result in the highest accuracies for female examples in particular. Additionally, for false negative rate (Figure 1), we find a 17.47% decrease when using CANE with SLS vs. the CAN model with SLS. Compared to the basic model with SLS, false negative rate decreases by 27.43%.

Question 3: What is the best data sampling strategy to increase fairness? No single data sampling strategy improves all metrics across the board. For the adversarial models (CAN and CANE), no sampling (NS) generally results in the best parity and equality metrics, followed by SLS. For the basic and split models, parity gap is lowest for SS and LS sampling, with no clear-cut pattern for equality gap metrics. Figure 1 shows the impact of the type of sampling and model architecture on classification error rates. Most of the variability is due to female examples, with female false positive and negative rates exhibiting greater changes due to architecture and sampling choices. We theorize that this means that the models learn the distribution of females at varying levels based on the way the data is supplied and the model chosen. It is interesting to note that the false negative rates for CAN and CANE are similar for each of the resampling methods.

Question 4: Can we combine good architecture and data sampling to achieve better results? We find that overall, sampling strategies have more positive impact on the basic architecture. The adversarial architectures

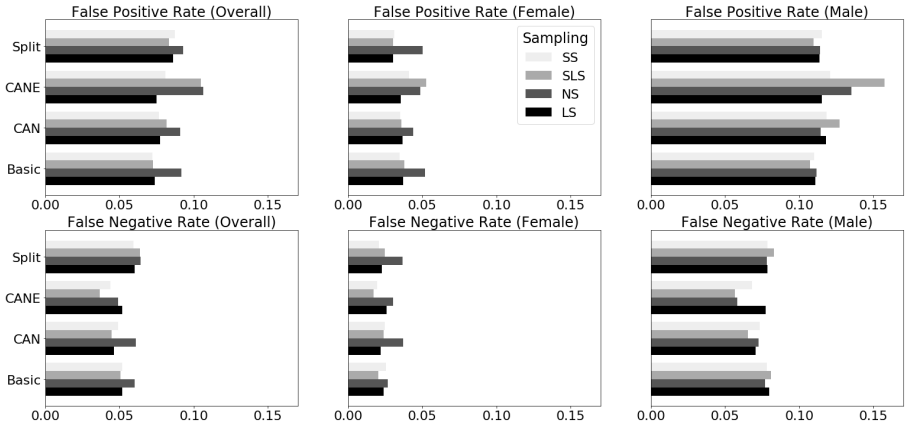


Figure 1: False positive (top) and false negative (bottom) rates for each sampling method grouped by architecture. The columns show (left to right) overall, female, and male rates, respectively.

perform better from a fairness perspective with no sampling, although sampling can lead to accuracy improvements. In general, the results suggest a trade-off between classification accuracy and fairness, with improvements in one coming at the cost of reduction in the other. While there is no one clearly best combination of architecture and sampling, CANE with SLS provides the best scores on the fairness metrics. For a good compromise between accuracy and fairness, we note that CAN with LS scored second-highest in overall accuracy while achieving fairness scores near the median of all experiments.

5 Conclusion

In this paper, we evaluate the impact of data sampling and neural network architecture on classification accuracy and fairness metrics with a series of experiments on the UCI Adult data set. We find that sampling and architecture can both have important effects on classification results, but that no single combination of approaches yields top scores across all measures. Instead, there is a trade-off that allows an approach to be tuned to a particular domain where one metric may be more important than another. For example, a low false negative rate might be vital for medical diagnosis, while for credit scoring, a provably low bias might be required by law. For the future, further work investigating explicitly incorporating fairness metrics into neural network training may provide valuable improvements to learned models. We are also interested in learning generative models of data distributions to support data augmentation of under-represented examples.

References

[1] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias, 2016.

[2] Alex Beutel, Ed H. Chi, Jilin Chen, and Zhe Zhao. Data decisions and theoretical implications when adversarially learning fair representations. In *Workshop on Fairness, Accountability, and Transparency in Machine Learning*, 2017.

[3] CL Blake and CJ Mertz. Uci repository of machine learning database, irvine, ca: University of california, 1998.

[4] Harrison Edwards and Amos Storkey. Censoring representations with an adversary. In *International Conference on Learning Representations*, 2016.

[5] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, 2016.

[6] Galen Harrison, Julia Hanson, Christine Jacinto, Julio Ramirez, and Blase Ur. An empirical study on the perceived fairness of realistic, imperfect machine learning models. In *Conference on Fairness, Accountability, and Transparency*, 2020.

[7] Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

[8] David Madras, Elliot Creager, Toniann Pitassi, and Richard Zemel. Learning adversarially fair and transferable representations. In *International Conference on Machine Learning*, 2018.

[9] Jamie Shotton, Ross Girshick, Andrew Fitzgibbon, Toby Sharp, Mat Cook, Mark Finocchio, Richard Moore, Pushmeet Kohli, Antonio Criminisi, Alex Kipman, et al. Efficient human pose estimation from single depth images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2821–2840, 2013.

[10] Zeyu Wang, Klint Qinami, Ioannis Christos Karakozis, Kyle Genova, Prem Nair, Kenji Hata, and Olga Russakovsky. Towards fairness in visual recognition: Effective strategies for bias mitigation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[11] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *AAAI Conference on AI, Ethics, and Society*, 2018.

AlgoScrum: Improving an Algorithms Course with Scrumage*

Scott Spurlock and Shannon Duvall
Computer Science
Elon University
Elon, NC 27302
{sspurlock,sduvall2}@elon.edu

Abstract

We present the design of an Algorithms Analysis course, based on the recently developed Scrumage approach, which allows students to work in cohorts if they choose and also allows each student to choose for themselves how they spend class time. We describe the course structure as well as the results from a survey to assess learning attitude outcomes. We show that the Scrumage method resulted in students taking more responsibility for their own learning and having improved impressions of the course and the course material.

1 Introduction

Algorithm Analysis is at the core of computer science [13]. At our institution, the anecdotal reputation of the course is somewhat negative, with end-of-term surveys from prior semesters surfacing student comments such as "material is miserable," "so much information," and "very confusing." Our observations are that students frequently struggle to engage with the material and, worse, often display a lack of understanding of how to improve, as well as a degree of passivity in their own learning.

In order to address these issues, we re-worked the course to follow the recently developed Scrumage pedagogical approach [6]. In Scrumage (SCRUM for AGile Education), course content is divided into short (2-3 week) units

called *sprints*, and students work in self-organizing teams to complete requirements (e.g., problem sets and programming assignments). A key component of Scrumage is that students have the freedom to choose multiple different learning approaches. Thus, one student may choose a traditional in-class lecture, while another may prefer a flipped model, watching videos outside of class instead. Students are provided with a variety of resources each sprint (textbook readings, slides, videos, sample problems, etc.) to allow them to dynamically choose how to learn the material.

We were interested to apply the Scrumage model to our Algorithms course with an eye toward helping students take more responsibility for their own learning and allowing them to have more choice in how they learn. With this idea in mind, we redesigned the course to include five sprints, each with its own set of assignments and a quiz at the end. Students had the freedom to choose a team (or to work individually) each sprint and to decide how to use the provided resources as they saw fit.

We have taught our Algorithms course using Scrumage for the past three offerings, observing each time improving student attitudes and evidence of metalearning, as students discover how they learn best. As instructors, we have found the experience of adopting Scrumage to have made the teaching process more enjoyable as students have transitioned psychologically from a model where a teacher pushes the material at them, to a model where they are empowered to make decisions about their own learning experience.

2 Related Work

As might be expected given the central importance of the topic, a variety of work has been published over the years related to teaching Algorithms to undergraduate students, including various pedagogical alterations to improve student learning. For example, one recent paper describes modifying a traditional course to focus on group-based problem solving [3], while another adapts a course to use an interactive e-book [9]. Including peer assessment in one algorithms course helped enhance students' critical thinking skills [5]. Other work looks at incorporating card games [10], puzzles [12], or programming competitions [8] to increase student interest.

The variety of approaches in the literature suggests that there is no single best pedagogy for teaching Algorithms. It seems clear from the scholarship of teaching and learning more broadly that active learning approaches are effective [11], but there appears to be no consensus on the effectiveness of particular techniques such as lecture [4], gamification [1], or a flipped model [2]. This observation informs the development of Scrumage, which aims to allow for all of these approaches to be available simultaneously to students within a single

classroom [6]. It is inspired by Scrum, a widely adopted project management technique that emphasizes lean processes (no busy-work), autonomous teams (no dictated decisions from management), and a fast feedback loop on both process and product (no static plans) [16]. Scrumage has previously been applied to a Discrete Mathematics course with a resulting improvement in student attitudes [7], but no work exists on using Scrumage for a higher-level course like Algorithm Analysis. In the following section, we describe the design of a Scrumage-based Algorithms course as well as a survey constructed to assess how student attitudes about learning changed from the start to the end of the semester.

3 Methodology

We have taught Algorithm Analysis at our institution many times in the past following a traditional, lecture-based approach; we have used the Scrumage approach in each of the last three offerings, observing each time similar improvements compared to our prior traditional approach. The authors have individually implemented Scrumage with slight differences in content, assignments, and incentives, but with the same core principles. In the following sections, we describe the most recent offering.

3.1 Course Structure

In the Scrumage approach, a course is divided into a series of units called sprints, each with its own set of topics and requirements. Each sprint begins with team assignment (after the first sprint, students have primary input into team formation) and the distribution of available resources (videos, readings, slides, etc.) and requirements (work to be completed by the end of the sprint). We provide a form for teams to make requests for how class time should be spent each day of the sprint. For example, a student might request the instructor to work through example problems similar to one of the requirements on a given day or play a review game prior to a quiz. Students are required to come to class each day to meet with their team at the start of class and to complete a check-in problem - a short (5-minute), lightly graded quiz designed to help students better understand how well they are progressing. After these required activities, students have the liberty to stay for the remainder of class or not, depending on whether the planned activities are helpful to them or not. On many days there is an optional lecture, followed by in-class work time when students can make progress on their requirements while the instructor circulates providing help as requested. The last day of each sprint culminates in a quiz.

Sprint	Topics	Requirements
1. Fundamentals I	Basics	PS1 - basic analysis
(3 weeks)	Analysis	PA1 - unique elements
	Sorting	Quiz 1
	Divide & conquer	
	Master method	
2. Fundamentals II	Recurrence relations	PS2 - more analysis
(3 weeks)	Quicksort	PA2 - k largest
	Lower bounds	Quiz 2
	Linear sorting	
3. Data Structures	Data structures	PS3 - heaps and trees
(3 weeks)	Binary search trees	Quiz 3
	Hash tables	
4. Graphs	Graphs, DFS, BFS	PS4 - graphs and greedy
(3 weeks)	Dijkstra's Algorithm	PA3 - fastest route
	MST	Quiz 4
	Huffman trees	
5. Adv. Techniques	Dynamic programming	PS5 - DP & NPC
(2 weeks)	NP-Completeness	Final exam

Table 1: The course is divided into 5 sprints, each focused on a subset of topics and with defined requirements to be completed. The requirements are either team-based problem sets (PS) or individual programming assignments (PA).

Table 1 shows a breakout of the topics and assignments for each of the five sprints. In particular, there are problem sets covering the more theoretical elements of algorithm design and analysis (15% of the total points), as well as more practical programming assignments (20% of the total points). The first four sprints end with a quiz (32% of the total points). Material from the fifth sprint is included on the (cumulative) final exam (23% of the total points). A small number of points (5%) is awarded for learning management activities, such as attending team meetings, making and following through on plans, and completing a retrospective survey at the end of each sprint. Finally, the daily feedback check-ins comprise a small number of points (5%) so students complete them thoughtfully.

3.2 Survey Development

To help identify changes in student attitudes over the course of the semester, we created a survey to be administered at the start and end of the course. The survey included 30 Likert-scale (1 - 7) questions. Of these, six related to student

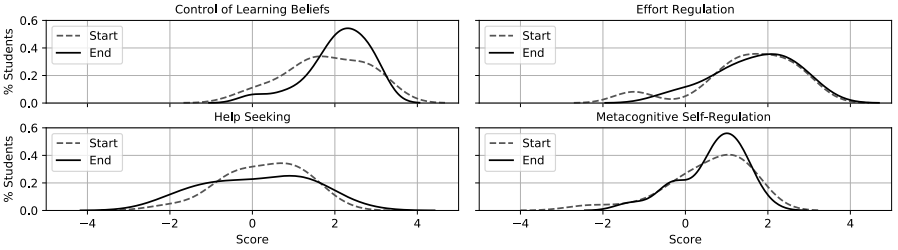


Figure 1: Kernel Density Estimate (KDE) distributions of average scores across categories from the learning attitudes survey questions show how student responses changed from the start to the end of the semester.

learning preference, such as "Small group discussion is an effective approach for me when I am learning new material." Also included were 24 questions focusing on student learning attitudes taken from the "Motivated Strategies for Learning Questionnaire" [15] from 4 categories: Effort Regulation, Metacognitive Self-Regulation, Help Seeking, and Control of Learning Beliefs. The surveys also included several free-text fields, such as "What are your impressions of the topic of Algorithm Analysis?" In the following section, we describe the survey results and our observations of student learning and attitudes.

4 Results and Discussion

The most recent course offering included 31 students across two sections. The majority of students were in their 3rd or 4th year in the program, with a few students in their 2nd year. We administered the survey the week prior to the start of the course, which established a baseline for our analysis, as well as providing input to initial team formation, where students providing similar survey responses were grouped together.

Of the 4 categories, the largest change in average student score was in the Control of Learning Beliefs category, which focuses on the extent to which students feel that they are able to learn the course material and are responsible for their own learning outcomes. While the sample size is small, this change was found to be statistically significant ($p = 0.05$) using the Wilcoxon signed rank test. This outcome suggests that we met one of our key goals: to help students take responsibility for their own learning. Figure 1 shows the distribution of student responses on each category of student attitude questions using kernel density estimation (KDE) [14] for the pre- and post-surveys.

In particular, the student attitudes questions with the largest absolute change over the semester were, "It is my own fault if I don't learn the ma-

terial in this course," which is part of the Control of Learning Beliefs category, and "Even if I have trouble learning the material in this class, I try to do the work on my own, without help from anyone," which is part of the Help Seeking category. We observed an average decrease in the Help Seeking category score, which we attribute to students being more inclined to make use of the provided resources rather than asking the instructor for help. While this outcome could indicate more independence in learning, we did not observe a decrease in student questions in class or by email, and attendance in office hours remained strong throughout the course. Other responses that showed significant increases related to learning preferences for working in small groups and for "jumping straight into problem-solving and looking up relevant information along the way" as opposed to a more structured introduction to material.

4.1 Text responses

In addition to Likert-scale questions, students responded to several free-text prompts. The following sections focus on changes in student impression of the topic, their feelings about ownership of their own learning, and their metalearning about their own individual approach to the course.

4.1.1 Student Impressions

One survey question related to student perceptions of the topic of algorithm analysis. Running the Vader sentiment intensity analyzer shows an improvement in student sentiment from pre- to post-survey. In particular, the compound sentiment score, which varies from -1 for extremely negative to +1 for extremely positive, increased from 0.1254 to 0.4543, over the semester. For additional insight, we coded the responses based on keywords related to four categories: difficult, interesting, useful, and enjoyable. The following example comments are typical for each category:

- Difficult: "Algorithm analysis was very difficult... but I learned a lot from the class. "
- Interesting: "Tough but interesting. I like the more academic lens of computer science that we have in this class."
- Useful: "It seemed to be very important and mostly a way of thinking. It helped me think of new ways to solve problems and approach programming from a different perspective."
- Enjoyable: "I enjoyed it. I liked going beyond writing functional code and evaluating what made code efficient/fast/generally 'good'."

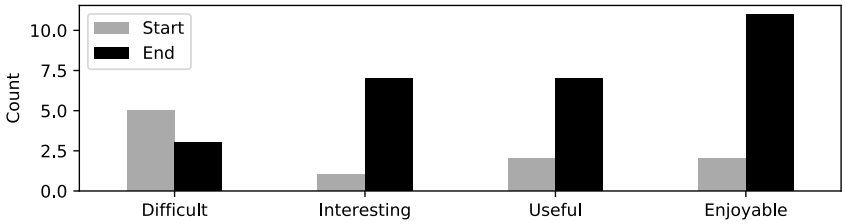


Figure 2: Counts of sentiment tags relating to student impressions of the topic of algorithms on a pre- and post-test.

Figure 2 shows how these counts changed over the course of the semester, with students indicating increased positive impressions and decreased negative impressions. (Note that a response could be tagged with multiple categories.)

4.1.2 Student Ownership of Learning

One interesting observation is that, while in-class lectures were explicitly labeled as optional, with students invited to leave if they preferred a different learning approach, in practice, almost all students stayed every day for lecture. However, they noted in surveys that they appreciated the freedom to choose, e.g., "every day I made the choice to stay for the lectures" and "···the optional lectures helped because I always stayed to advance my learning." We hypothesize that the psychological impact of "opting in" is a key part of the success of the Scrumage approach. One survey question related specifically to students' feelings about responsibility for their own learning. Multiple responses suggest that the Scrumage approach was effective at promoting student ownership:

- "I think this learning approach was successful for getting me to take ownership of learning material. It allowed me to be able to study at my own pace ... Compared to my other classes I would honestly rate this one the highest in terms of how much I' ve been motivated to understand the concepts and it' s the class I feel I' ve learned the most in."
- "This freedom to decide between videos, slides, readings, and lectures, not only made class more meaningful, but also helped dampen the feeling that showing up to class was a chore (ex. if you felt comfortable with the topic, the lecture may not have been necessary to stay for)."
- "···[The] sprint style was my favorite class organization I have seen... The division into smaller topics helped make the course load feel more manageable, and allowed additional emphasis to be put on each topic.

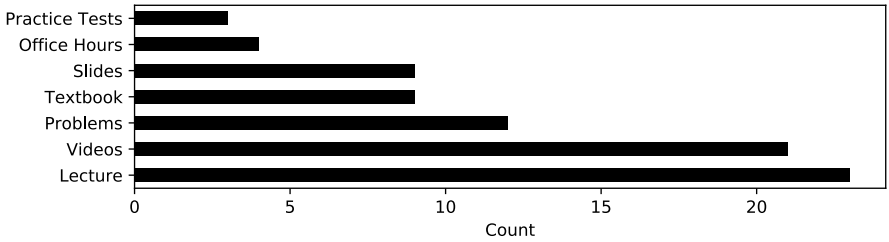


Figure 3: Counts of which learning approaches were mentioned in student responses to a survey question about how students chose to learn material.

> Further by having group and individual work for each sprint, I was able to improve my teamwork skills, and also ask questions to help work through problems and facilitate my own discussion and learning."

4.1.3 Metalearning

One survey question asked students to indicate their strategy for using the available resources to learn the material. Gratifyingly, many students reported progress in learning about their own learning:

- "I learned over the semester that my best way to learn was to watch the videos and look over the slides before a class. This way, I could ask questions during the lecture, and reinforce what I had taught myself."
- "I generally went over lecture slides first, then if I still had trouble understanding material I went to the book, and if I still wasn't sure I used the videos. I tried to understand the material first before going into solving problems so if I got stuck I would know where to refer to for answers."
- "I primarily learned from jumping into the problems and trying to piece it together with videos textbook and slide. If all else failed I went straight to office hours."
- "···My approach changed slightly over time as I realized the pre class prep is what helped me the most for truly understanding the topics."

Coding the free text responses indicates that lectures were the most popular learning approach, closely followed by videos (Figure 3). Of note, most students mentioned preferring multiple learning modalities.

5 Conclusion and Future Work

Our results suggest that the Scrumage teaching approach can be effectively applied to an Algorithms course and resulted in students taking more ownership over their learning, discovering how they personally learn best, and having better impressions of the course and the topic of Algorithm Analysis.

While we saw a marked increase in attitudes, grade outcomes from the course were similar to semesters that followed a traditional pedagogical approach. However, because a number of factors change from one course offering to another, no clear conclusions can be drawn from this outcome. We look forward to further studying the effect on content learning as well as the effect on instructor experience in the future.

Finally, we note that, from the instructor perspective, Scrumage resulted in a more enjoyable teaching experience. Students appeared more motivated and better prepared for class, often asking better questions that suggested they had already attempted to solve homework problems on the topic of discussion. Our experience has been that less time is needed for spoon-feeding students the basics, and more time is spent on applications of techniques and drawing connections between different ideas.

References

[1] Azita Iliya Abdul Jabbar and Patrick Felicia. Gameplay engagement and learning in game-based learning: A systematic review. *Review of educational research*, 85(4):740–779, 2015.

[2] Jacob Lowell Bishop, Matthew A Verleger, et al. The flipped classroom: A survey of the research. In *ASEE national conference proceedings, Atlanta, GA*, volume 30, pages 1–18, 2013.

[3] Florent Bouchez-Tichadou. Problem solving to teach advanced algorithms in heterogeneous groups. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, page 200–205. ACM, 2018.

[4] Mary Burgan. In defense of lecturing. *Change: The Magazine of Higher Learning*, 38(6):30–34, 2006.

[5] Donald Chinn. Peer assessment in the algorithms course. *SIGCSE Bull.*, 37(3):69–73, June 2005.

[6] Shannon Duvall, Dugald Ralph Hutchings, and Robert C Duvall. Scrumage: A method for incorporating multiple, simultaneous pedagogical

styles in the classroom. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 928–933, 2018.

[7] Shannon Duvall, Duke Hutchings, and Michele Kleckner. Changing perceptions of discrete mathematics through scrum-based course management practices. *Journal of Computing Sciences in Colleges*, 33(2):182–189, 2017.

[8] Tommy Färnqvist and Fredrik Heintz. Competition and feedback through automated assessment in a data structures and algorithms course. In *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education*, page 130–135. ACM, 2016.

[9] Tommy Färnqvist, Fredrik Heintz, Patrick Lambrix, Linda Mannila, and Chunyan Wang. Supporting active learning by introducing an interactive teaching tool in a data structures and algorithms course. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, page 663–668. ACM, 2016.

[10] Lasse Hakulinen. Card games for teaching data structures and algorithms. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, page 120–121. ACM, 2011.

[11] Alison King. From sage on the stage to guide on the side. *College teaching*, 41(1):30–35, 1993.

[12] Anany Levitin and Mary-Angela Papalaskari. Using puzzles in teaching algorithms. *SIGCSE Bull.*, 34(1):292–296, February 2002.

[13] ACM Joint Task Force on Computing Curricula. Computer science curricula 2013: Curriculum guidelines for undergraduate degree programs in computer science, 2013.

[14] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[15] P. R. Pintrich, D. Smith, T. Garcia, and W. McKeachie. A manual for the use of the motivated strategies for learning questionnaire (mslq). 1991.

[16] Jeff Sutherland and JJ Sutherland. *Scrum: the art of doing twice the work in half the time*. Currency, 2014.

Modified Specifications Grading in Computer Science: Preliminary Assessment and Experience Across Five Undergraduate Courses*

Kevin R. Sanft, Brian Drawert, and Adam Whitley
Department of Computer Science
University of North Carolina Asheville
1 University Heights, Asheville, NC, 28804
{ksanft,bdrawert,awhitley}@unca.edu

Abstract

Specifications grading has been proposed as a method to improve student outcomes and reduce faculty grading time. There are many ways to implement specifications grading that differ in their details, but the overarching philosophy is that student work receives credit only when it fully satisfies a given set of requirements. Therefore, grading requires only a satisfactory/unsatisfactory designation, reducing grading time. By allowing students to resubmit assignments, it encourages mastery of the material. We present a version of specifications grading that was implemented across five undergraduate computer science courses over four semesters, comprising twelve total course offerings. We find evidence for improved student outcomes, especially among middle to low performing students. Grading time was reduced for most programming assignments. Assignments with many small independent parts, such as short answer textbook problems, generally did not lead to grading time savings. Student enjoyment of specifications grading when compared to traditional grading was polarized with some students strongly disliking it. Overall, the modified specifications grading scheme presented here offers a number of benefits to students and faculty that make it an appealing option for undergraduate computer science courses.

1 Introduction

Specifications grading has gained popularity since it was described by Nilson in 2015 [5]. Several investigators have reported their experience in applying variations of specifications grading across a variety of fields [10, 7], including computer science [2, 6, 8, 3, 1]. In its prototypical form, ("standard") specifications grading is characterized by: satisfactory/unsatisfactory (i.e. pass/fail or binary) grading of assignments, final letter grades determined by "bundles" of satisfactory assignments, and allocated "tokens" that students can spend to resubmit unsatisfactory assignments [4]. Rather than a prescriptive scheme, however, specifications grading is better viewed as a philosophy that emphasizes experience and demonstrated mastery of material over partial credit grading. In that sense, specifications grading has much in common with longstanding pass/resubmit pedagogical approaches (e.g. [9]). Therefore, there are many variations that can be called specifications grading [5]. Independent of the implementation details, the purported benefits of specifications grading include improved student engagement and learning outcomes and reduced grading time for instructors.

In this work we describe the experience of using a modified specifications grading variant in several sections of five undergraduate computer science courses over four semesters at University of North Carolina Asheville (UNCA). The remainder of this paper is organized as follows. In the next section we describe the implementation details of the modified specifications grading scheme used in this study and the courses in which it was used. Section 3 presents results of a grade timing study and a student perception survey. Section 4 presents a discussion of lessons learned and various trade-offs to consider when implementing specifications grading. Section 5 provides a summary of the work.

2 Modified Specifications Grading and Course Descriptions

2.1 Modified Specifications Grading

The specifications grading scheme used in this study featured *pass/resubmit grading* on assignments. Assignments could include *homework*, which are mostly short answer problems or small programming problems, typically from a textbook; *labs*, which are small to moderate programming tasks that are often started in class, sometimes with students working in pairs; and *projects*, which are larger programming problems. Each assignment had an initial due date and was graded out of 10 points. An initial round of grading was completed

shortly after the due date with students receiving either 10/10 if they satisfied 100% of the requirements or 0/10 if they did not satisfy the requirements, which includes students who did not turn in the assignment. Any student with a score of 0 can update their solution and resubmit. Each submission (or resubmission) after the initial deadline incurs a penalty of 1 point. Therefore, the only way to earn the full 10/10 on an assignment is to submit it prior to the original deadline and satisfy all the requirements. After the initial deadline, instructors monitor for resubmissions and grade them as needed. In practice, instructors can use discretion to deem the assignment to be satisfactory even if minor errors exist. In some courses, there was an imposed deadline after which resubmissions would no longer be accepted. For example, in Data Structures students could resubmit any time within one week after the next exam following the assignment's original due date.

Beyond the pass/resubmit grading on assignments described above, the other aspects of the courses were structured traditionally. All courses had exams with traditional grading. Final course grades were determined using a weighting formula with heavy weight on exams. Different types of assignments, i.e. homework, labs, and projects, could have different weights in the final grade calculations. Since the pass/resubmit grading considered here results in a numerical grade, the only substantial difference between this modified specifications grading scheme and a traditional course is the grading and resubmission of assignments. The effect is that assignments require demonstrated mastery, since students receive no credit without successful completion, and low stakes, since students lose only 10% per attempt.

2.2 Courses in this Study

The authors used the modified specifications grading scheme in five different courses over four semesters, comprising twelve unique course offerings. This subsection presents brief descriptions of the courses. Table 1 summarises the course offerings and initial enrollments.

CS0 *Intro Programming for Web Applications.* This is an introductory programming course open to majors and non-majors. It is one of three CS0 options, of which CS majors and minors must take one. Non-majors receive credit for UNCA's *Scientific Perspectives* general education requirement. This CS0 option has a web focus, so students learn basic HTML, CSS, and programming in JavaScript. There are no prerequisites for this course.

Data Structures (D.S.) This is a standard data structures course in Java that is required for CS majors and minors. The prerequisites are CS0, which is described above, and CS1, which is an Introduction to Object-Oriented Programming in Java. Data Structures is considered the gateway course in the CS curriculum at UNCA, as a grade of C or better in this course is a prerequisite

Course	Semester(s)	Instructor(s)	Enrollment
CS0	F18, S19, F19, S20	BD	20, 20, 18, 18
Data Structures	S20	KRS, BD	32
Algorithms	S19, F19, S20	AW	29, 21, 27
Cybersecurity	S19, S20	BD	18, 18
Systems	F18, F19	BD	17, 17

Table 1: Courses in this study from Fall 2018 through Spring 2020. Enrollment is the number of students initially enrolled in the course. Note the Data Structures course comprised two sections that were team-taught by authors Sanft and Drawert.

for most upper-level courses.

Algorithms (Alg.) This is a standard upper level algorithms course in Java that is required for CS majors and is an elective for CS minors. The prerequisite is a grade of C or better in the Data Structures course.

Cybersecurity (Security/Sec.) This is an upper level elective for CS majors and minors. The prerequisite is a grade of C or better in the Data Structures course.

Systems (Sys.) This is an upper level course that is required in one CS major track (*Computer Systems*) and an elective in the other CS major track (*Information Systems*) and in the CS minor. The prerequisites are an Introduction to Systems course and a grade of C or better in the Data Structures course.

3 Results

Of particular concern for instructors considering modified specifications grading is whether the grading time will be reduced. Total grading time for specifications grading includes the initial grading session and potentially several additional resubmission grading sessions. In the next subsection we present the distribution of resubmissions and the results of a grade timing study. In Section 3.2, we present the results of a survey assessing student perception of specifications grading compared to traditional grading.

3.1 Resubmissions and Instructor Grading Time

Table 2 shows the distribution of student submissions per assignment. Data Structures had the highest number of submissions with 1.37 submissions per student per assignment, while also having the lowest number of students who

never submitted. This was expected, considering the course's placement within the curriculum. In CS0, assignment specifications are detailed and explicit, so students can more easily self-check correctness. In upper level courses, the specifications are higher-level but the students are more sophisticated and better able to test edge and corner cases. Data Structures occurs between those extremes with fewer details in the specifications but the students are less skilled at identifying edge and corner cases themselves. The number of students who do not submit assignments should be higher than in traditional grading, since students should not submit if they know their work is incorrect and will not receive credit. The number of zero submissions in Systems was unusually high for the semester included in Table 2, in part due to an atypical number of students (5 out of 17) who effectively quit during the semester due to personal circumstances without officially withdrawing from the course.

Course	Submissions per Assignment					
(term)	0	1	2	3	4	Mean (SD)
CS0 (S20)	14.6%	79.2%	6.3%	0%	0%	0.92 (0.45)
D.S. (S20)	8%	61%	19%	10.5%	1.5%	1.37 (.83)
Alg. (S19)	13.8%	60.3%	21.1%	4.3%	0.4%	1.17 (.73)
Security (S20)	22%	72%	6%	0%	0%	0.83 (0.5)
Systems (F19)	35%	55%	8%	2.2%	0.4%	0.78 (0.71)

Table 2: Distribution of number of submissions per student across all assignments. The maximum number of submissions encountered across all semesters was four, which occurred rarely.

Table 3 shows the results of a grade timing experiment comparing specifications grading to traditional grading for a selection of assignments. For each assignment, a random sample of approximately ten students per grading method was selected. Overhead (accessing the assignment, additional grading time for the first submission, setting up a grading spreadsheet, etc.) and the average time per submission were recorded. Based on the sample, the total grading time for the assignment was estimated for both grading methods. We note that all the samples were taken from courses that used specifications grading. To estimate the traditional grading time, we assumed that the number of submissions was equal to the number of initial submissions observed in specifications grading. We expect this to underestimate the traditional grading time, as traditional grading encourages submitting incorrect work to receive partial credit.

Grading correct work is relatively fast in both traditional and specifications grading. However, the time savings of specifications grading that are seen in Table 3 come from: 1) not spending time allocating partial credit points, and 2)

not grading as much incorrect work because students are less likely to submit work that they know will be worth zero points. However, the grading time benefits are offset somewhat by having to grade resubmissions.

There is substantial variation in the time savings of specifications grading. Some of the variation is due to the sampling of students in the timing experiment. For example, with Algorithms "HW4" where specifications grading actually took longer, 7 of the 10 submissions in the traditional grading sample were "good" submissions, while only 5 of the 10 initial submissions were satisfactory in the specifications grading sample. In general, the time savings for specifications grading is highest when assigning partial credit in traditional grading is most complex. For example, there was essentially no time savings benefit on Data Structures "HW2" , which featured several textbook questions that were graded independently and required few, if any, partial credit decisions in traditional grading. Conversely, it is more time consuming to assign partial credit on assignments with several interrelated components, as in programming project "Proj1" in Security.

Assignment	Overhead			Time per submission			**Time**
	Trad.	Specs.	Resub.	Trad.	Specs.	Resub.	**Savings**
CS0 Lab9	288s	120s	27s	84.2s	41.4s	26s	**47%**
D.S. HW2	123s	65s	50s	81s	71s	26.3s	**0.4%**
D.S. Proj3	270s	137s	97s	273s	74.5s	73.7s	**53%**
Sec. Proj1	556s	182s	89s	380.4s	52.2s	40s	**83%**
Sec. Paper1	249s	183s	25s	61.6s	41s	33s	**29%**
Alg. HW4	502s	298s	127s	141.7s	126.1s	87.4s	**-9.0%**
Sys. HW2	229s	125s	66s	83s	32.8s	33s	**49%**
Sys. Lab3	348s	304s	104s	68.6s	64.4s	32s	**-8%**

Table 3: Instructor grading time and time savings for specifications grading compared to traditional grading for a selection of assignments and random samples of students. Grading time is split into Overhead and an average grading time per submission. Traditional (Trad.) grading is done in one grading session. Specifications grading has an initial grading session (Specs.) and typically one to three resubmission grading sessions (Resub.). The number of resubmission grading sessions is usually the same as the maximum number of resubmissions for that assignment (see Table 2 for averages). Grading time for specifications grading was estimated by applying the grading time averages of the sample students to the actual distribution of submissions. Grading time for traditional grading was estimated by assuming the number of submissions was equal to the number of initial submissions in specifications grading.

3.2 Student Perception

Improved student outcomes is the most important reason for adopting specifications grading. Anecdotally, we find improved student outcomes, and subjective discussion appears in Section 4. Unfortunately, conducting a randomized trial to assess the direction and magnitude of the effect was not feasible in this study. However, we were able to conduct a small student perception survey in CS0 and Data Structures in Spring 2020. The survey asked students their agreement with two statements: 1) Compared to traditional grading of assignments, I prefer specifications grading, and 2) Compared to traditional grading of assignments, specifications grading helped me learn the material better. Responses used a standard Likert scale: 1=Strongly Disagree, 2=Disagree, 3=Neutral, 4=Agree, 5=Strongly Agree.

Statement (vs. traditional grading)	Course	Response (5=Strongly Agree)				
		1	2	3	4	5
I prefer specifications grading.	CS0	0	1	1	5	1
	D.S.	2	1	1	2	2
	Total	2	2	2	7	3
Specifications grading helped me learn the material better.	CS0	0	0	3	2	3
	D.S.	0	0	4	2	2
	Total	0	0	7	4	5

Table 4: Student perception survey results from Spring 2020 for CS0 and Data Structures. Responses are on a Likert scale with 1=Strongly Disagree, 2=Disagree, 3=Neutral, 4=Agree, and 5=Strongly Agree. Response rates were 8/18 (44%) and 8/21 (38%) for CS0 and Data Structures, respectively. The total N denominator values differ from Table 1 due to attrition, which was affected by the COVID-19 pandemic.

As shown in Table 4, students had mixed opinions on whether they preferred specifications grading over traditional grading. But no student who responded disagreed with the statement that specifications grading helped them learn the material better. Intuitively, this aligns with our experience that the opportunity and incentive to fix mistakes improves student outcomes. We note that the survey could be subject to selection bias, as only students who completed the course and were motivated to respond were counted. Of particular concern is that student attrition could be influenced by students who strongly dislike specifications grading, yet students who withdrew from the course did not participate in the survey. Despite this concern, the survey results in Table 4 from Spring 2020 generally agree with the anecdotal experience of authors Draw-

ert and Whitley from previous semesters that specifications grading helped students learn the material better.

4 Discussion

The modified specifications grading scheme presented here seems particularly beneficial for low to middle performing students. These are students who may in traditional grading receive individual assignment scores in the 50-70% range (i.e. low "C" to failing range) and may have an assignment average that is "passing" but are nonetheless likely to perform poorly on the next exam. With pass/resubmit grading, these students are incentivized to revisit incorrect assignments and fix their mistakes. In introductory courses like CS0, assignment specifications tend to be explicit and detailed because students are not yet sophisticated enough to fully consider edge and corner cases, while in upper level courses, the specifications can be higher level with less detail because the students are better at identifying special cases and testing. Intermediate courses such as Data Structures present the most interesting tradeoffs when considering specification detail. On the one hand, making assignment specifications explicit and exhaustive increases the likelihood that students will submit correct work and, in turn, reduces grading time. But, on the other hand, providing a complete checklist of test cases may prevent students from developing the crucial skills of determining these details themselves.

In one real worst case example, all students in Data Structures who submitted a solution to a particular assignment problem got it wrong on their first attempt! The problem was to write an instance method to determine the size (number of elements) for a particular circular array queue implementation without using a size instance variable. The correct solution requires students to correctly handle the cases where the array index of the front of the queue is smaller than, greater than, and equal to the index of the back of the queue. Furthermore, when the indexes of the front and back are equal, it could mean that the array is empty or full, requiring additional code to discriminate between those two conditions. However, the assignment specifications did not enumerate these cases. Some students perceived this as a failure of the instructor's teaching or poor assignment specifications, but these failures are important learning tools. By the end of the semester, students were much better at identifying corner and edge cases and were less likely to submit incorrect work.

4.1 Pros and Cons of Specifications Grading

There are several advantages and disadvantages of specifications grading. These are outlined in the next two subsections.

4.1.1 Pros

- Instructor grading time is usually reduced (see Table 3). The benefit is primarily due to spending less time assigning partial credit on programming tasks, with negligible benefit for short-answer response questions. The benefit is enhanced by eliminating the incentive for students to turn in incorrect work.

- Instructors are motivated to grade assignments quickly.

- Students learn the material better. We have only anecdotal support for this claim, but it seems especially true for low to middle performing students who are forced to revisit and fix incorrect submissions.

- Low performing students are better able to self-assess their performance. A student who receives a score of 60% on a traditionally graded assignment might feel they are "passing", even though they are positioned to perform poorly on an exam, while a student receiving zeros on assignments in specifications grading is motivated to revisit and correct their mistakes.

- Students learn that taking time to ensure that their work is correct will actually save them time in the end.

4.1.2 Cons

- Resubmissions require more grading sessions, and grade turnaround time should be short to be most effective. The unpredictable nature of resubmissions can be burdensome for instructors. Specifications that include programming practices or checking for plagiarism may require closer inspection of the code, limiting the time saving benefits.

- Students can feel frustrated that their grade of zero does not reflect how close they were to a correct solution or their effort.

- A backlog of assignments to resubmit, along with new assignments, can become overwhelming for students. A backlog can be exacerbated if the instructor does not provide fast turnaround on grading.

- Time spent by students correcting previous submissions could be spent on additional assignments instead.

- Instructors may be reluctant to review assignment solutions in class if some students are still working on resubmissions.

The pros and cons are interrelated. For example, specifications grading works best when feedback is timely, which encourages instructors to grade work promptly. However, this increases the burden on instructors, undermining the grading time saving benefits. In the student perception survey, students were given the opportunity to provide comments. The majority of negative comments were related to instructors needing to grade assignments quickly and assignments piling up. From the instructor perspective, author Sanft particularly disliked the unpredictable nature of grading resubmissions and the feeling that grading was always on his to-do list.

4.2 Implementation Details to Consider

When considering implementing specifications grading, there are several details to consider. In "standard" specifications grading, final letter grades are determined by bundles of assignments where students may need to complete, say, the first seven assignments to receive a grade of "C" and ten assignments to receive an "A". This may work well in introductory programming courses. Bundles also can be beneficial for top students who can be pushed with a set of more challenging assignments. However, we prefer the modified specifications grading scheme presented here, in part, because it reduces the incentive for academic dishonesty (e.g. plagiarism). By having traditional exams that have high weight in the final grade determination, the grade benefits of cheating on an assignment should be smaller than the negative effect on the exam grade from not learning the material. Assignments were given weights ranging from 15% to 30% in the five courses in this study. Top students can still be challenged by providing some difficult problems on the exams and/or by offering additional "stretch" assignments as extra credit.

In modified specifications grading, instructors can also consider the details of assignment resubmissions and regrading. In many implementations of specifications grading, students are issued or may earn "tokens" which they can spend to resubmit unsatisfactory assignments. Using tokens can reduce the number of resubmissions and, hence, grading time. This can be used with or without a grade penalty for resubmissions. A policy to earn tokens can also serve to motivate top students who might compete to earn the most tokens [4]. We did not use tokens, but grading turnaround time delays and diminishing returns of the 10% grade penalty naturally limited the number of resubmissions (see Table 2). However, it may be beneficial to have a grading policy that explicitly states when grading sessions will happen. An example policy could be that resubmission grading sessions will be done on Mondays, so any resubmission submitted by Monday at 8:00am will be graded on Monday, while anything after that will be graded the following Monday. An explicit policy can set student expectations appropriately and make resubmission grading more predictable for instructors.

The amount of feedback to give when grading assignments is also a consideration. Author Sanft tended to give less feedback in specifications grading compared to traditional grading. This encourages students to figure out the errors themselves or contact the instructor via email or office hours. Students who follow up via email or office hour visits tend to benefit from these interactions, however, this strategy might benefit certain types of students, i.e. those students who are comfortable contacting the instructor, over others. Author Whitley tended to give equivalent feedback in specifications grading and traditional grading. Though this limits the grading time savings somewhat, as he put it "[specifications grading] eases the mental burden of grading...it allows me to focus on the problem [the student is having] rather than on how many points to take off".

5 Conclusions

The experience of using modified specifications grading in five undergraduate computer science courses over four semesters has been largely positive. The grading scheme featured pass/resubmit grading of assignments, with a 10% penalty for each resubmission after an initial deadline. The remainder of the courses were structured traditionally, with relatively large weights on exams in the final grade calculations.

Allowing students to resubmit assignments leads to unpredictability in the amount of grading that instructors face. However, this is offset by reduced grading time per submission. Time savings accrue primarily from the simple pass/fail grading rather than spending time determining the amount of partial credit. The unpredictability in grading assignment resubmissions can be mitigated by an explicit policy stating the frequency at which grading sessions will occur.

Beyond reducing grading time, the most important benefit of specifications grading is that it improves student outcomes. While this is difficult to quantify objectively, our experience is that pass/resubmit grading encourages low to middle performing students to spend more time engaging with the material by revisiting and correcting their own mistakes. This anecdotal experience agrees with the results of a small student perception survey where students were all neutral or agreed with the statement that specifications grading helped them learn the material better than traditional grading. Furthermore, specifications grading reinforces in students the important lesson that taking the time to ensure their work is correct is worthwhile and will actually save them time in the end.

The modified specifications grading scheme presented here can be utilized throughout the computer science curriculum. In introductory courses, assign-

ment specifications can be given as an explicit checklist, effectively allowing students to self-grade their assignments. In upper level courses, specifications can be higher-level, forcing students to identify and test the edge and corner cases themselves. Specifications grading is perhaps most challenging for instructors and students at the intermediate level. In this study, the Data Structures gateway course featured the highest levels of assignment resubmissions, meaning students were initially submitting incorrect work and instructors were doing more grading. This should be viewed as part of the learning process; by the end of the semester nearly all students who remained in the course were able to satisfy the course learning objectives. Our experience across five courses suggests that modified specifications grading is an appealing option for computer science courses.

Acknowledgements

We thank the Center for Teaching and Learning and the Department of Computer Science at UNCA for their guidance and support of this work.

References

[1] Andrew Berns. Scored out of 10: Experiences with binary grading across the curriculum. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 1152–1157, New York, NY, USA, 2020. Association for Computing Machinery.

[2] James W. McGuffee, David L. Largent, and Christian Roberson. Transform your computer science course with specifications grading. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019.

[3] Grace M. Mirsky. Effectiveness of specifications grading in teaching technical writing to computer science students. *J. Comput. Sci. Coll.*, 34(1):104–110, October 2018.

[4] L. Nilson. Yes, virginia, there's a better way to grade. `https://www.insidehighered.com/views/2016/01/19/new-ways-grade-more-effectively-essay`, accessed 05/22/2020.

[5] L. Nilson. *Specifications Grading: Restoring Rigor, Motivating Students, and Saving Faculty Time*. Stylus Publishing, LLC, 2015.

[6] Christian Roberson. Techniques for using specifications grading in computer science. *J. Comput. Sci. Coll.*, 33(6):192–193, June 2018.

[7] Robin Pals Rylaarsdam and Cheryl Heinz. Specifications based grading: Nearly pointless education that points students to key concepts. *The FASEB Journal*, 30(1_supplement):662.13–662.13, 2016.

[8] Torstein J. F. Strømme. Pass/fail grading and educational practices in computer science. In *Proceedings of Norsk Informatikkonferanse*, 2019. `https://ojs.bibsys.no/index.php/NIK/issue/view/51`.

[9] R. Stutzman and K. Race. Emrf: Everyday rubric grading. *Mathematics Teacher*, 97(1), January 2004.

[10] Mai Yin Tsoi, Mary E. Anzovino, Amy H. Lin Erickson, Edward R. Forringer, Emily Henary, Angela Lively, Michael S. Morton, Karen Perell-Gerson, Stan Perrine, Omar Villanueva, MaryGeorge Whitney, and Cynthia M. Woodbridge. Variations in implementation of specifications grading in stem courses. *Georgia Journal of Science*, 77(2), 2019.

A Novel Framework for Integrating Mobile Machine Learning and L2 Vocabulary Acquisition*

Jacob Ginn,[1†] *Cesar Bautista Salas,*[1†]
Scott Barlowe,[1] *and Will Lehman*[2]
[1]*Department of Mathematics and Computer Science*
[2]*Department of World Languages*
Western Carolina University
Cullowhee, NC 28723
{jjginn1, cubautistasalas1}@catamount.wcu.edu
{sabarlowe, welehman}@email.wcu.edu

Abstract

Cross-cultural interactions are increasingly important and encompass many types of activities including recreational, educational, financial, and scientific. Exchange of ideas during these activities is often hindered by the inability of those involved to understand diverse written or verbal communication. Language acquisition is the process through which languages are learned and mastering vocabulary is the foundation of that process. However, techniques employed when teaching vocabulary often consist of exercises that are inadequate substitutes for meaningful and contextual interaction in the target language. This paper presents a novel system that utilizes real-time object recognition and game play to provide a high level of interaction with the surrounding environment during vocabulary acquisition. The system provides immediate feedback with either a structured or unstructured approach and supports multiple languages. We conclude with a discussion of the strengths and weaknesses of the system and areas of future work, all based on the cross-disciplinary perspectives of the authors.

†Contributed equally

1 Introduction

Interactions spanning multiple cultures and geographical locations are becoming increasingly common. The inability to express or perceive ideas during these interactions can cause a host of problems. For example, children may be kept from making academic progress in a new country, business leaders may find expansion into new markets more difficult, and research collaborations could take longer to find solutions to important problems.

One of the primary ways of bridging intercultural communication gaps is through the learning of other languages, a process known as language acquisition. While virtually everyone is able to communicate in his or her native language (known as L1) after just a few short years of life, most people find it extremely difficult to master a second language (known as L2) outside of the full-immersion experience, and this difficulty increases substantially with age. Vocabulary is the foundation of all language acquisition, and without it, nothing meaningful can be understood or communicated [7]. It follows that new methods of learning vocabulary that increase interactive and contextual interactions with the L2 lexicon can speed and deepen language learning [11].

Mobile devices are now ubiquitous and provide a platform for making language acquisition more interactive and contextual. Mobile technology users are not only comfortable using their devices to learn, but increasingly expect it. Mobile devices can provide students a way to interact with their immediate surroundings, making abstract ideas more concrete. Mobility also provides instructors the flexibility of accommodating students with various schedules, learning speeds, and physical locations.

We present Vocabulo, a novel mobile application that supports interactive learning in the context of the learner's current surroundings. Vocabulo differs from other mobile vocabulary learning applications by smoothly integrating:

- real-time object recognition and label translation;
- immediate feedback on vocabulary recall ability;
- creation, modification, and utilization of target word lists corresponding to the user's environment; and
- support of multiple languages.

The paper is organized as follows. First, we present background information on vocabulary instruction including the basics of vocabulary acquisition pedagogy and relevant computer use. We next describe both the system design and the user experience. The strengths and limitations of our work are then discussed. Finally, we conclude with future work.

2 Background

Language learning methodologies have a rich history spanning millennia. The techniques employed have continuously evolved as the specific needs of the learners have changed. Current pedagogy is nothing near homogeneous and is influenced by a variety of philosophies. However, all recent attempts at finding suitable learning tasks integrate computer technology.

2.1 Pedagogy for Vocabulary Acquisition

Strategies for building L2 vocabulary span a wide variety of approaches. Schmitt [11] reports that formal vocabulary instruction goes back at least to the second century BC in Greek and Roman societies. More recently, the Grammar-Translation method was the guiding philosophy during the nineteenth century and focused on the analysis and translation of (usually ancient) texts rather than on practical, conversational use. The Direct Method followed and favored realistic exposure to the new language. This technique shunned irrelevant translation exercises, but it failed to take into account the important differences between L1 and L2 acquisition. Later approaches included the Audiolingual Method and the now nearly ubiquitous Communicative Approach which, like earlier approaches, have their own preferred strategies and practices regarding vocabulary building. The fact that all these modern methodologies still coexist in formal instruction and continually compete with an onslaught of new products claiming to make language learning painless lends credence to the widespread belief among teachers that no single methodology is intrinsically better than others in all situations [8].

The lack of a single, proven method for L2 vocabulary acquisition makes defining a specific suite of tasks for student completion difficult. Instead, instructors may choose to rely on guiding principles when designing tasks. One of the most prominent frameworks specifically for guiding task development during vocabulary acquisition was developed by I.S.P. Nation [9]. Nation proposes a framework consisting of three processes in recalling vocabulary. The first step is *noticing*. Noticing is the acknowledgement by the learner that a word is important in the language. Motivation and interest are important during this phase. A definition and translation in the L1 language are common characteristics of noticing which often encompasses supporting tasks such as deliberately studying a word, guessing a word's meaning from context, or having a word explained. *Retrieval* is the next step and often requires repetition. The third step is *creative use* or *generative use* where there are encounters with target words in varying contexts and alternatives to previously learned meanings. There are many degrees of generative use and developing a classification system that appropriately conveys the level of generation is difficult and an

area of ongoing research.

2.2 Computerized Vocabulary Acquisition

Since the 1990s, the use of computer technology for L2 vocabulary building has increased dramatically such that now virtually every foreign language textbook includes computer-based supplements. The most common of these are simple, two-sided online flashcards with audio pronunciation for the L2 terms. The inadequacies of these basic tools are overcome by web applications like Quizlet [4] which allow instructors and students to build their own vocabulary lists and then quiz themselves. A few of these employ artificial intelligence to re-quiz a term at changing intervals, based on an individual' s quiz results with that term. This technique, commonly referred to as spaced repetition, is cited by Nation [9] as highly effective.

The use of more sophisticated artificial intelligence in L2 vocabulary building applications has only recently become possible, and at the time of this writing, there are no widely available applications that incorporate high-level artificial intelligence techniques such as the instant classification and translation of objects captured by mobile video. One mobile application that approaches the capabilities of Vocabulo is a system announced by Microsoft in 2019 called Read My World [10] which is still being developed and tested. According to the press release, the system allows learners of English to take pictures, after which the spelling and the pronunciation of objects recognized in the image are provided. The application then gives the user the opportunity to store the image for later review in one of the three provided games. The application also supports instantaneous translations and pronunciations of written text, a feature already available in a host of cloud-based machine learning platforms.

Read My World [10] and the system presented here both incorporate objects recognized in the user' s surrounding environment, use game play mechanics as a learning tool, and make extensive use of cloud services and machine learning APIs. However, our system and Read My World have important differences. The two frameworks differ in target audience, the time and type of feedback, the space needed for asset storage, the potential involvement of an instructor, and the specific platforms used. A more detailed description of Vocabulo' s features will be provided next.

3 System Design

Vocabulo utilizes real-time object detection to provide interaction with the learner's environment and immediate feedback during language acquisition. The user focuses the camera on an object of interest. In the simplest mode, an instantaneous translation of the object label is given. The user can also

choose a quiz mode that tests the learner on the translation of objects as they are encountered. Testing can take the form of either a multiple choice or fill-in-the-blank question. The sequence of events for the quiz mode is shown in Figure 1(a). In the unstructured approach, questions are based on any object of interest in the immediate surroundings. The structured approach uses an embedded text file containing a list of target objects to be found, recognized by the application, and reviewed. Storing a text file of target words instead of individual images provides greater flexibility and preserves storage space. The list of target objects can be created by either the learner or deployed by an instructor, an important characteristic of well-designed vocabulary programs [9].

Vocabulo consists of a main application, object capture, object recognition, and label translation. The application utilizes a mix of on-device and off-device resources which provides real-time access to popular third-party services. The system's general structure is illustrated in Figure 1(b) and consists of the following components:

- **Main application**. The main application is currently designed for Android devices, implemented in the Kotlin language [2], and utilizes several third-party services. After login, the main application is responsible for managing user options, directing data flow to and from camera capture and translation services, and controlling the scavenger hunt.
- **Object capture**. User experience depends on the application to recognize objects of interest for which the translation or review of the English label is desired. The object of interest is chosen by centering the object within the camera's focus.
- **Object recognition**. The application uses the on-device version of Firebase ML Kit [3] to provide the English label of the captured object. The on-device version of Firebase ML Kit eliminates potential delay from accessing cloud-based object recognition services and provides flexibility to add more complex, customized TensorFlow Lite [5] models if needed.
- **Label translation**. After capture and recognition, an object label is provided. The label is sent to Google Translate [1] which returns the L2 version of the word. The user can also retrieve the audio representation of the object using on-device text-to-speech.

4 User Experience

Vocabulo requires logging into the system with a Gmail account. Once logged in, the user is presented with several options, the most direct being immediate

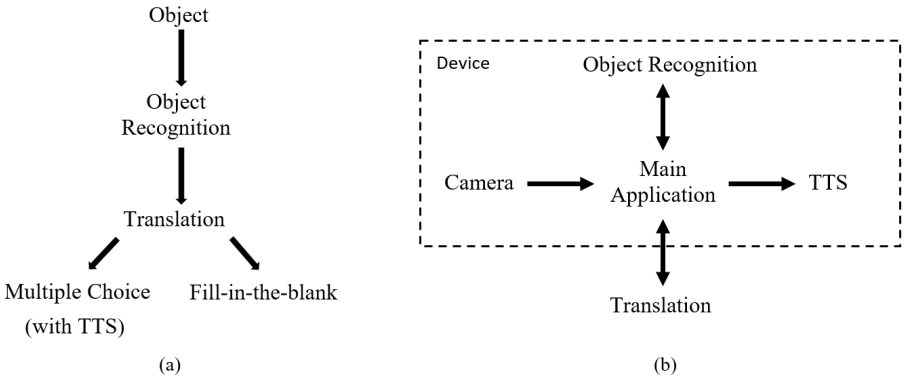


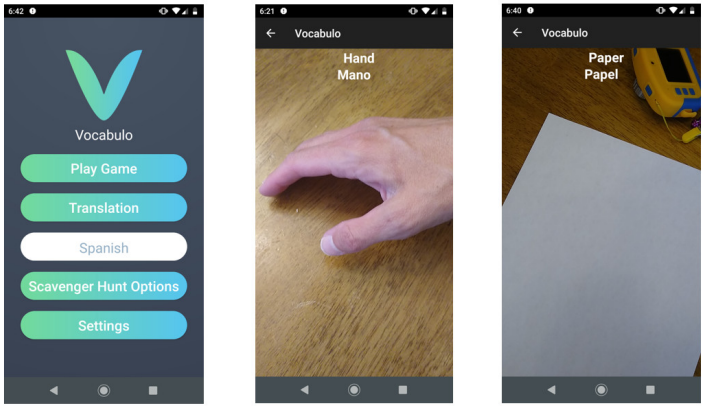
Figure 1: Vocabulo's general structure. (a) The user centers the camera on an object of interest which is recognized, given a label, and translated to the chosen language. In the simplest mode, the translation is immediately shown. In the quiz mode, the translation is used to form a multiple choice or fill-in-the-blank question. (b) The main application is responsible for managing data traffic and game play. An object of focus is captured by the camera and on-device machine learning recognizes and labels the object. The label is translated off-device. On-device text-to-speech translates the object label text to audio in the target language.

translation of surrounding objects (Figure 2). From the main screen, a scavenger hunt may be started, a list of target objects created or edited, and the language changed. Currently, the application supports English speakers who want to learn Spanish, German, French, Japanese, and Italian. The quiz format and characteristics of the audio translation can be changed in the settings menu.

4.1 Unstructured Scavenger Hunt

The scavenger hunt requires the student to find an instance of an object in their surrounding environment. In the unstructured approach, users gain points for knowing the correct translation of any object labeled by the system. The object of interest is simply centered in the camera's field of view and then a button in the lower right corner is selected to notify the application that the user has chosen an object (Figure 3(a)).

Users are immediately quizzed on their recall of the recognized object. After the object is recognized, Vocabulo sends the original label for the object



(a) (b) (c)

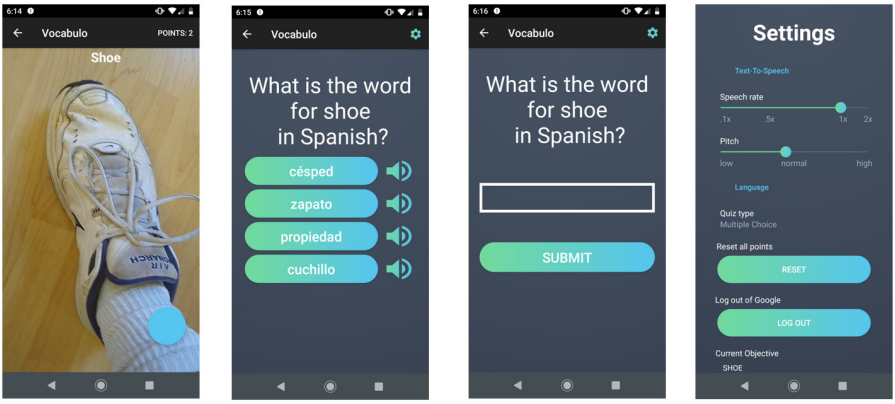
Figure 2: (a) Vocabulo's main screen. (b) and (c) The unstructured approach consists of real-time object recognition and real time translation.

of interest to Google Translate [1] and the L2 label is returned. If the multiple choice format is selected (Figure 3(b)), random words are selected from a dictionary of words stored as a text file. The same translation process just described is followed for the randomly selected words. The translations of the chosen object and the random words are merged to create a quiz question.

Beside each word in the question, there is a speaker icon. When the speaker icon is selected, the pronunciation in the target language is produced. Choosing the correct translation results in a point being added to the running total. If the fill-in-the-blank option is chosen (Figure 3(c)), the learner must enter the word solely from recall. Since this is a more advanced version of recall, two points are added to the user score.

4.2 Structured Scavenger Hunt

An instructor or the student may wish to target a more specific subset of words. To create a target word list (separate from the multiple choice dictionary), the user selects the scavenger hunt option in the main menu followed by the *Create Scavenger Hunt* option. Users can then enter a filename for a text file that will be stored on the device and contains the target words. The label for an object is added to the list by selecting a button in the lower right corner when the object of interest is recognized by the application and the label of interest appears on the display. Additional objects can be added to the list at a later time by selecting *Edit Scavenger Hunt* and entering the filename of the text



(a) (b) (c) (d)

Figure 3: (a) The user can choose the object currently in camera focus by tapping the blue button on the bottom right. (b) Multiple choice quiz question with optional audio translation. (c) Fill-in-the-blank where the user generates the entire word. (d) Settings include speed, pitch, and quiz type. The current search objective in a structured approach is shown in the bottom left.

file containing the target words. Multiple word lists can be stored on the same device resulting in flexibility in both the difficulty of game play and variation in languages.

To select a particular list for the scavenger hunt, the user chooses *Start Scavenger Hunt* and then enters the appropriate file name. The next target word is displayed on the settings screen which can be visited at any time during game play. When the current target is correctly identified, the next word to be found is updated and displayed for the learner (bottom left in Figure 3(d)).

4.3 Other Settings and Options

There are many options available in Vocabulo to facilitate both game play and learning in addition to those mentioned above (Figure 3(d)). Users can seamlessly move between the multiple choice and fill-in-the-blank question formats. There are also options for adjusting the audio translation pitch and speed. Scores are kept per language and users have the ability to reset all scores.

5 Evaluation and Discussion

The authors include two students from a senior capstone course in computer science, a computer science faculty member, and a world languages faculty member. The design was initially conceived by the computer science faculty member and implemented by the senior capstone students. The world languages faculty member provided expertise in language acquisition pedagogy.

Our system' s design is guided by widely accepted pedagogical standards for language learning, specifically the framework proposed by Nation [9]. Because the user is presented with and focuses on one recognized object at a time, our system encourages noticing, which is further enhanced if game play is guided by a list of target words deemed important by an instructor. Vocabulo supports retrieval by providing two quiz format types. Its game feature, the scavenger hunt, is one of the fundamental activities of the Communicative Approach, with the key difference that, in the traditional classroom setting, the information that the students are trying to find in their environment (from other students), is strictly audio, whereas with Vocabulo it is both audio and visual. This sensory combination increases retention [6] and therefore contributes to the student' s generative use of the term in future situations. In the system presented here, an entire scene is presented to a user in which an object of focus can be labeled, translated, and reviewed in real-time.

The primary drawback in the current system is the level of granularity provided by the artificial intelligence for recognizing complex objects, abstract ideas, or instances of generative use [9]. This is especially problematic in those instances where action, judgement, and context are central to labeling and represented in language at least partly by adjectives, verbs, a broad understanding of the relationships among multiple objects, and language nuances, such as gender. Recognition of complex ideas by computational methods are often investigated in areas of research specific to a given subarea of recognition and labeling. Simultaneous utilization of these techniques in a single mobile application having objectives other than labeling would require not only extensive future work for this application but would also require advances in streamlining and integrating multiple state-of-the-art techniques.

Improving the quality of the underlying artificial intelligence for the language complexities and nuances listed above is a long-term goal, but there are several additions to the current system that would ease adoption into the classroom in the short-term. The application needs to be accessible by operating systems other than Android. Implementation as a web application presents a more manageable approach but may increase latency for services now provided on the device. We plan to evaluate this tradeoff in the future. We also want to strengthen the link between specific application tools and language acquisition pedagogy. Specifically, increasing the functionality for spaced repetition

exercises that map directly to words for which a student needs more practice and deepening the integration with formal assessment, especially of correct pronunciation, are top priorities.

6 Conclusion

This paper presents a novel framework for integrating object detection, translation services, and user interaction. Our framework offers several advantages to other current systems including immediate feedback, decreased storage requirements, and potential instructor involvement. In the future, we want to perform a comprehensive user study, increase the suite of teaching tools, and continually integrate more advanced artificial intelligence techniques.

References

[1] Google Translate. `https://translate.google.com/`.

[2] Kotlin. `https://kotlinlang.org/`.

[3] ML Kit for Firebase. `https://firebase.google.com/docs/ml-kit`.

[4] Quizlet. `https://quizlet.com/`.

[5] Tensorflow Lite. `https://www.tensorflow.org/lite`.

[6] A. Baddeley. *Human Memory: Theory and Practice*. Lawrence Erlbaum Associates, 1990.

[7] D. Gardner. *Exploring Vocabulary: Language in Action*. Routledge, 2013.

[8] W. Littlewood. *Foreign and Second Language Learning: Language-acquisition Research and Its Implications for the Classroom*. Cambridge University Press, 1984.

[9] I.S.P. Nation. *Learning Vocabulary in Another Language*. Cambridge University Press, 2001.

[10] Sarah Perez. Microsoft' s new language learning app uses your phone' s camera and computer vision to teach vocabulary. `https://techcrunch.com/2019/05/23/microsofts-new-language-learning-app-uses-your-phones-camera-and-computer-vision-to-teach-vocabulary/`.

[11] Norbert Schmitt. *Vocabulary in Language Teaching*. Cambridge University Press, 2000.

Chasing Rainbows: Colorizing Black and White Images with Adversarial Learning*

Drew Bowman and Scott Spurlock
Computer Science
Elon University
Elon, NC 27302
{dbowman3,sspurlock}@elon.edu

Abstract

Given a black-and-white photograph, recently proposed methods can generate a single plausible colorization. However, generated colorizations are often biased toward average colors, creating outputs that are muted and unnatural looking. Our proposed model learns to generate multiple diverse colorizations for a given grayscale input. Multiple outputs allow each colorization to appear more vibrant and realistic. The final system is implemented as a Generative Adversarial Network (GAN) and incorporates a loss function that encourages diversity across multiple outputs. Experiments with datasets of indoor and outdoor photographs support the effectiveness of our approach qualitatively, and a study with human participants indicates that colorizations generated by our method are perceived as realistic by human observers.

1 Introduction and Related Work

The goal of image colorization is to generate a color image given a black-and-white, or grayscale, input image. The ability to algorithmically generate such a colorization is highly desirable because of how time-consuming this process is when performed manually by skilled artists. Given the number of

Figure 1: Example colorizations generated by our model for a grayscale input.

historical grayscale photos, it would take a very long time to color them all. The Library of Congress has over 8,500 photographs of the Civil War alone. If an artist colored one image per day, it would take nearly 25 years to colorize this collection manually, and there are of course many more historical photographs and even films that might benefit from colorization.

One factor that makes colorization such a challenging problem is that, for any gray pixel in the input, there are many possible colors consistent with it. For example, a gray balloon could have originally been red or blue or some other color, even, potentially, gray. Thus, the goal of colorization is not to generate the *correct* colors for a given grayscale input, but to generate *plausible* colors. Figure 1 shows several sample colorizations generated by our proposed approach for a grayscale hotel room input photograph. This type of problem poses difficulties for a traditional neural network-based approach seeking to minimize the difference between a predicted colorization and a known ground-truth color image. Such a network would tend to generate average colors, typically producing muted, brown-tinted outputs [6].

Automatic image colorization is a relatively recent area of study, with most work occurring in the last few years. Early efforts into scribble-based colorization [9] require human interaction, while transfer-based colorization [2] requires a reference color image. The most recent work is fully automatic and typically relies on deep learning-based approaches to train a neural network [4]. These approaches tend to suffer from desaturated colors. Zhang et al. directly addresses the inherent multi-modality of the colorization problem by generating a probability distribution over a discretized color space [12]; while this approach can predict realistic colorizations (and can be regarded as the current state of the art in colorization), it can suffer from unrealistic color variations in some instances and is limited to a single output for a given grayscale input.

A recently developed type of neural network, the generative adversarial network (GAN) [5], can generate multiple outputs for a given input. This approach is used in the Pix2Pix model, which we adapt in this work, for image-to-image translation, including predicting what a daytime photo would look like if it were taken at night [7]. GAN models include a stochastic component to allow them to generate random samples from a learned distribution, e.g.,

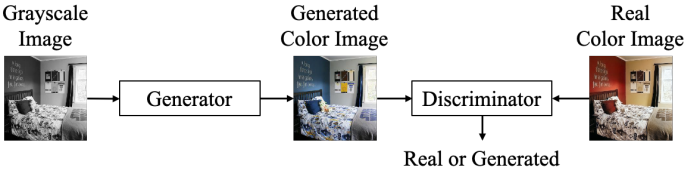


Figure 2: In our model, the generator network produces colorizations given grayscale input. The discriminator network predicts whether it has received real or generated input. The two networks are trained alternately so that they improve together over time.

the distribution of valid colorizations for a given grayscale image. However, in practice, GANs often suffer from mode collapse [10], where instead of sampling from across the range of a distribution, they are able to generate output only in a very limited range. For example, a GAN model designed to produce colorizations might always color balloons red, ignoring other modes in the color distribution for blue or other plausible colors.

Our proposed colorization model, based on a GAN architecture, is designed to address the issue of mode collapse by explicitly generating multiple colorizations for a given grayscale input, as shown in Figure 1. During the training process, each output learns to cover a different mode in the distribution of plausible colorizations. In the next section, we will describe the network architecture behind our model and the custom loss function that enables it to produce a variety of diverse outputs. In Section 3, we will show example colorizations produced using our method and discuss a user study designed to assess how well human observers are able to distinguish between generated and real color images. Section 4 concludes the paper and suggests directions for future work.

2 Method

As is common in other recent work on colorization [12, 1], we adopt the CIE LAB color space, which represents color images with three channels: L, the lightness, A, the green and red, and B, the blue and yellow colors in an image. LAB has the helpful property that the L channel is equivalent to the grayscale version of the color image. To train a model to produce colorizations, we use a set of N color images, $S = \{I_1, I_2, ..., I_N\}$, where, for each image, the L channel is the input and the A and B channels are the ground-truth, desired output. We represent the L channel as $\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$ and the A and B channels together as $\mathbf{Y} \in \mathbb{R}^{H \times W \times 2}$, where H and W are the image dimensions.

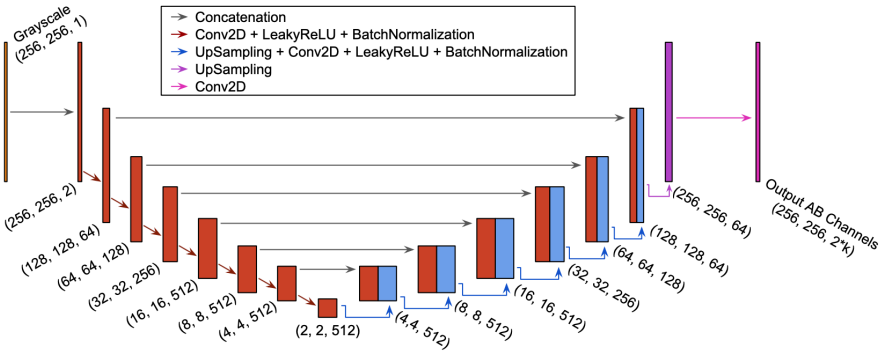


Figure 3: A grayscale image is input into the generator network, which produces K colorizations. The network is based on the Pix2Pix architecture [7], consisting of a series of downsampling and upsampling convolutional layers.

2.1 Model Architecture

Our proposed approach is based on a conditional generative adversarial network (CGAN), illustrated in Figure 2. A CGAN consists of two separate neural network models: a generator, G, and a discriminator, D. The generator learns a mapping $G : \mathbf{X} \rightarrow \widehat{\mathbf{Y}}$, where $\widehat{\mathbf{Y}}$ is a generated colorization. The goal of the generator is to fool the discriminator network, which takes as input either a generated colorization or a real color image and attempts to predict whether the input is real or fake. Training proceeds by alternating between the two models, with each model improving over time. Ultimately, after training, only the generator network is retained to produce future colorizations. The generator network architecture, based on Pix2Pix [7], is depicted in Figure 3. The discriminator architecture, not shown due to space constraints, is similar to the first 5 layers of the generator.

The objective of a CGAN can be expressed as follows [7]:

$$\mathcal{L}_{CGAN}(G, D) = \mathbb{E}_{X,Y}\left[\log D(X, Y)\right] + \mathbb{E}_{X}\left[\log(1 - D(X, G(X)))\right] \quad (1)$$

During the adversarial training process, the generator attempts to minimize this objective while the discriminator attempts alternately to maximize it over the set of training images, S, so that at convergence, the optimal generator is given by [7]

$$G^{*} = \text{argmin}_{G}\text{max}_{D}\mathcal{L}_{CGAN}(G, D) \quad (2)$$

Previous work [10] has found it advantageous to incorporate a second term in the network loss function to encourage the generator to produce outputs

similar to the ground truth input. For example, we could represent the summed absolute error (SAE) between the pixel values of a predicted colorization, $\widehat{\mathbf{Y}}$, and a known, ground-truth colorization, $\mathbf{Y}$, as

$$\mathcal{L}_{SAE}(\mathbf{Y}, \widehat{\mathbf{Y}}) = \sum_{h,w,c} \left| \mathbf{Y}_{h,w,c} - \widehat{\mathbf{Y}}_{h,w,c} \right| \tag{3}$$

where h, w, and c index the height, width, and color channel (A or B), respectively, of each pixel in the image.

For our method, rather than a single prediction, we would like to generate a diverse set of K plausible outputs. We adapt the hindsight loss from Chen and Koltun [3], which computes the SAE between the single output most similar to the ground truth:

$$\mathcal{L}_{H}(\mathbf{Y}, \widehat{\mathbf{Y}}) = \min_{k} \sum_{h,w,c} \left| \mathbf{Y}_{h,w,c,k} - \widehat{\mathbf{Y}}_{h,w,c,k} \right| \tag{4}$$

where k indexes over K outputs. The hindsight loss includes, for a given training example, only the output most similar to the ground truth in the loss calculation, encouraging specialization and diversity among the outputs. Adding this term into Equation 2 gives us our final objective function:

$$G_{opt} = \text{argmin}_G \text{max}_D \mathcal{L}_{CGAN}(G, D) + \mathcal{L}_{H}(G) \tag{5}$$

In the following section, we show a variety of colorizations produced by models trained to minimize this loss function.

3 Results

To evaluate our approach, we train two different colorization models, one for indoor and one for outdoor scenes, using two different datasets. The Large Scale Scene Understanding (LSUN) dataset contains images of a variety of scenes, from bridges to classrooms to dining rooms [11]. We select a subset of LSUN' s bedroom category to represent indoor scenes, using 50,000 images split into 49,000 for training and 1,000 for testing. The Places2 dataset contains an even broader variety of scenes with over 434 classes [13]. We sampled 5,000 images from each of the following categories: badlands, butte, canyon, cliff, field-wild, forest-broadleaf, forest-path, lake-natural, mountain, and tundra. From each category, we used 4,900 images for training and 100 for testing; resulting in 49,000 training and 1,000 test images. While images from the Places2 dataset all have a resolution of 256x256, LSUN images are a variety of sizes. Because the resolution of images input to our model must be consistent, the LSUN images are cropped around the center to ensure a size of 256x256.

The models are implemented using TensorFlow. (Code is available at `https://github.com/drew-bowman/Colorization-CGAN`.) Images are preprocessed by converting from RGB to LAB and normalizing values to $[-1:1]$. To prevent overfitting during training, data augmentation is performed with small random rotations and translations ($\pm 5\%$). To learn the optimal parameters for the two networks, we alternate training one step of gradient descent for the discriminator and the generator on each minibatch of 6 training examples using the ADAM optimizer with learning rate 2e-4 over 100 epochs. We set the hyperparameter, K, to be 5, observing empirically that producing 5 outputs results in a variety of diverse outputs, while higher values of K produce some very similar outputs. Training takes approximately 2 days on a Tesla K40 GPU, while generating K colorizations for a given input using a trained model takes 11.5 milliseconds on average.

Figure 4 shows several examples of colorizations generated by our outdoor and indoor models for gray input images taken from the test sets (i.e., images not used in training). For each example, the original color image is shown, along with the grayscale version used as input to the model. For comparison, we include a colorization produced by the method of Zhang et al. [12], the current state of the art in colorization, followed by four colorizations produced by our model. We observe that our colorizations exhibit both realistic colors and diversity. More diversity is evident in the indoor scenes than the outdoor scenes, which seems reasonable since there is greater diversity in the indoor scenes used for training.

While our approach tends to produce convincing colorizations, not all results are plausible. Figure 5 shows several representative failure cases. The first two examples include unrealistic artifacts that occasionally appear in generated images. The next two examples show images containing people which were not successfully colored by the model. The last example shows splotchy color changes that are uncharacteristic of natural images. Larger datasets of training images would help the model learn to overcome these issues.

3.1 User Evaluation

Because our goal is to produce a variety of plausible colorizations, rather than an exact replica of a particular input, it is difficult to quantify the results. We conducted a user study to measure how convincing generated colorizations appear to a human observer. We administered a survey consisting of 25 real color images and 25 colorized images asking participants to determine whether each image was real or artificially colored. Images were selected randomly from the test datasets for inclusion. For the generated images, we selected for inclusion the best scoring from among the K colorizations based on calibrated scores from our discriminator model. Among the randomly selected colorizations

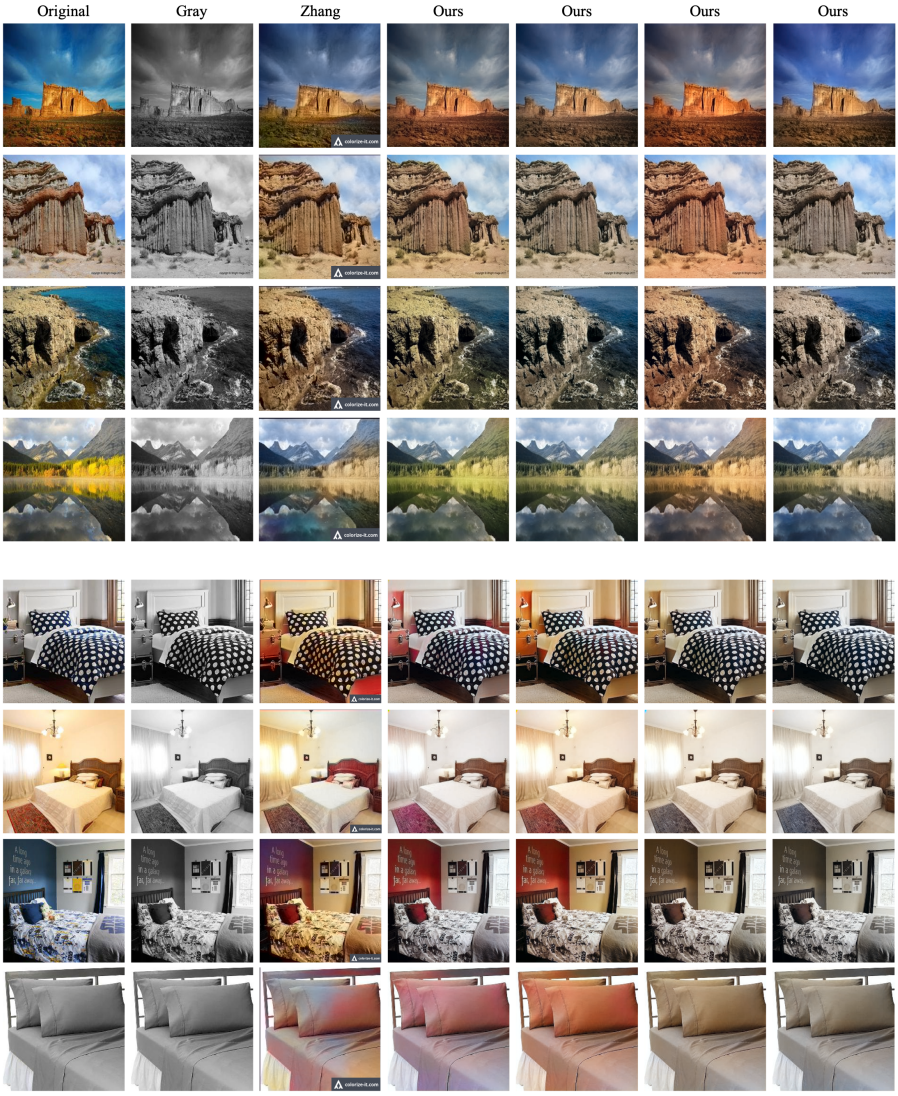


Figure 4: Each row shows an example color image, the grayscale version, the colorized version using the method of Zhang et al.[12], and multiple colorizations produced by our approach. The top four rows were colorized by a model trained on outdoor scenes, while the bottom four rows were colorized by a model trained on bedroom scenes.

Figure 5: Generated images occasionally contain unrealistic artifacts, unnatural colorings, or unlikely color splotches.

(a) 90.09% (b) 88.64% (c) 86.36% (d) 81.82% (e) 81.82%

(f) 88.64% (g) 70.05% (h) 63.64% (i) 52.27% (j) 40.91%

Figure 6: Images from the survey with the highest false positive (identifying a colorized image as real) and highest false negative (identifying a real image as colorized) rates are shown on the top and bottom rows, respectively.

were obvious artificial examples (e.g., the first image from Figure 5). This enabled us to perform a quality control step by discarding survey responses where these obvious fake colorizations were selected as real color images by a respondent. We collected 54 survey responses from undergraduate computer science students and discarded 9 based on this criterion.

Overall, respondents identified colorized images as real 52% of the time, suggesting that colorizations were convincing enough to reduce selection to random guessing. Supporting this finding, real images were identified as artificial 30% of the time. Figure 6 shows the images that were most often classified incorrectly. It appears that respondents suspected that more colorful real images were likely to be artificial, while more muted colorized images were likely to be real.

We also apply our model, trained on outdoor images, to historical pho-

(a) *Shadow Lake*, 1871 (b) *Sierra San Juan*, 1874 (c) *Cameron's Cone*, 1879

Figure 7: Example colorizations produced by our approach given input black-and-white photographs taken by Henry William Jackson in the 19^{th} century [8].

tographs taken in the 19th century by Henry William Jackson [8]. Figure 7 shows several examples of the original black-and-white photos and a selected colorization generated by our model. As is evident, the colorizations are realistic, capturing plausible colors for these nature scenes.

4 Conclusions and Future Work

This paper presents an approach to automatic image colorization using a generative adversarial network. We incorporate a loss term in the learning process to encourage the model to produce diverse colorizations. Our approach overcomes a weakness of some prior work to generate unrealistic, "average" colorizations as well as a common issue with GANs, mode collapse, by explicitly generating a variety of different colorizations. Experiments with datasets of indoor and outdoor photographs demonstrate qualitatively that our method produces varied realistic colorizations, and the results of a user study support these findings. We apply our trained outdoor model to historical black-and-white nature photographs and show realistic colorizations. Our models, created with a machine learning approach, are limited by the datasets used to train them. For the future, we would like to extend the scope of the datasets to improve the ability of a trained model to colorize more diverse scenes. We are also planning to explore other machine learning approaches, such as mixture density networks, that allow for more explicit learning of the parameters of the color distribution within an image.

References

[1] Yun Cao, Zhiming Zhou, Weinan Zhang, and Yong Yu. Unsupervised diverse colorization via generative adversarial networks. In *Joint European conference on machine learning and knowledge discovery in databases*, 2017.

[2] Guillaume Charpiat, Matthias Hofmann, and Bernhard Schölkopf. Automatic image colorization via multimodal predictions. In *European conference on computer vision*, pages 126–139, 2008.

[3] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *International conference on computer vision*, pages 1511–1520, 2017.

[4] Zezhou Cheng, Qingxiong Yang, and Bin Sheng. Deep colorization. In *International Conference on Computer Vision*, pages 415–423, 2015.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *International Conference on Neural Information Processing Systems*, 2014.

[6] Jeff Hwang and You Zhou. Image colorization with deep convolutional neural networks. Technical report, Stanford University, 2016.

[7] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[8] William Henry Jackson. William henry jackson collection. `https://www.nps.gov/scbl/learn/historyculture/william-henry-jackson.htm`.

[9] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Trans. Graph.*, 23(3):689–694, August 2004.

[10] Qi Mao, Hsin-Ying Lee, Hung-Yu Tseng, Siwei Ma, and Ming-Hsuan Yang. Mode seeking generative adversarial networks for diverse image synthesis. In *Conference on Computer Vision and Pattern Recognition*, 2019.

[11] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015.

[12] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, 2016.

[13] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

Teaching AI Ethics in a Flipped Classroom*

Greg Taylor[1] and Debzani Deb[2]
[1]Department of Management, Marketing and MIS
[2]Department of Computer Science
Winston-Salem State University
Winston Salem, NC 27110
{taylorg,debd}@wssu.edu

Abstract

A flipped classroom approach can solidify AI ethics lessons in a few sessions. The approach described here introduces the Montreal Declaration of Responsible AI Development then asks students to apply it to a few case studies. Students post threads and responses to an on-line discussion board prior to a class session where student groups explore the cases in depth. Feedback and grades encourage high student engagement. Instructors could integrate similar AI ethics modules into any class where students have a minimal conceptual understanding of machine learning or AI. The learning objectives do not depend on the cases selected so new articles would be used over time to ensure student engagement. Instructors can easily modify the approach for use in an on-line setting.

1 Introduction

Artificial Intelligence (AI) and Machine Learning (ML) based technologies play a crucial role in how we work, learn, communicate, and participate in society. As with many major scientific and technological breakthroughs, the use of AI and ML techniques has profound social and ethical implications. AI and ML technologies may reinforce racial and gender biases, perpetuate economic

inequality, and violate privacy rights. These systems can target user beliefs and psychological traits with minimal transparency and accountability while eroding societal trust. Many academic, government, and industry efforts to develop principles for developing ethical AI have stressed the need for education among programmers, users and managers [8]. We use one broad-based effort, the Montreal Declaration for a Responsible Development of AI, as a framework for ethical analysis in the course modules described here [1].

Some universities have incorporated ethics contents into their AI/ML curricula specifically targeted toward the CS/CE majors [8]. While ethical training is essential for future developers of AI-enabled products, it is equally important for general practitioners and users of such systems. As educators, we need to weave such social and ethical considerations across all majors in all disciplines. We want to make sure these future practitioners can explore the ways AI/ML technology can have an impact on business stakeholders and their communities.

Instructors occasionally embed ethics in more technical curricula [3] [10] [6] [7], however, these courses are often not available to all university students. Another approach is to embed AI ethical issues into discipline-specific courses, ideally those targeting large groups of students or all students of a specific major. This ensures the ethical training is relevant but often requires training instructors in AI and ML. We used the latter approach. An instructor embedded an ethics module into a Business Analytics course required for all business-related majors. The approach could easily be adapted into many other disciplines that teach modeling or computing techniques.

Incorporating AI/ML related ethical training using this approach involves 1) teaching students some necessary ethical guidelines in a non-technical way and 2) providing them the opportunity to apply those guidelines to an AI-enabled situation, identify ethical issues, and assess potential trade-offs and solutions. In this paper, we present a novel modular approach for teaching societal and ethical implications of AI systems to non-majors with very limited prior technical and programming experiences. Ultimately, the goal is for students to recognize and describe ethical issues in real-world AI-based systems affecting their daily lives.

The AI ethics module presented in Section 2 introduces students to a principles-based framework developed in Dec 2018 by a group affiliated with the University of Montreal [1]. This framework describes 10 principles for responsible AI development. Students apply the ten principles to assess ethical problems with an AI/ML case-study in a flipped-classroom format [9] that facilitates collaborative learning. Students create and respond to threads in an on-line discussion forum before working in classroom-based groups to make short presentations on these topics. The instructor built two such case-studies by posting engaging questions in the forum and providing opportunities for

learners to formulate answers (or sometimes to come up with inquisitive questions) independently and collectively.

Our empirical study is based on student performance and self-reflection survey data. They reveal that the on-line discussions act as catalysts to productive in-class conversations, persuasive arguments, and diverse viewpoints. These increase student engagement and academic performance. Our goal is to create a repository of case studies along with engaging questions and discussion activities.

The flipped-classroom approach could easily be adapted to an entirely virtual environment. Therefore, we anticipate that both the flipped-classroom modular approach and the repository of case studies will be useful in both traditional and virtual classrooms.

2 AI Ethics Module

We describe a modular, flipped-classroom approach to embedding AI Ethics instruction into a Business Analytics course in this section. The approach is well suited for courses focusing on modeling, computing or ethical issues in any discipline and is easily adapted to online learning. In the following subsections, we present learning outcomes, the flipped-classroom approach, and example case studies.

2.1 Learning Outcomes

We hope that students will be able to apply AI ethics frameworks to novel machine learning applications they encounter in their careers. Such frameworks will help them analyze, describe and suggest remedies for potential ethical violations. The specific learning outcomes for the presented module are

1. Learn about the Montreal Declaration for Responsible AI guidelines and apply the guidelines to recognize and describe ethical issues in AI-based systems. (LO1).

2. Discuss and reason, both alone and in collaboration with others, about the violation of the guidelines in an AI-enabled case-study and potential solutions of these violations. (LO2).

3. Gain enhanced awareness of approaches to minimize ethical problems that can arise in the development and implementation of AI-based systems. (LO3).

2.2 Flipped Classroom Approach

We followed a "flipped classroom" approach to teach the AI/ML ethics module. As an initial exercise, students read articles summarizing the Montreal Declaration for the Responsible Development of AI and efforts to build AI ethics boards. The module then utilizes two 75 minutes class sessions devoted to applying the framework to two AI-based applications (case study articles). Optionally, instructors can add more real-world case studies to explore other AI ethics issues with the Montreal framework.

Before each in-class ethics session, students contribute to an on-line discussion forum where they apply their knowledge of the ethical framework to assess a case study article describing an AI/ML application. Instructors assign students into one of two groups: each student in the first group creates a discussion thread and briefly describes and critically assesses the article based on engaging questions posed by the instructor. They also submit three questions or suggestions for in-class discussion. Each student in the second group then has one extra day to reply to one of the discussion threads created by a student in the first group. The instructor can reverse the roles of the two groups in subsequent case studies so each group has a chance to both assess the articles and respond to other student threads. At this phase, students worked independently.

The instructor then used the student-posed questions along with others to develop group discussion topics for an in-class session. Students split into groups of approximately five students. The instructor assigned each group a topic with a set of questions to orally present before the end of class. The groups had 15-20 minutes to formulate responses before presenting. Our classes were small enough to expect each student to contribute orally to the presentation and to respond to additional questions. Instructors could scale this for large sections by using parallel sessions supported by teaching assistants (TAs) and multiple presentation spaces. Instructors can encourage participation by grading each student' s discussion board response and oral contribution.

2.3 Case Studies

We utilized a case-study based approach to teach students about violations of ethical principles and possible solutions. In our modular intervention, we used two case studies based on popular newspaper articles. In this section, we detail our case studies as a template for use as is, or as inspiration for inclusion of other such case studies into the module. The first case study used a 2019 NY Times news article [2] detailing the facial recognition and tracking systems used by the Chinese government to monitor minority Uighur communities in Kashgar, China.

Half of the students described the monitoring system from the article, assessed it with the principles from the Montreal Declaration and posed three questions/topics for class discussion. Remaining students "added value" to their threads with answers to questions and/or expressing alternate viewpoints. Most students believed the Kashgar system violated all ten principles of the Montreal Declaration. A few questioned the neutrality of the article and wanted to read other viewpoints on this system. One important benefit of exposing students to these question/response activities is to allow them to think critically and independently about source information, ethical issues and possible solutions.

During the in-class discussion applying the Montreal Declaration principles to the Kashgar monitoring system, the instructor assigned a few principles for each group to apply to the AI-based monitoring system. Student participation was graded on critical analysis. Grades for individuals occasionally varied from the group.

A second case study given about four weeks after introducing the Montreal Declaration explored the ethical issues surrounding labeling and categorizing images. Students assessed Crawford and Paglen' s claim that labeling and categorizing is inherently political [4]. The article cited ethically problematic examples from the ImageNet datasets [5]. Subsequently, half of the labels and categories in the dataset have been deleted.

The roles of the students were reversed from the first exercise above –half created threads assessing the article and listed three topics for discussion while the other half responded to those threads. The instructor did not directly prompt students to use the Montreal Declaration for analysis. Unfortunately, most students did not choose to use the Montreal Declaration as a framework for discussing the ethical issues posed by the new case study. With a bit of instructor prodding during the in-class group presentations, it dawned on a few students to use the Declaration to describe ethical issues. At this point, many saw the benefit of the using the principles to inform their analyses.

3 Experimental Evaluation

3.1 Results and Discussion

To evaluate the potential usefulness of the flipped classroom approach to learning AI ethics, we conducted a pilot study. We hypothesized students in the Business Analytics class could identify, describe and respond to ethical issues in AI/ML enabled systems. To test this hypothesis, we assessed the three learning outcomes (LO1, LO2, LO3) outlined in section 2.1 with student performance data. We also utilized an anonymous student experience survey (IRB approved) conducted at the end of the semester.

To evaluate LO1 (competency on the declaration principles and their applications), we used the student performance data on three ethics related multiple choice questions (MCQ) utilized in the final course exam. Twenty-five students enrolled but one student did not do any work. We eliminated that student from the statistics. Summarized responses from three exam questions are listed in Table 1. The results show the majority students (83%) were able to grasp the Montreal Declaration for AI guidelines (Q1.) More than half (54%) of the students were able to identify the violation of declaration principles in Amazon' s automated firing of warehouse workers, a scenario not discussed in class (Q2.) Unfortunately, only 29% were able to correctly identify the three issues requiring removal of half of labels in the ImageNet dataset (Q3.) While 3 of the top 5 exam performers got the question correct, it was negatively worded (choose the non-issue) and the correct answer (replace human-generated labels with machine-generated) was not something discussed in class. The instructor should reword or replace Q3 in the future. Excluding Q3, about 68 percent of the students were able to correctly apply the ethical guidelines to recognize and describe ethical issues in AI-based systems (LO1). This result is consistent with observations during in-class labeling case study. Without prompting, students had difficulty applying the Montreal principles to a novel scenario. Perhaps additional case studies would improve future performance.

Table 1: MCQ Assessment in Final Exam

MCQ Question Learning Objective	Correct Responses (%)
Identify ethical principles from Montreal Declaration (Q1)	20/24 (83%)
Apply ethical principles to a new situation (Amazon' s automated firing of warehouse workers.) (Q2)	13/24 (54%)
Identify issues requiring removal of half of labels in the ImageNet dataset. (Q3)	7/24 (29%)

We assessed the second learning objective, LO2 (ability to discuss ethical issues in person and in collaboration with others,) by utilizing two on-line discussion forum assignments and two 75-minute in-class group discussions devoted to discussing ethical issues related to the two case studies discussed earlier. Figure 1 shows the student performance data for the rubrics (A-F) described in section 2.3. "N" refers to students who did not submit or were not present. The surveillance system used in Kashgar, China case study is referenced as CS1. The image labeling and categorization procedures used for AI training sets case study is CS2. Most (76-84 percent) of the 24 students posted on the

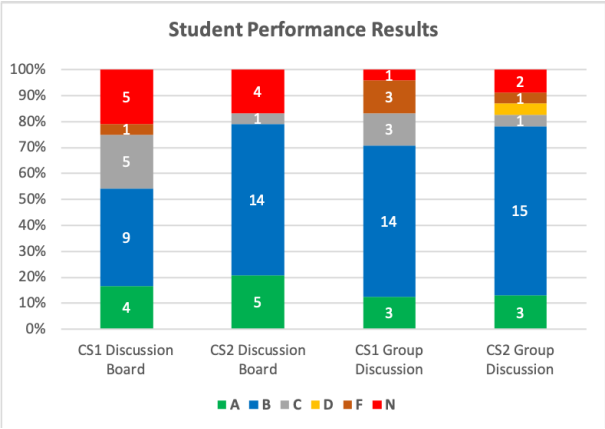


Figure 1: Discussion board and in-class group discussion results.

discussion boards prior to the corresponding in-class group discussion sessions. Higher percentages (84-92%) of students participated in group discussions and presented responses to questions during class.

Overall, 75-83 percent of the students earned C grades or better on the discussion board and in class on the two case studies. There was a small improvement in engagement on the second case study compared to the first. The grades on the two case studies made up 6 percent of the student' s final grade in the course. Those who did not participate in either case study received a partial letter course grade less than they otherwise would have. The instructor may have improved engagement with this policy.

The grades of the engaged students also improved on the second study. Some may have been disappointed with "C" grades on the first case and recalibrated their expectations on the required work. Perhaps students learned to apply the ethical principles better after the first case. Overall, most students were engaged both in the on-line discussion forums and the in-class group presentations. Based on these assessments, most students met LO2.

We assessed LO3 (awareness of ethical issues in AI/ML systems) using two self-satisfaction end-of-course survey questions. Students responded using a 5-point Likert scale. Results are shown in Figure 2.

- Q1: I understand how the data science topics covered in this course could be utilized for societal good.
- Q2. I can discuss ethical issues surrounding the use of artificial intelligence in a professional setting.

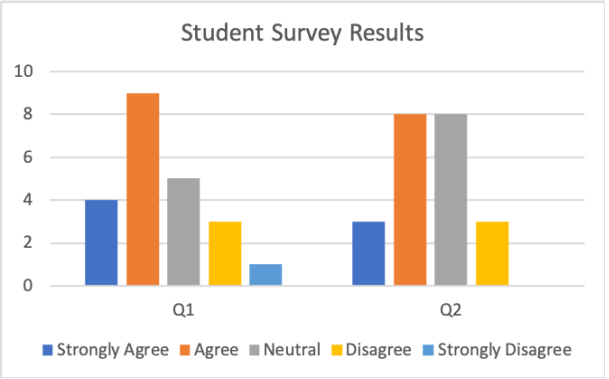


Figure 2: Student self-satisfaction survey results.

Of 22 questionnaire responses, only about half were confident about their ability to discuss AI ethical issues in a professional setting. A few more thought they understood how AI could be used for societal good. Fortunately, only a few disagreed with these statements. These questions produced more positive student feedback than 7 of the other 8 questions on the same survey. Most data science topics covered in the course involved statistical programming and the use of machine learning software. Based responses to the remaining 8 survey questions, we concluded that students are more confident about their knowledge of ethics than their ability to use software. Additional case studies might help more students achieve LO3.

3.2 Instructor Perspective

Based on these two sessions, the instructor thought the "A" students had a firm grip on the three learning objectives. They would be able to utilize guidelines in a professional setting to minimize ethical problems that could arise in the development and implementation of proposed AI systems. All students realized AI systems can have ethical problems.

Applying the Montreal Declaration to labeling and categorization without prompting was difficult for students after a month had passed since introducing the topic. Still, the goal is for students to recognize ethical problems with AI systems and use the Declaration to help describe these issues. Achieving this objective would have required at least one more ethical case study for these students.

4 Conclusions and Discussion

The flipped classroom used in this study requires students to participate in an on-line discussion board before attending an in-class session. It is an effective way to learn AI ethics. This approach allows students to assess AI-based systems with a set of ethical principles in as few as two in-class sessions. Once an ethical framework such as the Montreal Declaration for Responsible AI Development is introduced, applications from the popular media or academic literature may be assessed.

New applications appear frequently in these outlets and instructors can substitute them for the Uighur monitoring and labeling cases described above. One could easily construct cases to achieve the same learning objectives from descriptions of self-driving cars, social media monitoring, and other topics frequently explored in the popular press. Changing the cases from year to year limits opportunities for students to inappropriately use the work of others. Students must engage and add value to the discussion boards and in-class presentations to achieve the learning outcomes and receive positive instructor feedback.

The flipped classroom approach can facilitate discussions of AI ethics into any course. Students needed minimal prior knowledge to analyze the Kashgar case study. However, students need a conceptual understanding of machine learning before exploring some ethical issues. Appreciating the ethical issues involved with ImageNet requires some background in training machines, finding examples, labeling, modeling and predicting. The ImageNet labeling case can reinforce student ' s conceptual understanding of machine learning, especially in non-technical courses. Instructors could easily develop similar cases to reinforce other AI concepts.

The flipped classroom can be easily adapted to on-line learning environments. On-line instructors could replicate the in-class group discussions and presentations with the help of a synchronous conferencing tool. Chat facilities with breakout rooms can substitute for in-class group work. A moderator could replace in-class group presentations with chat responses or video presentations. In an asynchronous on-line environment, student groups could produce a written response to assigned topics on a discussion forum or similar venue.

5 Acknowledgements

This research was supported by National Science Foundation Award 1600864. The grant provides encouragement and training to embed data science topics and courses into curricula.

References

[1] Montreal declaration for a responsible development of artificial intelligence, 2018. https://www.montrealdeclaration-responsibleai.com/reports-of-montreal-declaration.

[2] Chris Buckley, Paul Mozur, and Austin Ramzy. How china turned a city into a prison. *The New York Times*. https://www.nytimes.com/interactive/2019/04/04/world/asia/xinjiang-china-surveillance-prison.html.

[3] E Burton, J Goldsmith, S Koenig, B Kuipers, N Mattei, and T Walsh. Ethical considerations in artificial intelligence courses. *AI Magazine*, 38(2):22–34, 2017.

[4] Kate Crawford and Trevor Paglen. Excavating AI: The politics of images in machine learning training sets, 2019. https://www.excavating.ai/.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. 2009. http://www.image-net.org.

[6] Heidi Furey and Fred Martin. AI education matters: A modular approach to AI ethics education. *AI Matters*, (4):13–15, 2019. https://doi.org/10.1145/3299758.3299764.

[7] Barbara J. Grosz, David Gray Grant, Kate Vredenburgh, Jeff Behrends, Lily Hu, Alison Simmons, and Jim Waldo. Embedded EthiCS: integrating ethics across CS education. *Communications of the ACM*, 62(8):54–61.

[8] Zheyuan Ryan Shi, Claire Wang, and Fei Fang. Artificial intelligence for social good: A survey. *arXiv*, 2020.

[9] Michael L. Wallace, Joshua D. Walker, Anne M. Braseby, and Michael S. Sweet. "now, what happens during class?" using team-based learning to optimize the role of expertise within the flipped classroom. *Journal on Excellence in College Teaching*, 25(3):253–273, 2014. http://152.12.30.4:2048/login?url=https://search.proquest.com/docview/1651854490?accountid=15070.

[10] Tom Williams and Qin Zhu. An experimental ethics approach to robot ethics education. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. http://par.nsf.gov/biblio/10125972.

The Japanese Fifth Generation Computing Project: Curricular Applications*

J. Paul Myers, Jr.
Department of Computer Science
Trinity University
San Antonio, TX 78212
pmyers@trinity.edu

Abstract

A particularly fascinating, but now largely forgotten, episode in computing history was the Japanese Fifth Generation Computing Project in parallel knowledge-based AI, based on logic programming (LP). Incorporating this Project into the curriculum (for even just a lecture or two, or a supplemental reading, in a variety of courses) might motivate a more lengthy and detailed technical discussion of LP. The logic paradigm, while out of fashion in the current machine-learning AI environment, still retains potential as an AI tool. Here we hope in part to illustrate LP' s techniques and to motivate its inclusion in the CS curriculum.

1 Historical Introduction

In the decade, 1983-1993, Japan was actively pursuing their Fifth Generation Computing Project (alternatively known as the Fifth Generation Computer Systems Project, and abbreviated herein as 5Gen). For a variety of reasons in technology history, this concerted effort of industry, academia, and government, resulted in considerable anxiety in the U.S. and the West [10]. Briefly, the project was to produce a pervasive artificial intelligence that, while containing

a hardware component emphasizing parallelism, was to be primarily software-based. This itself was striking; the previous four "generations" were hardware-based (vacuum tube through VLSI microprocessors).

5Gen, however, relied on inference processing using a relatively new computing paradigm known as logic programming (LP: symbolic logic as an approach to computation) and its then exemplar language, Prolog (PROgrammable LOGic). And the intent was for a high-performance personal Prolog machine with a "beautiful appearance" [16]; the initial vision was that the user would communicate intelligently with the computer using natural conversational language and images. That is, to some extent this vision in the 1980's contained the seed of what much later became the smart phone.

The West was caught by surprise with the announcement of 5Gen with its seeming promise of Japanese domination of yet another industry: computing. Artificial Intelligence experts Edward Feigenbaum (Turing Award recipient) and Pamela McCorduck wrote in their 1983 book, *The Fifth Generation: AI and Japan's Computer Challenge to the World*, "We are writing this book because we are worried" [4]. At about the same time, the September 1983 *Communications of the ACM*, the flagship publication of the America-based leading international organization for computing, published a cover (Figure 1) further indicating the sense of threat from the Japanese initiative [2].

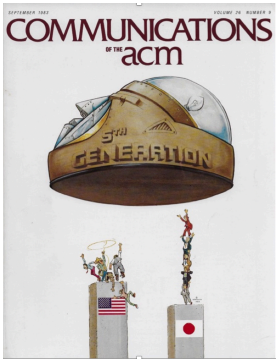


Figure 1: *CACM* Cover: 09/83

The West responded in several ways, notably the creation of the MCC (Microelectronics and Computer Technology Corporation) in the U.S. For additional historical details, see [10]. Suffice it to say here that a cursory evaluation of 5Gen after its decade of intense work is that it "failed"; that is, something akin to a pocket computer or smart phone did not emerge by the mid-1990's. That this evaluation is based on incomplete assumptions of the purpose of 5Gen and its various achievements is beyond the scope of this discussion; any final assessment is complex (see [17]).

2 Applications

Prolog is most closely associated with the applications area of artificial intelligence, but currently, machine learning (ML) is dominating AI. Prolog represents a symbolic approach to AI, utilizing logic; whereas ML represents a statistical approach, utilizing neural nets, data mining, etc.

There are advantages to each approach. A symbolic approach lends itself to formal verification, and the explicitly stated rules can be checked as to completeness. Statistical approaches become relevant when rules are too complex for explicit expression (e.g., learning facial recognition). In such cases statistical and big data approaches are most suitable.

And there are disadvantages to each approach. As mentioned, in complex scenarios rules may be too complex to list explicitly; and then statistical approaches, based on training and validation, allow the machine to derive its own rules. But then we don' t have explicit access to those rules –rules that might allow for mis-classifications that are hard to remedy.

Indeed, as pointed out in [10], Vardi [15] and others believe that philosophically a robust approach to AI will incorporate both "lizard brain" (quick/fast: machine learning) and "fore-brain" (slow/logical reasoning for decisions). So, logic may emerge within the main-stream again, giving reason enough to introduce the LP paradigm.

In any event, currently, as a symbolic/logic-based approach, Prolog is well suited for and is utilized in a number of diverse applications. While Prolog is not exactly booming, it still occupies a substantial role in applications within business, law, medicine, AI, etc. [13]: rule-based expert systems and training, theorem provers, intelligent search, semantic web development, natural language processing, decision support systems, semantic nets, combinatorial and optimization tasks, planning, scheduling, and cryptography.

2.1 Curricular Application I: History

Whether 5Gen is seen as a failure or success (too coarse of a dichotomy), because of the farsighted nature of the Project, the intensity of reactions to it, and the current dominance of AI, 5Gen should not fade into obscurity. The historical aspects of 5Gen are fascinating; some have posited that it led to an "AI Spring," ending a sustained "AI Winter" [10]. In any course/seminar involving the history of computing or AI, 5Gen is a most apt topic for lectures, discussions, modules, or supplemental readings. And in other related classes, a brief acknowledgement of 5Gen would certainly enrich the students' appreciation of the topic' s rich history and the importance of logic [8].

The relevance of 5Gen historically is the topic of [10]; its use in a computing history course was discussed in [9]. To be developed further is an account of a History/Ethics seminar class led in Spring 2020 where some student-selected covered topics were old or ancient (abacus, Euclid, Babbage/Lovelace etc.); but most were much more recent (e.g., 5Gen).

But we now turn to our primary focus here: illustrating, and hopefully motivating, the incorporation of the logic programming (LP) methodology in the CS curriculum.

2.2 Curricular Application II: The LP Paradigm

The technical 5Gen aspects of logic programming, resolution, unification, knowledge representation, etc. are well suited to be included in any course that emphasizes symbolic logic, AI, knowledge bases, etc. Such course inclusion would provide an alternative or supplement to neural nets and machine learning for AI: namely, the application of logic. The following discussion is oriented toward a technical presentation within a course or module in a computing curriculum. These are often challenging topics for students; nonetheless they are often required for an undergraduate CS major (e.g., in a Discrete Structures course). A full special-topics course can be (and has been) developed for undergraduates using the LP paradigm. And, given the importance of logic for CS, a course on Logic & Computing [14] would be appropriate and could easily incorporate LP. Shorter modules would be appropriate for courses in (see Appendix 3): Discrete Structures (the logic component); AI (e.g., reasoning and knowledge representation); Database (alternate and historical approaches); Logic & Computing (various forms of logic); and Programming Languages (the LP paradigm).

Because these curricular uses are focused on more technical issues, we will forthwith assume that the reader is familiar with basic concepts and notation for symbolic/mathematical logic (e.g., as typically presented in a Discrete Structures course).

LP is a separate programming paradigm, distinct from procedural, functional, and object-oriented. That in LP, the knowledge base = the program is a truism! Regarding standard relational databases, the leader of 5Gen, Kazuo Fuchi wrote [5], "At present it is common that databases and programming languages belong to different systems. This is not a desirable situation. Their unification appears to be quite feasible." That the program = the data is the ideal. Dahl [3] notes that "the logic program serving to define the data serves at the same time to compute it. ···The user need only be concerned with the declarative semantics of [the] database." This is demonstrated in examples below, with a trivial but semantically correct knowledge base (KB), or see any standard Prolog textbook (e.g., [1]).

The key to fulfilling the above ideals was the fact that the large and cumbersome apparatus of standard logical systems (axiomatic, natural deduction, ···) can be replaced with just one rule of inference: *resolution*. However, resolution necessitates listing all of a KB' s statements and relationships in a specified format called *clausal form*. With standard first-order predicate logic (FOPL, using variables and quantification), these clauses constitute a normal (standard, universal) form in that there is a straight-forward algorithm to convert any arbitrary statement of FOPL into a logically equivalent set of clauses. These clauses may be further refined into more intuitive and useful

Horn clauses (a stricter version of clausal form with only one consequent):

Clause:

$$A_1 \wedge A_2 \wedge ... \wedge A_m \rightarrow B_1 \vee B_2 \vee ... \vee B_n$$

Horn Clauses:

$$A_1 \wedge A_2 \wedge ... \wedge A_m \rightarrow \text{B}$$

$\rightarrow$ A [*assertion* of A]

A $\rightarrow$ [*denial* of A: $\neg$ A (not A)]

$\rightarrow$ [(or $\square$) *null clause* (falsehood)]

Variables in FOPL are, of course, quantified universally ($\forall$) or existentially ($\exists$); for example: over the integers: $\forall x \exists y(x + y = 0)$. Here y is within the *scope* of x and so its choice is dependent on the choice for x (y = -x). However, while $\exists y \forall x(x + y = 0)$ is false, this is true: $\exists y \forall x(x \times y = 0)$: (y = 0).

Interestingly, all clauses are universally quantified (happily obviating the need for resolution to deal with quantification at all). And fortuitously there is a particularly elegant procedure to eliminate existential quantification through the use of *Skolemization* where, for each statement, quantified variables become, depending on their scope, either constant symbols or functions of the relevant universal variables (see, e.g., [6] or [7]; for a more theoretical approach: [12]. In our example statements above:

(i) $\forall x \exists y(x + y = 0)$; the choice of y is dependent upon x. Thus, we have $\forall x(x + f(x) = 0)$ [of course, here the relevant function is f(x) = -x].

(ii) $\exists y \forall x(x \times y = 0)$; the choice of y is independent of the choice for x; so that y becomes a constant c, regardless of x. Thus, we have $\forall x(x \times c = 0)$ [of course, y=0 is the relevant constant here].

Thus, e.g., a more complicated statement $\exists x \forall (y, z) \exists t P(x, y, z, t)$ is Skolemized into $P(c, y, z, f(y, z))$:

–x exists globally, so it is replaced by a *constant symbol* c;
–y, z are universally quantified, so they remain;
–t exists locally (dependent on y and z since it is within their scope); so, replace t with *function symbol* f showing the dependency on y and z;
–now the remaining (*free*) variables (y & z) are taken to be universally quantified; so "$\forall$" , being implicitly there, may be disregarded.

In Appendix 1, the algorithmic conversion from FOPL to clausal form is demonstrated on a representative FOPL statement. However, the even more restrictive Horn clauses are a very natural way of asserting knowledge

and relationships; hence, the conversion algorithm, from arbitrary FOPL statements, rarely needs to be applied. Its primary purpose is to show that Horn clause resolution has robust expressive power and is Turing-complete.

As above, the knowledge base (KB) *is* the program; the KB has just that one rule of inference: *resolution* –a generalized *modus ponens* (→-*elimination* or *detachment* in natural deduction). Resolution is available at all times to work within the KB; hence the system is called a KIPS (knowledge information processing system); this fact makes a KIPS amenable to intense parallelization. Resolution operates by a kind of cancelling out of terms on the RHS of a clause with identical terms on the LHS of another clause, followed by a merging of those terms remaining. Propositional logic example, resolving on B:

Resolution [RES]:	*Modus Ponens* [MP]:
$A \wedge \cancel{B} \wedge C \wedge D \rightarrow S$	$\cancel{B} \rightarrow \mathrm{S}$
$C \wedge E \rightarrow \cancel{B}$	$\rightarrow \cancel{B}$
$\therefore \quad A \wedge \ C \wedge D \wedge E \rightarrow S$	$\therefore \ \rightarrow S$

This is easy to understand intuitively. Indeed, $A \wedge B \wedge C \wedge D \rightarrow S$ indicates that all of the A,B,C,D need to be satisfied for S to be derived. But now $C \wedge E \rightarrow B$ indicates that C,E are sufficient for B to be satisfied. So, we may eliminate/resolve B in the first clause by replacing it with the sufficient $C \wedge E$ (removing a now redundant C). And, of course, this is most clear in modus ponens where B is an asserted statement (no sufficient conditions).

So now, assuming that we have a concrete knowledge base (KB) of relevant Horn clauses, the computational strategy is to append the *denial* of the desired outcome A to the KB: $(A \rightarrow)$ as the *goal clause*. Thus, if the KB also included the "fact" $(\rightarrow B)$ and the implication/relationship $(B \rightarrow A)$, then

1.	$\rightarrow$ B	[in the KB]
2.	B $\rightarrow$ A	[in the KB]
3.	A $\rightarrow$	[goal clause (denial of A)] - appended to the KB
4.	B $\rightarrow$	[RES: 2,3] - resolution application on statements 2 & 3
5.	$\rightarrow$	[RES: 1,4] - the null clause $\rightarrow$ (often abbreviated as $\square$)

So, the augmented (with the goal clause) KB is shown to be inconsistent (it will allow derivation of the falsehood $\square$). So, negated A (i.e., $A \rightarrow$) is disproved, resulting in (standard notation): $\neg(\neg A)$. And, as is the case in indirect proofs, $\neg\neg A \equiv A$, so that A is proved.

Most interestingly, this one rule of inference, RES, replaces the entire suite

of the familiar Gentzen natural deduction rules: ten introduction/elimination rules of inference; see Appendix 2 here (for propositional logic) [11].

The computational simplification is obvious: one rule to replace ten! However, there are still complications aplenty (there is no free lunch); in order to resolve (cancel) two terms they must be identical. Hence variables must be unified (made the same): instantiated with identical constants and functions. There are *unification* algorithms to accomplish this, probably the single trickiest aspect of applying RES; their use can be computationally intensive. For technical details on this and also on how a LP-Resolution program (= KB) performs a non-Boolean (numeric) computation, see [1] or [6]. As a simple, example employing straightforward unification, consider the following knowledge base K (recall that all variables are universally quantified; so there is no need to attend to matters of scope). Each step adds a new statement to K:

1. parent(x,y) ∧ parent(y,z) → grandparent(x,z)
2. father(x,y) → parent(x,y)
3. → parent(Tom, Jane)
4. → parent(Jane, Ann)

5. grandparent (t, Ann) → [*goal clause*: append to find who Ann' s grandparent is, we deny that Ann even has a grandparent! (t, like all variables, is universally quantified)]

6. parent(x,y) ∧ parent(y, Ann) → grandparent(x, Ann) [unify 1 (with 5); z := Ann]
7. grandparent (x, Ann) → [unify 5 (with 6); t := x]
8. parent(x,y) ∧ parent(y, Ann) → [RES: 6 & 7]
9. parent(x, Jane) ∧ parent(Jane, Ann) → [unify 8 (with 4); y := Jane]
10. parent (x, Jane) → [RES: 4 & 9]
11. parent (Tom, Jane) → [unify 10 (with 3); x := Tom].
12. → (or □: null clause) [RES: 3 & 11].

Thus, not only does Jane have a grandparent (the indirect proof), but by tracing the variable t back to x and then to Tom, we also know who her grandparent is (Tom). Suffice it to say that while the LP resolution-driven computations are *indirect* proofs (*reductio ad absurdum*, as above), they are nonetheless *constructive* since the concrete answer (calculated value) results from backtracking the instantiations provided by unification.

While the above provides a reasonable look and feel for a resolution computation, there are issues that do sometimes arise and must be attended to: renaming of variables, order of clauses, ∞-loops, and so on [1, 12]. These are

too intricate for our brief purpose here.

Acknowledgements

I am indebted to many. I especially acknowledge Dr. Katsuo Nishikawa who arranged for my first trips to Japan; Dr. Yoshihiro Omura UTF8min (小村吉弘先生) who hosted me at Kindai University UTF8min (近畿大学) ; and Dr. Mario Gonzalez for his continual encouragement. Trinity University awarded me a Summer Stipend for this project.

References

[1] W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, New York, 1981.

[2] Cover. *Communications of the ACM*, 26(9), 1983.

[3] V. Dahl. On database systems development through logic. *ACM Transactions on Database Systems*, 7(1), 1982.

[4] E. Feigenbaum and P. McCorduck. *The Fifth Generation: AI and Japan' s Computer Challenge to the World.* Addison-Wesley, Reading, MA, 1983.

[5] K. Fuchi. Aiming for knowledge information processing systems. In *International Conference on Fifth Generation Computer Systems*, October 1981.

[6] M. Ginsberg. *Essentials of Artificial Intelligence.* Morgan-Kaufmann, San Mateo, CA, 1993.

[7] G. F. Luger. *Artificial Intelligence: Structures Strategies for Complex Problem Solving.* Addison-Wesley, London, 2002.

[8] J. P. Myers, Jr. The central role of mathematical logic in computer science. *SIGCSE Bulletin*, 22(1), 1990.

[9] J. P. Myers, Jr. Looking back: computing history, 2020. (Lightning Talk, CCSC:SW Conference. March 2020).

[10] J. P. Myers, Jr. and K. Yamakoshi. The Japanese fifth generation computing project: A brief overview. *Journal of Computing Sciences in Colleges*, 36(2), 2020.

[11] R. E. Prather. *Elements of Discrete Mathematics.* Houghton-Mifflin, Boston, 1986.

[12] T. Richards. *Clausal Form Logic: An Introduction to the Logic of Computer Reasoning.* Addison-Wesley, New York, 1989.

[13] M. Triska. The power of prolog (https://www.metalevel.at/prolog). 2005.

[14] R. Turner. *Logics for Artificial Intelligence.* Wiley, New York, 1984.

[15] M. Vardi. Conversation with the author, September 2018.

[16] D. H. D. Warren. A view of the fifth generation and its impact. *AI Magazine*, 3(4), 1982.

[17] K. Yamakoshi and J. P. Myers, Jr. Assessing the Japanese fifth generation project. (in preparation).

APPENDIX 1: Algorithmic Conversion of a FOPL Sentence to Clausal Form

We will convert this sentence into clausal form, thereby revealing most of the steps of the algorithm: $\forall x(\forall y(P(y) \rightarrow Q(y,x)) \rightarrow R(x))$

1. **remove →' s** (using the equivalence $A \rightarrow B \equiv \neg A \vee B$):
   $\forall x(\neg\forall y(P(y) \rightarrow Q(y,x)) \vee R(x))$; and again, concluding with:
   $\forall x(\neg\forall y(\neg P(y) \vee Q(y,x)) \vee R(x))$

2. **move ¬ inward** (here using De Morgan' s Law: $\neg\forall t(A \vee B) \equiv \exists t(\neg A \wedge \neg B)$:
   $\forall x(\exists y(\neg\neg P(y) \wedge \neg Q(y,x)) \vee R(x))$; then using the equivalence $\neg\neg A \equiv A$:
   $\forall x(\exists y(P(y) \wedge \neg Q(y,x)) \vee R(x))$

3. **remove ∃' s** (Skolemize existential variables; here, ∃y is within the scope of ∀x):
   $\forall x((P(f(x)) \wedge \neg Q(f(x),x)) \vee R(x))$

4. **remove quantifiers** (since all variables are now universally quantified):
   $(P(f(x)) \wedge \neg Q(f(x),x)) \vee R(x)$.

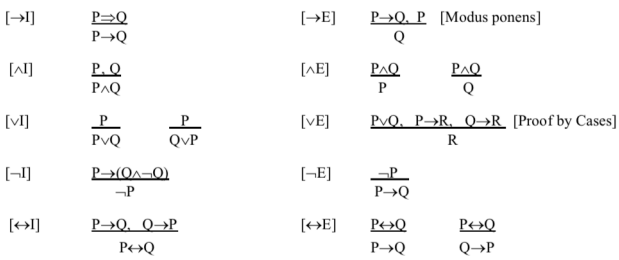
5. **place into conjunctive normal form** (distribute ∨ over ∧, with the equivalence $(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$:
   $(P(f(x)) \vee R(x)) \wedge (\neg Q(f(x),x)) \vee R(x))$

6. **reintroduce →'s** (again using the equivalence $A \rightarrow B \equiv \neg A \vee B$):
   $(P(f(x)) \vee R(x)) \wedge (Q(f(x),x)) \rightarrow R(x))$.

7. **separate the conjuncts** to produce a *set of clauses* (implicitly conjuncted):
   $\rightarrow P(f(x)) \vee R(x)$
   $Q(f(x),x)) \rightarrow R(x))$

[Note: there is a small technicality not needed here, involving scope and the possible renaming of variables between steps 3 & 4 (see [1]). But the above shows the basic conversion process.]

APPENDIX 2: Gentzen' s Natural Deduction Rules for Propositional Logic [11]



APPENDIX 3: Curricular Relevance & Uses of LP

The author previously taught a course on LP (clausal-form logic upon which Prolog is based) in a study abroad course in Osaka, Japan for undergraduate CS majors. The course was on Knowledge-Based AI where the emphasis was on principles rather than programming techniques. Hence, the main prerequisite was a standard Discrete Structures course. The bulk of such a course is FOPL; students completed the course with a substantial grounding in logic, a topic central to CS [8].

Suggested possible courses for the presentation of LP:

[DS]	Discrete Structures	[DB]	Database
[LC]	Logic & Computing	[LP]	Logic Programming
[AI]	Artificial Intelligence	[PL]	Programming Languages

Over the years, the author has taught each of the above to undergraduates (except DB), incorporating at least a discussion of LP and/or Prolog. Below are broadly listed syllabus topics for an entire course on LP (excerpts may guide the use as lectures or modules in relevant courses as shown in the above table).

(A). Introduction: The Central Role of Logic in CS and AI; discussion [8]:

(B). The Japanese Fifth Generation Computing Project; text [4]:

1. Background, history, and purpose.
2. Reactions worldwide.
3. "Failure" ? "Success" ?
4. Legacy.

(C). Logic Programming; text: [6]:

1. AI knowledge representation & reasoning.
2. Propositional & predicate logics.
3. Clausal form & resolution.
4. Logic as computation (programs).
5. Skolemization.
6. Unification.
7. Applications.

Certain aspects of this LP course outline might be selected for the recommended courses:

- (A) is germane for: DS LC AI DB LP PL
- (B) is germane for: AI DB LP
- (C) is germane for: DS LC AI LP PL

Towards Hardware Literacy for Undergraduate Computer Science Students*

Gongbing Hong and Kenneth Trussell
Information Systems and Computer Science
Georgia College and State University
Milledgeville, GA 31061
{gongbing.hong,kenneth.trussell}@gcsu.edu

Abstract

We propose a hardware course component to a course such as *Computer Organization and Architecture*. Along with it, we present a delivery method of the course component. This hands-on component gives undergraduate computer science students an experience to build a *very simple* but *working* computer from a given design. Through this experience, students achieve a level of hardware literacy so that they can expand their career into software development areas that require a basic understanding of hardware.

1 Introduction

In the last two or three decades, computer science programs have gradually shifted to a more software-focused program, more precisely, an application development oriented program with much less focus on low-level hardware knowledge. If one compares the hardware content required by the latest computer science curricula [3] with its previous versions [2, 7], it is not hard to see that hardware content has been reduced to a bare minimum. Small colleges where the teaching load is heavy and many professors teach out of their personal interest areas are estimated to suffer even more.

In practice, hands-on hardware labs are most likely no longer offered. Some replace hands-on labs with simulations, which do not necessarily simulate every aspect of a hands-on lab. As a result, recent computer science graduates are more likely to be hardware *illiterate*. They are less likely capable of working in the fields of software development for industrial control systems, embedded systems, device drivers, network software stack, and system software.

With the proliferation of embedded systems and hand-held mobile devices, we are still motivated to offer a hardware experience to our undergraduate computer science students. In this paper, we introduce a hands-on hardware course component that helps enhance the undergraduate computer science experience. It is relatively easy to implement even in a small liberal arts college such as ours where technology oriented disciplines are not the institution's main focus. Our goal is for our computer science graduates to achieve a basic level of hardware literacy so that they can expand their career horizon beyond common enterprise applications development to software development areas that require a basic understanding of hardware.

The paper is organized as follows. In section 2, we review the relevant background information. We discuss the current trend of a reduced hardware computer science curriculum and review the very simple computer [4] this work is based upon. In section 3, we present our design and implementation of the very simple computer for the proposed hardware course component. After that, we suggest a content delivery method in section 4. We provide some discussion in section 5. Finally, we conclude the paper with remarks in section 6.

2 Background

Despite hardware being so fundamental to computing, we have seen hardware content being drastically reduced from computer science curricula. In this section, we will first review the trend in the reduction of hardware requirements for computer science curricula. Then we will review the Very Simple Computer that forms the base of our proposed hardware course component in the paper.

2.1 Trend of Hardware Requirements in CS Curricula

From both theoretical and practical perspectives, Hoffman [1] argued for the inclusion of more digital logic components in a computer science curriculum. He compared two past revisions of the computing curricula from year 1991 [7] and year 2001 [2]. He found that the required core hours in the knowledge area of computer architecture and organization (AR) was reduced from 59 to 36, a 39% reduction. Fast forward to the latest 2013 revision of the computer science curricula [3], the number of required core hours in AR is further reduced to only 16 core-tier2 hours with 0 core-tier1 hours.

While this trend occurs with good intentions such as being able to accommodate new and popular topics, it may eventually lead to the complete hardware illiteracy of computer science graduates. If it continues, computer science graduates who are trained to specialize in software development may no longer be prepared to do development work in the areas of software that require a basic understanding of hardware. For certain areas such as network software stack and system software that are so critical to computing, whether or not that is a good thing is hard to know right now.

2.2 Review of the Very Simple Computer

We disagree with the current trend of a reduced hardware curriculum and believe our computer science graduates should be equipped to do software development work that requires a basic understanding of hardware.

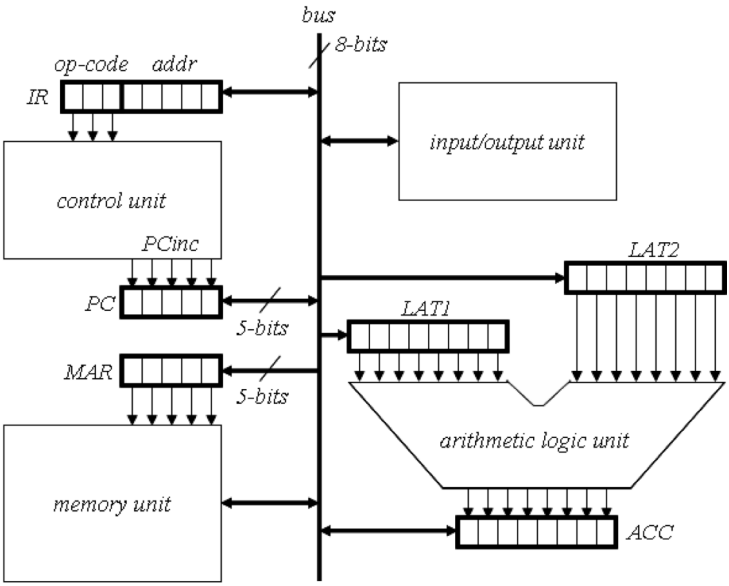


Figure 1: Organization of the Very Simple Computer (reprinted from [5]).

We have incorporated a hardware course component based on Pilgrim's Very Simple Computer (VSC) [4, 5]. VSC as its name suggests is indeed very simple but still sufficiently educational with all basic functional components of a working computer (Figure 1). While the 8-bit VSC only supports 8 instructions, its design still provides a great insight into the hardware realization of a working computer. In addition, hardware realization of a control unit remains

as an elective component in the latest computer science curricula [3]. VSC includes a small memory in its design. Although it only supports 32 memory locations, it fully demonstrates the design of a RAM. VSC also includes a simple I/O system which can be used to program the simple machine and view its output.

3 Design and Implementation of the VSC

The design of VSC given by Pilgrim in [5] includes a number of PowerPoint slides with block diagrams but without a detailed implementation. While it is certainly possible for a few stronger students to create a working schematic from the design and then successfully wire the VSC on a few breadboards, most students will likely be at a loss. Therefore, we created a schematic based on the design for the students. This is reasonable because our goal is simply for the students to become familiar with hardware. Our objectives for the students are two-fold: (1) to be able to read a schematic using publicly available datasheets of various ICs so that they can do software work that requires the understanding of hardware in the future; (2) to wire a digital system for an undergraduate study experience using small- to medium-scale ICs on breadboards. We think this is a more realistic expectation out of a computer science student.

Our original breadboard version of the schematic is available here[1]. From this schematic, a printed circuit board was made for verification purposes. Even with a given schematic and plenty of time, we still expect students to have difficulties with successfully building a VSC on breadboards because so many practical problems are stacked against them. For example, the inexperience of the students, loose wires and mis-wires, poor electrical contacts, bad IC components, etc., are among the common problems.

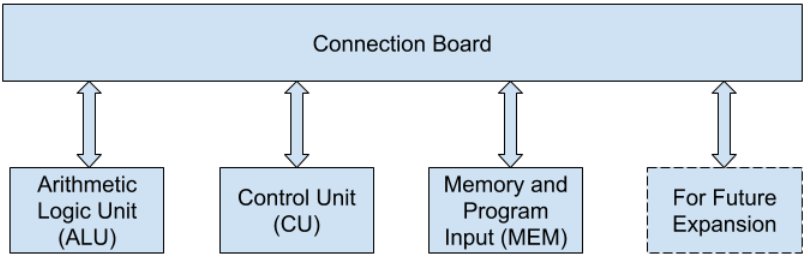


Figure 2: VSC Module Connection Diagram.

To address these difficulties, we further modularize our schematic into an

[1] https://drive.google.com/open?id=1HM-UsD69HjbEj2McoSp7mHA_Osca8MJd

updated version (available here[2]). This new schematic divides the VSC into four modules: the Arithmetic Logic Unit (ALU), the Control Unit (CU), the Memory and Program Input (MEM), and the Connection Board. In addition to providing power, clock, and output display functions, the Connection Board plays a key role of interconnecting the other three modules (Figure 2).

With this new design, the work of building a VSC by the students can be phased into several hardware labs. To make testing easier, we implemented small (2.6-inch × 3.0-inch) separate printed circuit boards for each of the ALU, CU, and MEM modules. When the boards are plugged into the connection board via standard 40-pin header connectors, the VSC is complete and operational. The complete assembly is shown in Figure 3. Now students can build the entire VSC on breadboards *module by module* and test each module independently with confidence that the remainder of the system is functional. For example, if a student builds an ALU module, s/he can replace the pre-built ALU module with her/his breadboard realization of the module for a test run, which can simply include running a few VSC assembly programs and check to see if they produce correct results. In this way, the student can focus on troubleshooting just *one* module before moving forward to working on another. Students can build the entire VSC in this step-by-step fashion.

Our finished VSC has a number of useful features including:

- LEDs showing the status of the data bus, address bus, and ALU latches
- Ability to vary the clock speed from sub-1 Hz to about 500 Hz via a potentiometer
- Ability to disable the free-running clock and instead single step through each of the eight fetch and execute cycles
- Ability to load a program into memory via DIP switches or from an interface to a computer.
- An expansion connector on the Connection Board for future use

4 Content Delivery

In this section, we present a content delivery method, which we have used for years, for the course component we proposed in the paper. If the reader adopts this course component in any of their courses, they should adjust their method of delivery according to their own institutional setup.

We first have a two-in-one *Assembly Programming and Digital Logic Design* course ahead of the *Computer Organization and Architecture* course. Students

[2] `https://drive.google.com/open?id=1zFFv0SK3egY_gbWgLgEB2YY1wjwM0FFU`

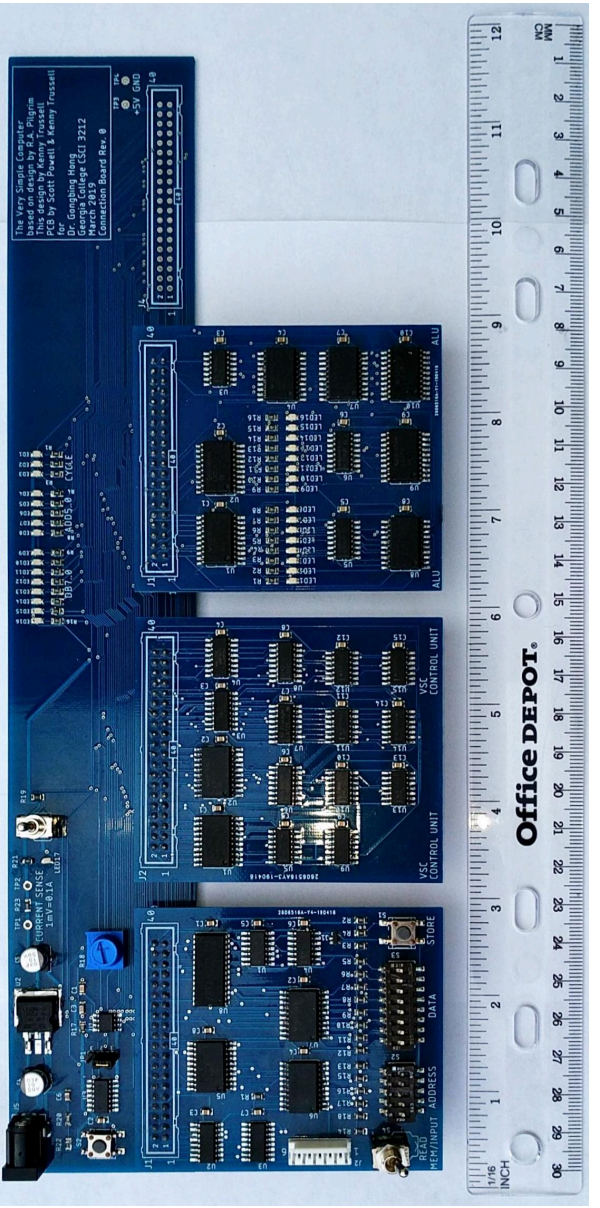


Figure 3: VSC Modular Boards

in that two-in-one course spend roughly a quarter of a 15-week semester studying introductory combinational logic design using only the most basic logic gates. The course has two labs for students to implement two combinational logic functions with different requirements using TTL IC chips.

In the *Computer Organization and Architecture* course, we continue the study of combinational logic with popular small- to medium-scale ICs such as adders, decoders, and multiplexers. Students do a multiplexer lab to implement a logic function and learn the use of chip select. After that, we start our content on sequential circuits, for which we introduce register transfers and memory structure. Students complete a lab using RAM chips and tri-state line drivers.

Following that, students can start building the ALU and then MEM modules on their own. Simultaneously we introduce the design of VSC and its instruction set. Assembly code examples using these instructions are given. Students learn how to manually assemble the examples into machine code. When students finish building a module, they must immediately hook it up with the Connection Board for a test by running some of the example code such as these[3].

This is then followed by the introduction of a simple computer [6] including its organization and hardware design. By this time, students are ready to build the CU module. Again, the module, once built, must be hooked up with the Connection Board for testing. While it is encouraged, we do not expect students to build the Connection Board module unless time permits as this module is most labor intensive. Finally the entire course component culminates with the students integrating all of the modules they have built. The sequence of the lab work leading to the final hardware module integration is summarized in Figure 4.

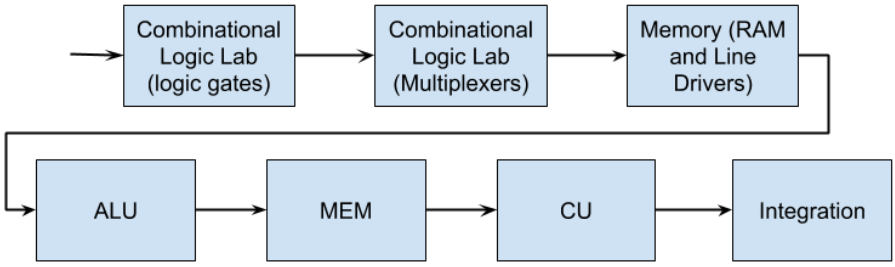


Figure 4: Lab Sequence

[3] https://drive.google.com/file/d/1ROGIMXtbOOxJqnIaT9NpEWcQGBAHbQgo

5 Discussion

For departments in small colleges similar to ours, one immediate concern for adopting this hardware course component is the cost. Overall, the cost of building a VSC using small- and medium-scale TTL ICs is estimated to be no more than $100. We use TTL ICs because of their robustness. The necessary hardware components including ICs, breadboards, and wires can be ordered from online electronics distributors such as Digi-Key and Mouser for reasonable prices. The most consumable supply is likely wires but most of the hardware components can be reused year after year. Although the initial cost may be a little high, over the semesters the cost on average becomes quite low. We charge students a $40 course fee. Over the years, using these fees, we have built a substantial inventory of IC chips, breadboards, and lab equipment. We do not have a dedicated lab space but a shared single-room, multiple-purpose lab.

Another concern is likely time. Lecture time is always at a premium, so the addition of these hardware labs seems problematic. We only allocate class time for the introductory labs. The rest of the labs are done as outside assignments. Students may work in groups of two or three. Group projects help students learn the dynamics of teamwork and the value it brings. Class time is still available for important topics such as interrupts, memory hierarchy, and parallel architectures.

In addition to getting students familiar with hardware, the hands-on experience provides other benefits to students. "Tinkering" with hardware was quite routine for the old-timer computer science students but is completely new to many current students. As Pilgrim already observed in 1993, "In such a rapidly changing area as computer technology, it is easy to forget that even small-scale integrated digital logic technology is new to the student." [4]. Having the opportunity to know this level of hardware details is considered a "bonus" by many of our computer science students. At a minimum, it helps demystify the *magic* machine that can follow instructions.

Before this new modular design, we only achieved moderate success with the building of a working VSC. Some groups of students have had some of their modules working. For years only one group of students have ever gotten their entire VSC working. Despite the low success rate, the feedback from our students was mostly positive. Due to the COVID-19 global pandemic, in-person meetings were canceled this semester. As a result, we did not get the chance to try our new modular design. However, we do expect the new modular design will increase the success rate of building a working VSC. We will report the results with this new modular design in the future.

6 Conclusion

We disagree with the current trend of a reduced hardware computer science curriculum. This trend, if continued, will eventually leave computer science graduates completely hardware illiterate. Computer science graduates will no longer be able to do software development work that requires even a minimal understanding of hardware.

In this paper, we proposed a hardware course component to a computer science course such as *Computer Organization and Architecture*. Through a hands-on hardware experience, we intend to achieve a level of hardware literacy among our computer science graduates so that they can expand their career horizon beyond common enterprise applications development to the fields of software development such as industrial control systems, embedded systems, device drivers, network software stack, and system software.

For this hardware course component, we provided a design and implementation of the Very Simple Computer (VSC) by Pilgrim [4]. For educational purposes, this 8-bit VSC is really simple to follow. With a modular design, it is easy to do and cost-effective. In addition, we proposed a delivery method we have been using for anyone willing to incorporate this hardware course component into any of their courses. We also addressed potential concerns about the delivery of this course component. During our delivery of this course component, students have given mostly positive feedback. We believe building the VSC is an experience they will cherish through a lifetime.

Acknowledgments

First and foremost, we would like to thank Dr. Robert. A. Pilgrim for his work on the Very Simple Computer, upon which this work is based. We would also like to thank our former students, Harrison Statham, Scott Powell, and Christian Jimenez, for their work on the project in the past. Specifically, we thank Dr. Tanya Goette, our department chair, for her full support on this project. Finally, we would like to thank the anonymous reviewers for their invaluable feedback, which makes the paper better.

References

[1] Mark E. Hoffman. The case for more digital logic in computer architecture. In *Proceedings of the Sixth Australasian Conference on Computing Education-Volume 30*, pages 137–143. Australian Computer Society, Inc., 2004.

[2] Association for Computing Machinery (ACM) and IEEE Computer Society Joint Task Force on Computing Curricula. Computing curricula 2001. *J. Educ. Resour. Comput.*, 1(3), September 2001.

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

[4] Robert A. Pilgrim. Design and construction of the Very Simple Computer (VSC): A laboratory project for an undergraduate computer architecture course. In *Proceedings of the twenty-fourth SIGCSE technical symposium on Computer Science Education*, pages 151–154, 1993.

[5] Robert A. Pilgrim. The very simple computer. `http://csclab.murraystate.edu/~bob.pilgrim/405/lectures/Chapter_8_Very_Simple_Computer.ppt`, Accessed May 12, 2020.

[6] Sajjan G. Shiva. *Computer Organization, Design, and Architecture*. CRC Press, 2013.

[7] Allen B. Tucker. Computing curricula 1991. *Commun. ACM*, 34(6), June 1991.

Automated Classification of Collaboration Skills in Typed-Chat Collaborative Problem-Solving*

Jung Hee Kim[1], Joelle Banks[1], Duy Bui[2], Michael Glass[3]
[1]Computer Science Department
North Carolina A&T State University
1601 E. Market St.
Greensboro, NC 27411
jungkim@ncat.edu, jlbanks@aggies.ncat.edu
[2]L3Harris
2235 Monroe Street, Herndon VA 20171
duyquangbui111893@gmail.com
[3]Computing and Information Sciences
Valparaiso University
Valparaiso, IN 46493
michael.glass@valpo.edu

Abstract

This experiment trained classifiers to monitor the dialogue of students working together in a Java programming class. The classifiers recognized four activities within the problem-solving conversation: sharing ideas, negotiating, regulating, and maintaining conversation. These dialogue acts are characteristic of problem-solving conversations. This experiment trained classifiers that utilize specific words in the dialogue. It also trained classifiers that use a statistical topic model built from the dialogue transcripts. If dialogue acts can be recognized, then the counts of student interactions could be used for computer monitoring of online student collaborative group exercises.

1 Introduction

This study works toward computer identification of the collaboration dialogue acts of students working together solving problems. The students are in a Java programming class. In this class, small-group problem-solving exercises are administered as a way for students to learn and apply conceptual knowledge. As they work, student conversations are partly monitored by teaching assistants and the instructor [6].

A goal of the COMPS project is to provide computational assistance in overseeing the student conversations. The computer could judge in real time whether the conversation groups are productive or could benefit from intervention. The computational model should work for a broad variety of college classes [11]. It should also detect a variety of conversational phenomena, e.g. students becoming frustrated or not working together.

Dialogue acts are the different actions that a person can take while interacting with other people in a conversation. For example, in a problem-solving conversation some dialogue acts might be advancing a new idea or referring to some information from the problem statement. Other dialogue acts are disagreeing with another person, summarizing earlier ideas, and suggesting the next problem-solving move. Categorizing the parts of the conversation in terms of dialogue acts provides a basis for analyzing or assessing the conversation. Relative frequencies of dialogue acts are potentially diagnostic. For example if students never negotiate then it is not likely that all students are engaged in the problem-solving task. Counting pairs of successive dialogue acts can reveal interactive behaviors, e.g. [1]. The theoretical basis of this project is by first analyzing a dialogue as a sequence of small dialogue acts, it could be possible to approximately assess a variety conversational phenomena. This method does not depend on knowing the substance of the students' conversation. Thus it could be generally applicable to student small-group problem-solving exercises.

Prior work toward machine-assessment of COMPS project Java dialogues targeted linguistic phenomena that were less specific to problem-solving conversations: initiate-respond pairs [4] and whether students were contributing substantive turns and agreeing or disagreeing [3, 11]. Compared with dialogue acts, the phenomena studied earlier were less directly related to collaborative problem-solving skills. For example, when students are chatting about their summer vacations their conversation will contain initiate-respond pairs and could contain agreement/disagreement.

In this experiment, 1395 turns of dialogue were annotated manually according to a simplified set of four problem-solving dialogue acts. The annotated dialogue turns were used for training and testing four different classifiers, one for recognizing each type of dialogue act. The features from the text utilized by the classifiers were derived from the dialogue text in three ways:

- the presence or absence of individual words, using the most frequent English words attested in the transcripts. These did not include words which were specific to the topic of the conversation, which was analyzing Java code.
- individual words including Java-related words such as "method" and "double".
- topic modeling [9], a statistical technique based on word-co-occurrences which models each turn as a combination of a small number of latent feature values.

A motivation for training with different feature sets is to train classifiers that are independent of the particular problems the students were discussing. A danger is that machine learning can utilize the variable names from the Java code under discussion. This classifier won' t work as well when presented with students discussing a different problem. By being careful in selecting individual words as features, we were able to experiment with a classifier that is not cognizant of any words from the computer programming domain. We also experimented with a classifier that was cognizant of Java programming terms generically, but excluded words from the specific problem.

This paper describes the dialogue act categories and the transcripts of student dialogues. It then reports on experiments in training classifiers for the acts.

2 Background

2.1 Dialogue Act Categories

For this study we adopted categories of dialogue act as defined by Hao et al. [5]. Each different dialogue act corresponds to a skill that students can evince while engaged in collaborative problem-solving (CPS). The list of dialogue acts was developed for the purpose of assessing student CPS ability. The published scheme has 33 skills among four different categories. For purposes of this experiment we utilize only the categories, as shown in Table 1.

2.2 Theoretical Background

The theoretical underpinning of this experiment is: it is possible to detect fingerprints of various collaborative problem solving dialogue phenomena from the dialogue acts. For example [2]:

- Counts of successive dialogue-actions in CPS discussions show sequences of sharing followed by negotiating occur much more frequently than chance.

Table 1: Dialogue Act Categories, after Hao et al. [5]

	Dialogue Act Category	Definition
A	Sharing ideas	Student advances an idea or points to problem-relevant information.
B	Negotiating ideas	Student agrees/disagrees, rephrases or completes or elaborates, identifies a conflict or a gap or modifies the ideas from teammates, modifies or updates own previous ideas.
C	Regulating problem-solving	Metacognitive processes for the team: identifying the goals, expressing lack of understanding, suggesting next step, reflect on problem-solving process, etc.
D	Maintaining communication	Student engages in social interactions, apologizes, corrects spelling, offers help, prompts other students, etc.

- The counts of dialogue interactions are distinctly different when the students are working among themselves, versus when the teaching assistant is involved in the conversation.
- The different roles of students within a discussion can be discerned by counting dialogue acts. The student who was most prepared at the start of the conversation (measured by pre-test) produces more sharing dialogue acts. A less-prepared student negotiates relatively more often. The least prepared student engages in relatively more frequent conversation maintenance.

The skills categories of Hao et al. [5] correlate with categories of transactive activities cataloged by Weinberger and Fischer [10] that are often used in dialogue analysis. Transactivity is the social mode of knowledge co-construction, the ways that students can interact while engaging in a group cognitive exercise. For example the "sharing" skill corresponds to an "externalization" transactive act. Skills such as rephrasing or identifying a conflicting idea, which are labeled "negotiating" by Hao et al., are "integration-oriented" and "conflicted-oriented" consensus building transactive acts. The correspondence between student CPS skills and transactive acts indicates that these categories of dialogue act are likely to be fruitful for dialogue analysis.

Student	Text	Categories
S1	its calling the method foo and then it is reading throught he array	A
S1	the*	D
S2	yea but what its it printing out	B, C
S3	"foo" is an array, not method	B
S3	0|0|0	A
S1	your right my mistake. so they are both printing 0|0|0	B
S3	this is what I think the second one is printing out 0|0|0 1|1|2 2|2|4 3|3|6 0|0|0	B
S1	yes i agree your using the array values in the tostring method so its fromatted the same but has different values	B
S3	yeh and foo[4] wouldve been 4|4|8 but then you have that line of code foo[foo.length-1] = f; which sets foo[4] to f	B
S1	ok so we have an answer	C
S3	so im bout to type the answer so he can check it	C

Figure 1: Transcript of Discussion with Manually-Annotated Dialogue Acts.

2.3 COMPS Project Collaborative Exercise Dialogues

Figure 1 shows an extract from students discussing a problem in a second semester Java programming class. The code and questions are visible to the students in a separate document outside the chat window. Three students S1, S2, and S3 are participating. This transcript shows the categories of dialogue acts for each turn.

The dialogue example shows students analyzing Java code. The students do not execute the code, COMPS project exercises emphasize learning and operationalizing Java concepts. Figure 2 shows part of the Java code under discussion. The students in the Figure 1 transcript are debating the result of the print statements, the proposed answers are in the form of numbers and strings the code would print out.

Part of the theory of group problem-solving exercises is that group discussion forces students to verbalize concepts and explain their reasoning [7]. To encourage this process, the exercise protocol asks students to come to agreement on parts of the exercise. Students positively affirm their agreement by clicking a button. Then an instructor or TA inspects the agreed-upon answer and provides feedback. The Java concepts involved in solving the exercise in

```
public class Foo {
    private int x, y;
    private static int z;
    public Foo() { z++; }
    public Foo( x, y) {
        this();
        this.x = x;
        this.y = y;
        z += x + y; }
    public String toString() {return x + "|" + y + "|" + z; }
}
public static void main( String [] args ) {
    int i = 0;
    Foo f = new Foo();
    Foo [] fooarray = new Foo[5];
    System.out.println( f.toString() );

    for (i=0; i<fooarray.length; i++)
        fooarray[i] = new Foo( i, i );

    fooarray[fooarray.length-1] = f;
    System.out.println(fooarray[fooarray.length-1].toString());
```

Figure 2: Java Code Discussed by Student Problem-Solving Group.

this experiment included: classes, instance variables, static variables, constructors with different signatures, arrays, and instantiating objects within an array. In addition to the printed output discussed Figure 1, other questions concern how many objects were created at various points in the code (including the array itself), and how many objects are still accessible.

The Java terms associated with these concepts occur frequently in the dialogues. This raises possibility that the machine classifier of dialogue acts might be trained to work well with Java collaborative problem-solving exercises if it were cognizant of Java terms.

3 Experiment

3.1 The Dialogue Data

The dialogue data for this experiment consisted of 1385 turns of typed dialogue, from transcripts of 8 collaborative discussions involving 24 students. Each discussion was approximately 1 hour. The discussion exercises were administered

during the programming lab time of a 2nd semester university Java class. The groups were also joined at intervals by teaching assistants. Different dialogue act patterns occurred when a TA was participating in the discussion. As the goal of this experiment is to study student dialogue for purposes of assessing whether the students are working together. The segments of discussion with the TA present were thus not included in the data for this experiment.

An artifact of typed-chat conversation is that the speaker can end a typed chat turn by pressing "enter" and then continue typing, maintaining the conversational focus without anybody intervening. In spoken dialogue this might be a momentary pause in a dialogue turn. When one person had control of the conversation for several successive typed-chat turns, we combined them into a single dialogue turn for classification. Merging successive turns from the same person single turns resulted in a data set with 870 turns.

Every turn was annotated by two coders, who then resolved disagreements. One turn could contain several dialogue acts. Figure 1 illustrates a turn where the student first agrees with the preceding student (a negotiating act), then focuses the discussion on deciding the program' s printed result (a regulating problem-solving act). The result was 1270 dialogue acts. Table 2 shows the frequencies of the four categories of acts.

Table 2: Distribution of Dialogue Act Categories.

Category	Count	Pct
A. Sharing	377	30%
B. Negotiating	406	32%
C. Regulating	259	20%
D. Maintaining	228	18%

3.2 Feature Sets for Classification

Machine classifiers require extracting a set of features from each dialogue turn. The classifier algorithm then learns to predict the dialogue act categories within the turn based on the values of the features derived from the turn.

To be generally useful, it is important to train classifiers that don' t require the presence of words that are specific to the problems under discussion. The word "foo" in these dialogues is the name of a variable. It tends to occur in dialogue acts that are sharing or negotiating but not in dialogue that is simply maintaining conversation. The presence or absence of "foo" could be excluded from the feature set, so that the classifier won' t utilize its predictive association.

The first feature set is a vector indicating the presence (1) or absence (0) of 41 different words. For each turn, a vector is produced from the words in that turn. The set was produced from the 50 most common words in the dialogue transcripts. Words that were specific to the problem under discussion, mostly names of variables in the Java code, were excluded [3].

Another feature set started with the 41 common word features but further excluded common English words which were being used in their Java context. These included, e.g., "static" and "class." This feature set of 33 common words was thus even further removed from the specific problems the students were discussing.

A final feature set is a Latent Dirichlet Analysis topic model [9] derived from the transcripts. The topic model contained a 10 number vector for each turn. Conceptually, a topic model is a form of dimensionality reduction. The original text of a dialogue turn could be modeled as a high-dimensional vector with one dimension for the presence/absence of each vocabulary word. In the topic model, each dimension represents bundles of words which tend to co-occur in the same contexts. For the topic model, words which were not in a lexicon of the 10,000 most common English words were excluded.

For the 41-word and 33-word feature models, we checked whether the vocabulary words were likely to contribute information to a machine classifier algorithm. For each word and each dialogue act, a Fischer exact test showed the likelihood that the word contributed information toward diagnosing the presence of the dialogue act. The confusion matrix of presence/absence of word and presence/absence of dialogue act was counted. The Fischer test computed the likelihood that the co-occurrences of that word and dialogue act could be due to chance. Most of the words were significantly related to at least one dialogue act at $p \leq 0.05$.

4 Results and Conclusions

4.1 Classifier Experiment

Scikit-Learn was used for training linear regression classifiers. We used 60% of the data for training, randomly drawn, and 40% for testing. Each classifier was trained and tested with the three feature sets: the topic model features, the 33 common English words, and the set of 41 words including Java concept terms. Table 3 shows the F1 combined precision and recall scores.

4.2 Conclusion and Future Work

The three different feature sets performed similarly. The topic model features, which utilized the largest set of words, was more accurate at recognizing shar-

Table 3: Classification accuracy of dialogue acts using different feature sets.

Feature Set	Sharing	Negotiating	Regulating	Maintaining
Topic model	0.623	0.473	0.545	0.329
Generic words only	0.546	0.534	0.556	0.388
Plus Java terms	0.568	0.510	0.563	0.437

ing. Sharing is the dialogue act which most directly expressed the ideas of the problem. The small sets of generic words more accurately recognized negotiating acts, such as agreement/disagreement. Achieving problem-independent recognition of dialogue acts might be harder for some types of acts.

Although these classifiers are not very accurate, it might be possible to detect anomalies in the relative percentages of dialogue acts. Table 2 shows that each category of dialogue act can be expected to occur frequently. If no regulating turns or no negotiating turns were occurring, for example, even a poor classifier might reveal their relative absence in a typical 150-turn conversation. An earlier COMPS dashboard assessed conversations based simply on relative occurrence of substantive turns and agreement within the dialogues [3].

A deeper model of assessing dialogue would count successive pairs of dialogue actions. The interaction between two students can be recognized by how the second responded to the first, e.g. sharing followed by negotiating. Counting these interactions entails correctly tagging both acts in the pair. Higher classification accuracy will be needed for successfully recognizing these interactions in student dialogues.

Future work includes finding better-performing feature sets. The models used in this experiment used bags-of-words, where the order of the words in sentences did not contribute. We will try word embedding feature vectors that are more sensitive to sequences of words, for example the doc2vec model [8]. We will also develop and test a revised, simpler, annotation rubric, that we hypothesize may result in more consistent categorization.

Potential future work includes applying the same experiment to the threads in class-related discussion board postings, where the students interact with each other but do not post their dialogue turns in real-time interactions.

References

[1] Duy Bui, Jung Hee Kim, and Michael Glass. Dialogue act pairs for automated analysis of typed-chat group problem-solving. In *Intelligent Tutoring Systems, 16th International Conference*, pages 378–381. Springer, 2020.

[2] Duy Quang Bui. Collaboration pattern model for student participation in problem-solving typed-chat. Master's thesis, Computer Science Dept., North Carolina A&T State University, 2019.

[3] Michael Glass, Yesukhei Jagvaral, Nathaniel Bouman, Emily Graham, Stamatina Kalafatis, Lindsey Arndt, Melissa Desjarlais, Jung Hee Kim, and Kelvin Bryant. Problem-independent text analytics for real-time judgment of CSCL typed-chat dialogues. *J. Comput. Sci. Coll.*, 34(1):145–154, October 2018.

[4] Michael Glass, Jung Hee Kim, Kelvin Bryant, and Melissa Desjarlais. Indicators of conversational interactivity in COMPS problem-solving dialogues. In *Third Workshop on Intelligent Support for Learning in Groups (ISLG)*, 2014.

[5] Jiangang Hao, Lei Liu, Alina A von Davier, and Patrick C Kyllonen. Initial steps towards a standardized assessment for collaborative problem solving (cps): practical challenges and strategies. In *Innovative Assessment of Collaboration*, pages 135–156. Springer, 2017.

[6] Jung Hee Kim, Taehee Kim, and Michael Glass. Early experience with computer-supported collaborative exercises for a 2nd semester java class. *J. Comput. Sci. Coll.*, 32(2):68–76, December 2016.

[7] Timothy Koschmann. Understanding understanding in action. *Journal of Pragmatics*, 2(43):435–437, 2011.

[8] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[9] Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[10] Armin Weinberger and Frank Fischer. A framework to analyze argumentative knowledge construction in computer-supported collaborative learning. *Computers & Education*, 46(1):71–95, 2006.

[11] Angelica Willis, Ashana Evans, Jung Hee Kim, Kelvin Bryant, Yesukhei Jagvaral, and Michael Glass. Identifying domain reasoning to support computer monitoring in typed-chat problem solving dialogues. *J. Comput. Sci. Coll.*, 33(2):11–19, December 2017.

Multi-Cohort/Multi-Tier/Cross-Disciplinary Instruction and Research via Short Film Production*

Jerry Tessendorf[1,2], *Timothy Mclaughlin*[3], *Sharath Giramaji*[4]
[1]*School of Computing*
Clemson University
Clemson, SC 29634
[2]*Hagler Institute for Advanced Study*
Texas A&M University
College Station, TX 77843
jtessen@clemson.edu
[3]*Department of Visualization*
Texas A&M University
College Station, TX 77843
timm@tamu.edu
[4]*Department of Ocean Engineering*
Texas A&M University
College Station, TX 77843
girimaji@tamu.edu

Abstract

The creation of a short animated film is the organizing theme behind assembling and instructing multiple teams of students across two universities and multiple generations. The variety of cross-disciplinary skills that the students acquire in the process fall into many categories of creative and technical subjects. Here we describe the concept and origins of the project, faculty and research components, and student experiences from the first two years of the project.

1 Concept and Origins

Our approach to this project utilizes project-based, studio-style teaching and learning for collaborative teams of students. The students are an interdisciplinary group along the spectrum from art, design and humanities to science and technology. The studio-style approach for instruction in undergraduate computing-intensive courses has been used in practice for only about 30 years, as documented by Tomakyo at Carnegie Mellon University in 1987 [16]. Their courses and the subsequent revisions primarily focused on capstone and graduate-level subjects that mirror industry methods for project design, analysis, and code review. Identified benefits of the studio approach for computing-intensive work are the extension of the skill set of the group, the capacity to employ technology attuned to the individual needs of group' s members, and ownership of responsibility for outcomes assumed by the group members, and student agency over aspects of the project for which each assumes responsibility [11][3]. In 1990, Rensselaer Polytechnic Institute in Troy, New York, instituted curriculum-wide changes embracing studio-style instruction [17]. Such substantial change, partially funded by the National Science Foundation, was undertaken to both improve the educational experience in science and engineering, and to adapt a model for delivery of education that was extensible through distance learning.

The embrace of studio-style learning for computing-intensive subjects follows over 100 years of studio-style education in art and design and repeated empirical analysis of the connection between studio learning and creative thinking [7]. Information technology is changing the way studio courses are taught, both in computing and design [13]. The challenge faced by instructors and course developers is how to integrate technology and technology development into the process of iterative discovery rather than allowing technology to become either a barrier to learning or the primary focus of learning; i.e., teaching problem solving, not how to use software.

A new frontier in science and engineering is the design and development of complex systems that operate in extreme environments: e.g, autonomous ships in stormy seas, off-shore energy platforms in hurricanes; airplanes in rough weather; and urban response to flooding/tsunamis. Scientific analyses of these phenomena are difficult (if not impossible) in laboratory set-up or field experiments due to the extreme and stochastic nature of these events. A novel approach to studying systems in extreme conditions is to develop a virtual rendering of the event. To full utilize the data from these simulations, it is important to animate and visualize the full environment so that it can be observed as in a physical experiment. Simulated data field that is animated and visualized is loosely termed a "digital-twin" of the corresponding physical realization. If a digital twin (DT) can be trained with appropriate physical

laws, it can then be used to explore a broad parameter range of the phenomena including extreme conditions. It is evident that DT can also be used for teaching and training. In summary, development of a high-fidelity DT can have a transformative effect on science and engineering disciplines.

One of the biggest challenges in developing DT is the in situ high-fidelity simulation of underlying physical phenomena. Flow physics simulation is particularly difficult due to stochastic long-range interactions, complex non-linear effects, inherent chaos and most importantly, (nearly) infinitely many degrees of freedom (DOF). The state-of-the art of computational fluid dynamic (CFD) simulation of large systems is still in its infancy: it only handle large DOF in highly localized domains or small DOF over large domains. It must be noted even with current computational capability, high-fidelity simulations are restricted to highly localized flow phenomena and only in a small parameter regime. Yet, full-system flow simulations, albeit of very low fidelity, are routinely performed in entertainment industry. These simulations are visually realistic, but need infusion of more physical models for engineering use. A high-fidelity digital twin can be used as a: (1) Research tool for design, implementation and operation in extreme environments; (2) Operational tool for playing out different scenarios; and (3) Training tool for simulating emergency conditions. The long-term goal of DT development is to bridge the divide between rigorous scientific CFD, and flow animation from the entertainment industry, to develop a unique and transformative computational capability that can be used for research, training and teaching. The short film represents a key step in the progress toward the development of a high-fidelity digital-twin model. The short film represents an important event in maritime engineering history.

The creation of a computer animation short film provides a problem set that addresses a range of learning goals in computing-intensive studio-style learning. Computer animation is a complex and time consuming process, in both professional and academic environments. Participants must acquire and apply skills in many subjects, including aspects of software development, computer graphics, animation, natural sciences, engineering, mathematics, anthropology, psychology, optics, and art. In our case study we added to the design and problem solving difficulty by requiring that the forms and motion elements developed must accurately reflect known factual details of engineering, science, and society that drive the storyline of the short film. Creating a compelling visual story with fact-based visuals and high production values can have enormous instructional value for the student participants, who must develop and exercise, as a team, skills in all of those subject areas while applying them with craftsmanship. For such an ambitious project, the organization of the team focuses on software development and usage based on skills and processes

spanning a variety of disciplines, training, experience, educational status, and temporal presence.

Our project began in 2018 and was initially planned to take roughly one year to complete. The project has proven too complex for that period of time and is still underway, using multiple generations of student cohorts to continue the work, at both undergraduate and graduate levels. The undergraduate students major in computer science, visualization, and anthropology. The graduate students are working toward MS, MFA and PhD degrees in Visualization, Digital Production Arts, and Human Centered Computing. At launch, the team consisted on five undergraduate seniors and two graduate students. During the second year the student team reduced to a single graduate student, and as the project spins up for its third year, a new group of students will participate.

The project benefits an ongoing DT software development research project. An environmental scene simulator that includes maritime environments, is in a continual development process since 2016. This software consists of a mix of C++ libraries and Python modules, and creates highly detailed scenes that are rendered with its own global illumination renderer. The simulator, named Gilligan, is mostly developed in-house by faculty, staff, and students. A few open source libraries are used for efficiency and standardization. At the outset of this project, Gilligan's feature set was not up to the full range of needs for this short film, and so the film is a driver of new development.

2 Story Within the Short Film

On November 10, 1975, between 7:15 p.m. and 7:30 p.m., the cargo ship The Edmund Fitzgerald sank in Lake Superior during a strong storm, costing the lives of all 29 crew members and a cargo of 26,000 tons of taconite iron balls. While suffering some damage earlier in the voyage, its complete and sudden loss was unexpected and remains unexplained today. The ship and crew remain at 530 feet below the surface, the wreck has been designated a gravesite, and since 1995 diving expeditions are prohibited. The events and tragedy were the inspiration for the popular song "The Wreck of The Edmund Fitzgerald" by Gordon Lightfoot in 1976. The surviving families of the lost crew have suffered over the years, and apparently have never been officially notified of their loss by the company operating the ship [4].

The sudden and total loss has motivated numerous maritime-safety and engineering assessments of potential factors and failure modes [2, 15], but no concensus has been reached. Multiple explanations, or combinations of explanations, cannot be ruled out, including rogue waves, human error, and structural failure. Interest in this event has come from a vary wide range of disciplines. Known weather data has been tabulated [5]. Weather researchers

Table 1: Description of sequences in the short film.

Sequence	Time	Description
1	5pm, Nov 9	Initial traversal in good weather
2	10pm, Nov 9	Strong storm on the first night
3	10am, Nov 10	Calm waters
4	2pm, Nov 10	Increasing stormy water in the afternoon
5	5pm, Nov 10	High waves in late afternoon; ship damage
6	7pm, Nov 10	Highest waves; ship passes under the surface

and forecastors have analyzed the events in detail [1]. Computer simulations of the weather during the event time frame help to better understand wind and wave conditions, supplementing limited in-situ measurements Documentarians and biographers have interviewed the friends and family of the crew, and witnesses of the final voyage of The Edmund Fitzgerald (although none of them witnessed the moment of sinking), and presented results in books [12] and videos [6, 14]. Engineering analyses, initially from the NTSB [2], up to a very recent study with modern engineering failure models [9], have never found a definitive cause of the event, but point to multiple potential factors.

Ocean engineers, who study the design and function of equipment in maritime environments, are a specific group who find the sinking particularly of interest. This event represents a catastrophic failure of engineering systems – perhaps from a design flaw, perhaps from an operational mistake, or perhaps from tolerances being exceeded – but there is insufficient knowledge to make conclusions. A virtual "re-enactment" of the event that faithfully depicts the conditions at sea and the operational history would be of value to this group as a tool to better inform engineers about harsh operating conditions.

All of these factors of weather and sea conditions, operational details, and societal stress-points strongly influence the design and content of the short film described here. The intent is to create a photo-realistic depiction of the factual events up to the sinking, using all of the data that is available via the extensive literature. Facts that are not known are either avoided by appropriate selection of story points that do not require those unknown facts, or by inferring reasonable conjectures. This short film does not pick a particular scenario for the final moments, or try to develop a new one, because of the speculative nature of such a choice. The depiction of the final moments will visually emphasize the strength and violence of the storm as the ship slips below the surface. Six sequences have been selected depicting key moments during the 30 hour voyage, as described in Table 1.

3 Short Film Production

Production of a short computer graphic film invokes a very broad range of technical and artistic disciplines, shared among team members with distributed responsibilities and skills. For this project, the sharing also takes place across generations of students and faculty that come to the project, contribute their expertise for a time, and depart. A single faculty member serves as overall coordinator across all of the groups. Up to this moment in the project, students and faculty have worked the following aspects of the production:

1. Research the known facts of the story.
2. Gather visual reference material.
3. Construct a timeline of known events and weather conditions.
4. Research factual details about the ship The Edmund Fitzgerald.
5. Install and improve software for production pipeline workflow. This software is patterned after a professional production pipeline [8].
6. Create pipeline scripts for visualization.
7. Create training materials for using pipeline software.
8. Capture 360-degree images of stormy skies.
9. Create composite stormy sky from the 360-degree images for image-based-lighting, as depicted in Figure 1.
10. Initial low resolution modeling of the ship.
11. Create multiple sequences of storyboards of key moments in the timeline.
12. Create animatic video from the storyboards and low resolution model.
13. Research camera options and create a layout reel and animatic video.
14. Generate a high-resolution model of The Edmund Fitzgerald, with many parts.
15. Test initial ideas for whitewater and splash dynamics against the ship hull.
16. Create first surfacing textures for the hull, pilot house, and stern.
17. Rig the ship for animation.
18. Assemble all of the model components into a complete ship model.
19. Revise model shapes and texture coordinates.
20. Create revised surface textures for the entire ship.
21. Create revised rig system for animating the ship.

Figure 1: Two of many 360-degree photos of stormy skies on the Texas A&M College Station campus, and (far right) the resulting composite sky.

Figure 2: Renders combining the ship model, composite sky, and ocean.

22. Create rough "hand"-animation of ship cruising with response to waves.
23. Create Python scripts for exporting the ship model from proprietary software.
24. Create Python scripts for exporting the camera data from proprietary software.
25. Set up ocean surface conditions for each of the story sequences.
26. Render initial demonstrations of the ship, ocean, and sky together. Figure 2 shows the revised model with lambertian shading, combined with ocean conditions corresponding to sequence 1, and the composite sky.
27. Research models of paint Bidirectional Reflectivity Distribution Functions (BRDFs).
28. Create custom code for surface shaders of a paint BRDF model.

There are many more steps to come in completing the short film:

lighting: There are two primary sources of light for these scenes to be rendered: (1) the sky, although most of the film takes place at night in a heavy storm, and (2) the "streetlights" that run the length of the ship on either side of the deck. This very limited lighting will be implemented directly through the custom global illumination renderer in Gilligan. A shader based on the paint BRDF research has to be written in order to get realistic illumination of the ship.

animation: Initially we elected to not use physical simulation techniques to dynamically force the motion of the ship in response to wave motion, in favor of hand-animating the motion of the ship, including ship flexing. Recently one of the graduate students has embarked on developing a rigid body simulator for floating bodies in height-based ocean, extending recent work by others [18].

This will allow us to mix the hand-animation with simulated animation of the ship motion.

FX: To provide splashes against the side of the ship, and up onto the deck, preliminary studies have explored a systematic process of using particles undergoing a variety of dynamical forces and emission mechanisms[10]. This method requires a substantial amount of coding skills, understanding of computer graphics techniques, and hard work.

scene integration: All of the elements have to be integrated into a custom-built Python rendering framework that is under development.

rendering: The complete length of the film is approximately 6500 frames at 24 FPS. Render time for each frame has not yet been budgeted, but it is expected to be between 8 and 80 core-hours per frame, depending on scene complexity.

post production Foley sound and titles.

4 Undergraduate Student Participation

The role of undergraduate students up to this point in the development of the short film is described. Opportunities exist for additional involvement of undergraduate students as the project progresses. At the start of this project, in the Summer 2018 Semester, the participants were five female senior honors students at Texas A&M University. Four were enrolled in the Bachelor of Science in Visualization program and one was pursuing her Bachelor of Arts in Anthropology. The Department of Anthropology is home to Texas A&M's nautical archaeology program. They were guided by supervising faculty to organize their work along the structure of an animated film production, including significant concentration on the front end work of story development, visual development, and tools and pipeline development. Both the story and the visual development were heavily reliant on their research into the form and structural behavior of the ship, weather and sea conditions at the time of the event, and material and light conditions of the ship and its environment. During the summer and fall 2018, the five undergraduates completed items 1-17 of the list in section 3. For this work, the students received the Award of Excellence for Best Presentation of a Creative Work at the 2018 Capital of Texas Undergraduate Research Conference[1]. A joint honors thesis was approved for these students, and the students graduated in the spring of 2019.

The most difficult task was research and development of a system to simulate whitewater and splashes against the ship. Because of the intensity of the

[1] https://sites.google.com/view/cturc/past-conferences/2018-awards-and-pictures

storm, direction and speed of the ship, the waves traveled faster than the ship, approaching from the stern, and were tall enough to crash completely over the deck. Violent splashing of this sort is normally accomplished in industry settings using sophisticated commercial simulation software. For this project, in which we want to avoid commercial software and create capabilities as much as possible, one of the students embarked on a process of assembling simple particle simulations with enhancements from volumetric tools in Gilligan, following the process described for FX in section 3. This required to development of numerous Python and shell scripts to accomplish the basic task of converting particles to volumes, and workflow scripts to conduct huge amounts testing of variations.

5 Graduate Student Participation

During the 2018-2019 academic year, two graduate student volunteers joined the production team. These students were in the graduate program in the Department of Visualization at Texas A&M University. One took on the task of cleaning up pipeline code and organization, and the other tackled research and development of a shader for oil-based paint for rendering the ship in our custom renderer. Both contributed scripts for exporting data from commercial modeling/animation software into the renderer.

During the summer of 2019, a Ph.D. graduate student in the Human Centered Computing division of the School of Computing at Clemson University worked quickly to revise and improve the ship model, animation rig, and texture painting. That student was able to be very productive because he had a previous history in this topic, having received an MFA from the Digital Production Arts program in the School of Computing, and having spent several years working in an animation studio for feature films and commercials. A second Ph.D. student in Computer Science at Clemson University has developed sophisticated algorithms for creating detailed and realistic splash and foam events (a publication is in preparation), which will be incorporated into the film production workflow soon. A Digital Production Arts MFA student recently joined the production team and is focused on implementing rigid body dynamics for the ship in response to forcing from the ocean waves.

6 Faculty, Administration, and Staff Participation

Faculty, Staff, and Administration have acted throughout this project in both supportive and leadership roles. The project arose from the combination of an initiative jointly driven by the Department of Visualization and LAUNCH program at Texas A&M University to involve honors students in research and/or

creative development, from the curiosity of faculty in the Department of Ocean Engineering about the long-standing engineering mysteries in the sinking of The Edmund Fitzgerald, and from the interest by the Hagler Institute of Advanced Study in collaborative out-of-the-box research and creativity. The years after the first one have brought opportunities for students in the Digital Production Arts program at Clemson University to participate in a way that hones skills developed in that program. This project is an opportunity to collaboratively mix many organizations and ambitions within our university administrative organizations.

At the outset of this project, studio space was provided by the Department of Visualization for the first year, so that students, supervisors, and faculty could work together in close proximity. This strongly facilitated and focused the creative process for the students. Because of the computational demands of the computer graphic elements of the project, specialized computing facilities and IT support were provided by the Department of Ocean Engineering during that initial one year period. In later years, as we near the moment of rendering the full frame set for the film, we will be able to use the Palmetto Linux Cluster[2], a resource available to all faculty and students at Clemson University.

7 Conclusions

Faculty working in Visualization and Digitial Production Arts have, for many years, recognized the unique educational character and opportunities afforded by students building a short film from start to finish. Despite the affordable availability of very advanced commerical graphics software running on consumer grade hardware, production of carefully crafted short films requires the participants to hone skills in art, computer science, psychology, coding, physics, and many other disciplines. By choosing a topic for the short film that is complex and relevant to engineering interests, and setting the quality expectations high, we are able to extend this educational opportunity to a larger group of students across disciplines, institutions and time, while encouraging them to develop skills beyond normal outcomes. Simultaneously, this project successfully induces new research and development of "Digital Twin" environmental simulation software with capabilities well exceeding those otherwise available.

Acknowledgements

This project has been supported by the Texas A&M University Departments of Visualization and Ocean Engineering, the Hagler Institute for Advanced Study,

[2] https://www.palmetto.clemson.edu/palmetto/userguide_palmetto_overview.html

the Naval Information Warfare Center, the Texas A&M University Undergraduate Research LAUNCH program, and the School of Computing at Clemson University.

References

[1] Steve Ackerman. The storm that sank the edmund fitzgerald. 2016. https://www.youtube.com/watch?v=NLUzyNuMqTM.

[2] National Transportation Safety Board. Marine accident report ss edmund fitzgerald sinking in lake superior. Technical Report NTSB-MAR-78-3, National Transportation Safety Board, 1978.

[3] Huseyin Ergin. Instructor-formed capstone teams based on interest and technical experience: The road to success. *J. Comput. Sci. Coll.*, 35(5):37–49, October 2019.

[4] Private Communication from a docent at the Great Lakes Shipwreck Museum. 2018. From a discussion with a docent at the Great Lakes Shipwreck Museum.

[5] Michigan Sea Grant. Data set: Weather conditions before and after the sinking of the edmund fitzgerald on nov. 10, 1975. http://www.miseagrant.umich.edu/downloads/lessons/datasets/earthscience/edmund_fitzgerald.xlsx.

[6] Mickey Hanson. Shipwreck: The mystery of the edmund fitzgerald. 1995. https://www.youtube.com/watch?v=Q3uOnnIv5Qs.

[7] Deniz Hasirci and Halime Demirkan. Understanding the effects of cognition in creative decision making: A creativity model for enhancing the design studio process. *Creativity Research Journal*, 19(2-3):259–271, 2007.

[8] Chris Johnson, Josef Tobiska, Josh Tomlinson, Nico Van den Bosch, and Wil Whaley. A framework for global visual effects production pipelines. 2014. https://vimeo.com/116364653.

[9] Sean Kery and Ben Fisher. A forensic investigation of the breakup and sinking of the great lakes iron ore carrier edmund fitzgerald, november 10th 1975, using modern naval architecture tools and techniques. 11 1975.

[10] Markus Kurtz and Jerry Tessendorf. Simulation, simulation, simulation: Integration of multiple simulation techniques to create realistic water motion in order to maintain highest level of flexibility. In *ACM SIGGRAPH*

2007 Sketches, SIGGRAPH ' 07, page 94–es, New York, NY, USA, 2007. Association for Computing Machinery.

[11] P. A. Laplante. An agile, graduate, software studio course. *IEEE Trans. on Educ.*, 49(4):417–419, November 2006.

[12] B. Lynn, C. Winters, and J. Lothman. *The Legend Lives on: S.S. Edmund Fitzgerald*. Running Light Press in partnership with the Great Lakes Shipwreck Historical Society, 2015.

[13] M. L. Maher, S. Simoff, and A. Cicognani. *Understanding Virtual Design Studios*. Springer-Verlag London, 2000.

[14] Ric Mixter. Library lecture series | the edmund fitzgerald exploration. 2017. `https://www.youtube.com/watch?v=CH5936AVGq4`.

[15] Office of National Marine Sanctuaries, Office of Response, and Restoration. Screening level risk assessment package - edmund fitzgerald. Technical report, National Oceanographic and Atmospheric Administration, 2013.

[16] J. E. Tomayko. Teaching a project-intensive introduction to software engineering. Technical Report SEI-SR-87-I., Software and Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1987.

[17] J. Young. The studio classroom. *ASEE Prism*, page 15, January 1996.

[18] Alexandra L. Zheleznyakova. Physically-based method for real-time modelling of ship motion in irregular waves. *Ocean Engineering*, 195:106686, 2020.

Writing and Speech Instruction in an Introductory Artificial Intelligence Course*

Andy D. Digh
Mercer University Computer Science Department
1501 Mercer University Drive, Macon, GA 31207
digh_ad@mercer.edu

Abstract

Providing opportunities in both written and oral communication continues to grow in importance within the undergraduate Computer Science major. An introductory course in Artificial Intelligence that includes the ethical issues surrounding how society will respond to automation and machine learning systems in the future workplace is the perfect place to provide some instruction in writing and diction specific to computing. This paper explicates how students in a recent first course in AI became stronger writers and speakers because of this instruction.

1 INTRODUCTION

In the criteria for accrediting computing programs, the "ability to communicate effectively with a range of audiences" is imperative [5]. In the last two years at Mercer University, we have been tasked with teaching our own majors skills in writing and oral communication, which are specific to each of our disciplines within the College of Liberal Arts and Sciences. This is designed to build upon the foundational skills they acquired in the first two years of their general education courses.

1.1 Background

Assigning writing is not the same thing as giving students writing instruction. Writing is a slow, difficult, and recursive process. Writing should be repeated over and over again throughout the undergraduate curriculum for students to become better and stronger at it [3]. As Zobel indicates in his book *Writing for Computer Science*, "many researchers undervalue the importance of clarity, and underestimate the effort required to produce a high-quality piece of writing" [20].

In our college at Mercer University, we have a three-course core integrative curriculum for students in their first two years with writing instruction that includes modules on thesis development, citing sources responsibly, and structuring arguments and paragraphs. Students are also exposed to different types of writing so that once they arrive in their junior and senior level courses they are well prepared for writing, documenting, and speaking about significant research.

Although we had been assigning some writing and oral presentations throughout our computer science curriculum for many years, we had not been doing any specific instruction in these areas especially in our upper level courses. As Taffe argues, "professional writing must be taught in Computer Science courses as a continuation of the more general writing instruction of general education courses" [15].

1.2 Exposing Students to Different Kinds of Writing Early in the CS Curriculum

Ideally, writing can be taught throughout the major discipline, and begins in those early courses [8]. By including writing in any computer science course, you emphasize that "writing is important" and that "principles taught in English apply to technical areas as well" [19]. Also, students benefit when they do different kinds of writing in the major [2]. We recently introduced a lab in our CS1 course that exposes students to looking at scholarly articles and using the ACM Digital Library. In our CS2 course, we require significant documentation within the code as well as an accompanying analysis document, which defines the problem, states the input/output specifications, and provides insight into the design of the project. This document forces them to think critically about the choices behind their data structures and algorithms, as well as any relationships between classes used.

1.3 Related Work in Writing Instruction

More writing instruction within the major has been done at other peer universities with great success in recent years. At Furman University, a few lessons using scholarly articles in the first courses in the CS major have "helped reinvigorate students" and "exposed them to reading real research without having them worry about understanding everything in the article" [16]. They also have a mandatory 400-level seminar course that requires students to prepare papers from research journals and give oral presentations. Rollins College requires a Senior Capstone course in their major which exposes students to "primary sources, facilitates a close reading of those sources, and encourages students to reflect on the connections between the reading and their experiences in the major" [14].

The *SIGCSE 2018* panel "Writing in CS: Why and How?" discussed ways that writing can be brought into the major "without massively increasing the load on teachers and students". Strategies were discussed by panelists for assessing results such as "designing clear rubrics" which have shown "clear evidence of improvement in student writing in the context of project reports". Panelist Maxwell mentioned the importance of integrating five different types of writing throughout the major (analytic, code, descriptive, explanatory, and persuasive). And, panelist Minnes gave evidence that students who were required to do weekly reflections during internships showed "greater depth of integrative learning" in their writing by the end [1].

1.4 Writing Instruction in the Discipline

Most computer science professors are not trained in writing instruction. The easiest way for computing professors to facilitate this instruction in an upper level course is to spread it out using scaffolded writing assignments. By doing this, you provide "a variety of instructional techniques used to move students progressively toward stronger writing" [7]. This can be done by emphasizing the prewriting steps as well as multiple drafts. It means giving students feedback and/or doing peer review on a first draft, and then requiring them to revise their work. Another great way to scaffold is to simply require students to submit an annotated bibliography of their sources before actually beginning the writing of their research.

2 DISCUSSION

2.1 Format of an Introductory AI Course

There are many different, reasonable approaches to teaching a first course in AI. In our curriculum, the introductory AI course is offered at the junior level with prerequisites of data structures and discrete mathematics. It is an overview course which tends to explore the breadth of many areas in the field including the history of AI, search techniques, expert systems, machine learning, natural language processing, and genetic algorithms. We typically do one of these topics about every two weeks paired with a lab or assignment. Guest speakers or alumni from the field currently involved in any of the areas of AI covered are also invited into the classroom to complement the instruction.

The AI course textbook is supplemented by popular current books, which interweave many of the ethical issues and challenges in developing a unified theory of artificial intelligence. This past year these included *The Master Algorithm* by Pedro Domingos [6], *The Sentient Machine* by Amir Husain [13], and *Rise of the Robots* by Martin Ford [9]. These short, interesting books are all examples of excellent writing on AI. Reading them can help students improve as a writer and communicator. Selected portions or chapters can easily be assigned, and these books can become excellent sources to later use in their debates or research.

In the first week of the course while pondering what intelligence is, we read Alan Turing' s original 1950 paper "Computing Machinery and Intelligence" where he proposes what is now known as the Turing Test [17]. To make time to facilitate the writing and speech instruction throughout the semester, some of the coverage from the textbook had to be excluded. But, "it is possible to both teach the students the important content of the course and to improve their writing and critical thinking abilities" [10].

2.2 Debates in the AI Classroom

Artificial intelligence in particular raises many ethical questions. On the first class day, I polled them on which of these questions in AI they were most interested in discussing and learning more about. The two topics that generated the most votes were used to form the scope of two in class debates we would do later in the course. The two debate topics selected were:

1. Robots, Automation, and AI will destroy low-wage and middle class jobs the world over in the next 50 years.
2. Autonomous self-driving cars will make driving safer than human-driven cars.

I asked them for suggestions on others they might like to work with in a group, and then used that information to assign them to one of four debate groups. Each of the four groups was then assigned to one side (Yes or No) of one of the two topics. I felt it was important for them to research the point of view and arguments for that particular side. In addition, they had to decide on a role they would play during the debate. For the first topic on automation, the student roles they decided on included Economics Professor, Data Scientist, Software Engineer, and Automation Engineer. The roles for the second debate on self-driving cars were most creative. They included Vice President of Autonomous Vehicles, Chief Technology Officer, Self-Driving Car Engineer for Waymo, and Environmentalist.

2.3 Oral Communication Instruction and Feedback

Students prepared well for the debates and we held them on two separate class days near midterm of the course. Each debate was structured for fifty minutes broken down into an opening statement from each participant on each side, a question and answer session, and a free form discussion with points, counterpoints, and rebuttals. The students not participating in that debate acted as members of the audience who were providing peer review on five components: their opening statement, persuasion, clarity of communication, grasp of subject matter, and teamwork. Students were asked to provide both comments and a numerical evaluation on a five-point scale for each of these components.

Prior to the debates, I taught them about *verbal citations* and their importance when giving a speech or oral presentation. Students are often not aware of the need to incorporate these types of citations [18]. Students were instructed to introduce points or quotes of someone else using phrases like "According to", "As reported by", etc. to clearly demarcate when using quotes. They were reminded "listening to a live debate is a linear process, and it is best to introduce a source before presenting information, so the audience is ready to evaluate the information with the source" [4].

Both debates were a great way to engage students and get them excited about many of the course topics. The discussions were quite lively, and included a nice interaction with questions from both sides as well as the audience members. Students listened and respected arguments from both sides as well as used critical thinking skills to formulate meaningful questions in real time. The students made integrative connections from computing to many other disciplines throughout the debates. They also learned to argue their point of view with someone who may be in disagreement.

Following the debates, I compiled the anonymous peer review comments from the non-debating students. I added up the numerical scores received on each of the five components. This was used to later help provide a holistic evaluation of a group' s performance along with my own comments and those from their peers. As a teacher, I really enjoyed seeing how the debates energized their imaginations and helped prepare them well to think about writing a research paper related to many of the debate topics during the second half of the course.

2.4 Readings and Topics on Artificial Intelligence and Society

Following the midterm break of the course, students in my course were required to pick one of the eight topics below that would become the focus of a final research paper due at the end of the course.

1. Robots – their Rights & Roles in our Future; Place of Robots in Healthcare & Military
2. Dealing with Bias in Artificial Intelligence
3. Glass Cage: The Dangers of Too Much Automation
4. Technological Singularity: When Artificial Intelligence Exceeds Human Intellectual Capacity
5. Books & Newspapers of the Future; Will some be written by Robots (Robo-Journalism)?
6. The Ethical Challenges of Self-Driving Cars
7. Data Mining & Machine Learning for Recommendation Systems or Social Media
8. How Artificial Intelligence and Deep Learning Can Be Used to Evaluate and Create Art; Can Machines Recognize Beauty Itself?

As a class, we spent one day at the library learning more about creating an annotated bibliography with current sources from the last five years for their selected topic. Students practiced finding good journal articles using the academic databases our library had access to and gained more experience in different citation styles. Their annotated bibliography had to have at least four books, as well as four scholarly journal articles. It was good to get them away from always using Google, and searching for articles using academic databases specialized in computing research. Learning to dig, collect, and evaluate sources from different databases is a great skill to develop. By doing so, they also become "part archaeologist and part anthropologist" [12].

2.5 Implementing Writing Instruction and Peer Evaluation

There is no one way of writing well. By requiring students to do an annotated bibliography followed by a first draft of their research paper, you make writing more of a developmental process spread over time. They begin to see how writing is like a puzzle, and that writing and critical thinking go together. The annotated bibliography helps them generate and organize ideas gleaned from the sources they have gathered. Most importantly, it prevents them from procrastinating on their writing, and helps combat writing block. As Garvey indicates, "the writing process students go through is at least as important as the writing products that they eventually produce" [10].

My students were required to hand in their annotated bibliography two weeks before the first draft of the final research paper was due. Each source in their annotated bibliography included a short paragraph annotation, which evaluates the author, audience, relevance to their topic, quality of scholarship, and any connections to prior readings or research. I was able to quickly grade the annotated bibliographies using a rubric which simply checked off whether each of their annotations cited the source properly and included each of the five required components. The annotated bibliography prepared them for the actual writing of their paper, and moved them further along in formulating better quality research.

Like Garvey, I stressed that the final draft "will receive a lower grade (regardless of quality) if they do not hand in a serious effort for the first draft" [10]. After they completed their first draft, I randomly distributed these drafts to other students. Their classmates used a peer evaluation rubric to give them feedback in four areas: format, clarity of argument, use of evidence, and suggestions for improvement. Students appreciated the feedback on their work from their peers, and commented how much they learned from reading the writing of their peers. After returning and reading their peer reviews, I reminded them "expert writers do extensive rewriting" and of the motto to always "write once and edit twice" [2]. If time permits, doing a writing conference with the student on their first draft before they revise it can also be very beneficial.

By scaffolding the writing of the research paper into drafts, it gives students a chance to not only improve their writing, but also improve their grades. As Bean tells his students, "A C paper is an A paper turned in too soon" [2]. This helps them not be afraid of making mistakes, and take more risks in their writing. As students were revising their first draft, I encouraged them to read it aloud or to a friend. I also provided a few helpful *They*

Say / I Say writing templates for strengthening their arguments. I wanted them to see a final draft like an academic conversation with their audience [11].

Grading student writing can be incredibly taxing. To help save time grading their final drafts, I used a rubric broken down into these five categories.

1. Completed First Draft and Peer Review (30%)
2. Use of Evidence / Content from Sources (25%)
3. Organization of Paragraphs, Thesis, Clarity of Argument (25%)
4. Sentence Structure, Spelling, and Mechanics (10%)
5. Format and Documentation Style (10%)

Completing the first draft and peer review process was worth a lot and could be cursory checked off and scored when grading. For the other four categories, I provided an explanation of poor, fair, average, very good, and excellent along with the possible range of points that could be earned in each category. On average, it would take me about half an hour to read, score, and provide comments for a five to seven page paper using this rubric.

3 CONCLUSIONS AND FUTURE WORK

At the end of the course, students were asked four specific questions regarding the writing and debate components of the course. They could respond on a five-point scale from Strongly Disagree (1) up to Strongly Agree (5).

1. As a result of this course, I am better able to write for different purposes and audiences.
2. As a result of this course, I am better able to analyze sources/evidence.
3. As a result of this course, I am better able to use library resources.
4. The in-class debates helped me to improve my communication and critical thinking skills.

All four questions scored an average of at least 4.6 on the five-point scale with my class of 18 students from this past year. Being able to better use library resources scored the highest. Student comments mentioned how helpful the annotated bibliography assignment was in guiding and producing better writing. Their comments also showed they responded well to both debates and really enjoyed them. I highly recommend recording any in class debates to share with colleagues or to document and point out improvements students can make. By recording, you can also provide evidence of oral communication in the classroom for assessment or accreditation purposes.

Overall, the writing and speech instruction of this course gave students invaluable experience, which can benefit them greatly in their future careers.

For those that choose to go on to graduate school, it can boost their confidence for the writing and presenting of a thesis or dissertation. In addition, many of our students now have a GitHub account to display their coding projects. Writing samples would make a great addition here as well. These samples on a GitHub account can serve as a valuable *e-portfolio* for students upon graduation in addition to their resume.

For future research, it would be good to survey both graduating seniors and graduates from the last couple of years to see the long-term benefits of this writing and speech instruction in the major. I would also like to do an initial writing assessment at the start of the course, and compare it to an assessment of their writing in the final research paper. In addition, we hope to start doing more writing instruction in the discipline in courses that lend themselves to writing like Programming Languages, Theory of Computation, and Software Engineering.

References

[1] Philip Barry, Bruce Maxwell, Mia Minnes, and Stephanie Taylor. Writing in cs: Why and how? *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, 2018.

[2] John Bean. *Engaging Ideas: The Professor's Guide to Integrating Writing, Critical Thinking, and Active Learning in the Classroom.* Josey-Bass Publishers, 2011.

[3] Chris Blankenship. Writing is recursive. *Engagement: How We Utilize Literate Practices to Write*, 2020.

[4] Citing sources in an oral presentation 2003-2020. `http://study.com/academy/lesson/citing-sources-for-listener-comprehension.html`.

[5] Criteria for accrediting computing programs 2018-9. `http://www.abet.org/accreditation/accreditation-criteria/criteria-for-accrediting-computing-programs-2018-2019`.

[6] Pedro Domingos. *The Master Algorithm: How The Quest for the Ultimate Learning Machine Will Remake Our World.* Basic Books Publishers, 2018.

[7] Education reform glossary. `http://www.edglossary.org/scaffolding`.

[8] Harriet Fell. Writing across the computer science curriculum. *Proceedings of the 27th ACM Technical Symposium on Computer Science Education*, 1996.

[9] Martin Ford. *Rise of the Robots: Technology and the Threat of a Jobless Future*. Basic Books Publishers, 2016.

[10] Alan Garvey. Writing in an upper-level cs course. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, 2010.

[11] Gerald Graff. *They Say / I Say: The Moves That Matter in Academic Writing*. W.W. Norton & Co Publishers, 2018.

[12] Ed Grisamore. 2020 'class of vision' never saw this coming. *The Macon Telegraph*, 2020.

[13] Amir Husain. *The Sentient Machine: The Coming Age of Artificial Intelligence*. Scribner Publishers, 2018.

[14] Valerie Summet. Reflective writing through primary sources. *Journal of Computing Sciences in Colleges*, 35(4), 2018.

[15] William Taffe. Writing in the computer science curriculum. *Writing Across the Curriculum*, 1(1), 1989.

[16] Andrea Tartaro. Scholarly articles in the introductory computer science curriculum. *Journal of Computing Sciences in Colleges*, 34(2), 2018.

[17] Alan M. Turing. Computing machinery and intelligence. *Mind*, 49:433-–460, 1950.

[18] Understanding when & how to cite sources during a speech. `http://study.com/academy/lesson/citing-sources-for-listener-comprehension.html`.

[19] Henry M. Walker. Writing within the computer science curriculum. *SIGCSE Bulletin*, 30, 1998.

[20] Justin Zobel. *Writing for Computer Science*. Springer London Publishers, 2016.

Shifting Traditional Undergraduate Software Engineering Instruction to a DevOps Focus*

Brian T. Bennett
Department of Computing
East Tennessee State University
Johnson City, TN 37614
bennetbt@etsu.edu

Abstract

Classical Software Engineering education often includes traditional methodologies that do not adequately describe today's industry practice. DevOps is a culture that promotes fast delivery, continuous feedback, and an environment of learning. Its non-linear path requires a shift in software engineering pedagogy. This case study describes the redevelopment of a second-semester course in software engineering to focus on DevOps principles. The study evaluates student performance using formative and summative assessment through a team project tracked throughout the semester and final exam results. Results indicated that students developed DevOps skills during the course, but may have needed more reinforcement of some traditional Software Engineering topics.

1 Introduction

Software Engineering (SE) continues to evolve as new and diverse technologies become pervasive. The waterfall model emerged in the late 1960s, describing a flow from one phase of development to another. The waterfall methodology was a natural fit for the time, considering the long software development cycles caused by technological constraints that existed in the 1970s. As technology

Software Engineering 1	Software Engineering 2
1. Introduction to Software Engineering 2. Requirements Gathering, Analysis, and Specification 3. Software Architecture, Detailed Design, and Design Patterns 4. Implementation and Refactoring 5. Unit, OO, and Integration Testing 6. Software Maintenance 7. Introduction to Project Management 8. Software Engineering Ethics	1. Planning and Estimation 2. Configuration Management 3. Working with Teams 4. Metrics and Quality Assurance 5. DevOps Overview 6. Emerging Architectures: Microservices, Serverless, and IoT 7. System and Acceptance Testing 8. Security and Social Impacts of Software Engineering

Table 1: Original Software Engineering Curriculum

improved, software engineers strove to make methodologies more agile [2] so customers could receive a faster return-on-investment. Today, the extensive use of cloud-based technology has enabled software developers to continuously produce and deliver software, leading to the DevOps paradigm [5]. DevOps relates to the development and operations tasks involved in the rapid delivery of software in a way that breaks down the silos associated with software development and system provisioning. SE education requires a shift in its focus to align with this paradigm shift in the industry.

Traditional SE instruction has focused on classical waterfall development phases and project management. Some university programs offer only a single overview course in SE, potentially limiting their ability to expand on various topics. Such courses focus more on classical waterfall model phases than on newer practices that graduates will see in the industry. University programs that offer two SE courses have a better opportunity to separate concepts: the first course could focus on development phases, ethics, and life cycle models. The second could focus more on project management and newer development methodologies like DevOps.

Previous work in this area proposed ways to incorporate DevOps into existing coursework, without assessing effectiveness [3]. This paper describes a new curriculum for a second SE course that shifts the focus to DevOps practices. While the classic material remains, the new curriculum uses DevOps as the guiding methodology for course content and application. The curriculum was implemented and assessed during the Fall 2019 semester, and the results of the various learning outcomes were assessed.

2 Evolving Software Engineering Curriculum

This paper assumes a two-course sequence in Software Engineering. Table 1 shows the major topics covered in previous iterations of each course. Software

Engineering 1 discussed each main phase of the Waterfall Model: Requirements, Design, Implementation, Testing, and Maintenance. The first course also introduced project management and software engineering ethics. Software Engineering 2 provided an in-depth discussion of project management: planning, estimation, configuration management, working with teams, and using metrics for quality assurance. The second half of the second course included a selection of contemporary topics like DevOps (including continuous integration and continuous delivery), emerging architectures, issues with testing, and the security and social impacts of Software Engineering [3]. Because of the emergence and prevalence of DevOps in today's industry practice, a paradigm shift in the software engineering curriculum has become necessary.

Software Engineering education researchers have begun to investigate how to incorporate DevOps practices into the curriculum. Kuusinen and Albertsen [6] suggested collaborating with industry partners to teach DevOps and continuous delivery practices. The authors included a DevOps-focused set of learning outcomes that dealt directly with the automation of the DevOps pipeline. By using both examination and student survey results, the assessment found that improvements were necessary for explaining Infrastructure as Code, but students generally understood the DevOps pipeline. Also, results showed a gap between student understanding of DevOps as it relates to other agile methodologies.

Christensen [4] listed several challenges for teaching DevOps with traditional SE curricula. These challenges included a lack of instructor experience with DevOps and the fact that DevOps crosscuts many traditional courses. Christensen [4] also noted challenges related to DevOps being a skills-heavy discipline and added that creating a realistic environment for students to experiment could be difficult. To address these challenges, the author proposed a seven-week course that included a tool-heavy curriculum. Therefore, the author focused on the details of using specific tools like Docker to extend an existing project continuously. Because of this focus, the study may have missed some of the team dynamics required in a DevOps environment.

Bennett and Barrett [3] took a more robust team-based approach when discussing a framework for incorporating DevOps into an existing SE curriculum. Their work suggested three significant steps to help instructors begin to introduce DevOps. First, the framework suggested that instructors encourage DevOps understanding by placing students from various Computing programs into project teams. Because not all institutions require SE across all Computing programs, this suggestion might prove problematic. Second, the framework encouraged the introduction of emerging architectures like microservices, serverless architectures, and the Internet of Things. Introducing these architectures without requiring their use might limit the effectiveness of the instruction.

Software Engineering 2	
Learning Objective	Abbreviation
1. Project Management in a DevOps World	1. SPM
2. The First Way: Maximizing Flow	2. 1W
(a) Planning and Estimation	(a) PE
(b) Configuration Management and Continuous Integration	(b) CMCI
(c) Testing Review and Automated Testing	(c) TG
(d) System and Acceptance Testing	(d) SAT
3. The Second Way: Principles of Feedback	3. 2W
(a) Metrics and Quality Assurance	(a) MET
(b) Reviews and Inspections	(b) RI
(c) Create and Analyze Telemetry	(c) CAT
4. The Third Way: Continual Learning and Experimentation	4. 3W
(a) Emerging Architectures	(a) EA
5. DevSecOps	5. DSO
(a) Security Design Patterns	(a) SDP
(b) Secure Use Cases	(b) SUC

Table 2: Updated Software Engineering Curriculum Using DevOps

Third, the framework encouraged instructors to require project planning and management activities within team projects using Kanban boards. Without proper guidance, a student might miss the point of using the tools to encourage project progression.

3 Proposed DevOps Curriculum

3.1 Learning Objectives

Table 2 shows an evolved curriculum in the second SE course based in part on the DevOps Handbook [5]. This curriculum creates a DevOps thread throughout the semester, using the "Three Ways" that help define the paradigm. The First Way discusses continuous integration, continuous builds, and continuous testing to move the software to the customer quickly. The Second Way allows for feedback from deployments to inform developers about how to improve the product and the process. The Third Way encourages developers to try new things because the DevOps framework has built-in safety features that prevent failures from becoming catastrophes. Each of these "Three Ways" allows for a different means of exploring traditional SE topics.

3.2 Project Objectives

Because DevOps requires a team structure, a team project is an integral part of the course. Creating a DevOps scenario involved a simple project that required students to use DevOps techniques across four sprints. The project's objectives

appear in Table 3. Each sprint evaluated four particular items: *CALMS evaluation*, *Code Development & Management*, *Work In Process (WIP)*, and the *Use of Scrum Practices*. CALMS is an acronym used to evaluate DevOps implementation based on Culture, Automation, Lean practices, Measurement, and Sharing [1]. Students reflected on their use of DevOps at each stage in the process by writing a brief CALMS report. Grades for *Code Development and Management* assessed students' use of BitBucket to manage their code repositories. *WIP* using Trello was evaluated based on a constant left-to-right movement of work, from inception to completion. The team's effective use of standard Scrum practices to complete the project across sprints provided the basis for evaluating the *Use of Scrum Practices*.

Course instructors evaluated additional practices at a more granular level, based on topics covered in class. API Development occurred during Sprints 1 and 2 and involved creating a microservices architecture that interacted with a database. Sprint 3 focused on creating a testing environment that the team could automate while also collecting other telemetry metrics about the software itself. The final sprint involved a comprehensive retrospective of the entire process.

<table>
<tr><th>Sprint 1</th><th>Sprint 2</th><th>Sprint 3</th><th>Sprint 4</th></tr>
<tr><td colspan="4">CALMS Evaluation</td></tr>
<tr><td colspan="4">Code Development & Management</td></tr>
<tr><td colspan="4">Work In Process (WIP)</td></tr>
<tr><td colspan="4">Use of Scrum Practices</td></tr>
<tr><td colspan="2">API Development</td><td>Metrics & Testing</td><td>Retrospective</td></tr>
</table>

Table 3: Project Objectives Across Sprints

4 Evaluation

An evaluation was performed across two sections of the Software Engineering 2 course in Fall 2019, with 31 students and six project teams. Although the population size is small, this study gives an initial idea of effectiveness and highlights areas where adjustments are necessary. The assessment was both summative and formative. Summative assessment used the final exam to evaluate all 15 learning objectives. Formative evaluation assessed project objectives throughout the four sprints during the semester.

4.1 Learning Objectives

Each learning objective result comes from each student's average percentage for that learning objective on the final exam. These scores determine if a student exceeds expectations (scores at or above 90%), meets expectations (scores between 70% and 89.99%), or does not meet expectations (scores below 70%). Aggregate counts for each learning objective, as assessed by the final exam, appear in Figure 1 as red (does not meet), yellow (meets), or green (exceeds).

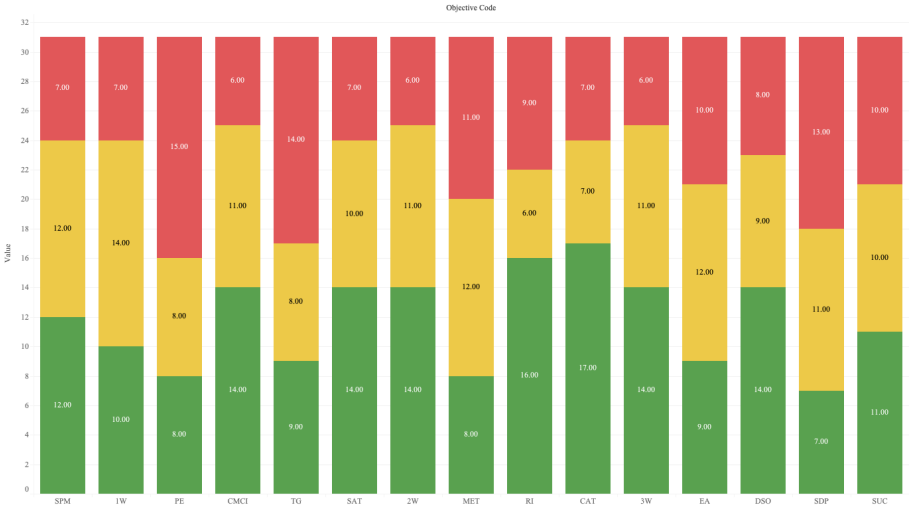


Figure 1: Learning Objective Results from Software Engineering Final Exam

Figure 2 shows the average percentage scores for each learning objective, clustered by similarity. The objectives showing the lowest performance include: *Testing Review and Automated Testing* (TG: 45% Does not meet, 61% average score), *Security Design Patterns* (SDP: 42% Does not meet, 69% average score), and *Planning and Estimation* (PE: 48% Does not meet, 72% average score). DevOps topics were among the objectives showing the best performance, including: *Creating and Analyzing Telemetry* (CAT: 77% meets or exceeds, 94% average score), *Configuration Management and Continuous Integration* (CMCI: 81% meets or exceeds, 92% average score), and *The Second Way* (2W: 81% meets or exceeds, 92% average score). Other similar objectives showing positive results include: *Reviews and Inspections* (RI: 88% average score), *The Third Way* (3W: 87% average score), *The First Way* (1W: 86% average score), *System and Acceptance Testing* (SAT: 84% average score), and *DevSecOps* (DSO: 82% average score). *Software Project Management* (SPM),

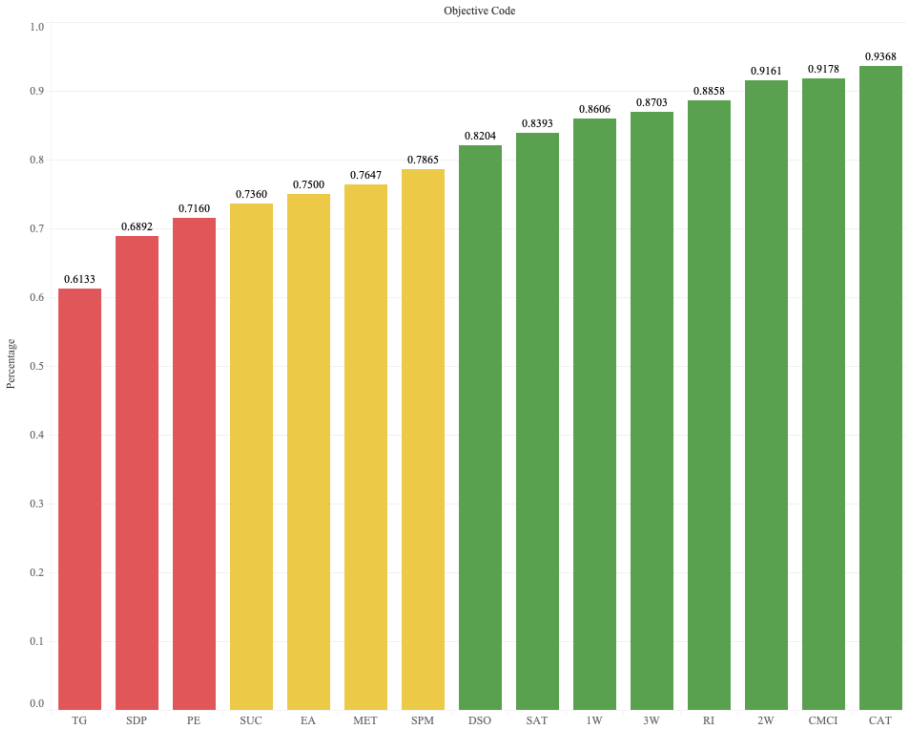


Figure 2: Learning Objective Percentages from Software Eng. Final Exam

Metrics and Quality Assurance (MET), *Emerging Architectures* (EA), and *Secure Use Cases* (SUC) showed average scores between 79% and 74%.

4.2 Project Objectives

Project objectives were assessed for each of the six project groups. Each group received a letter grade that converted to a grade-point value on a four-point scale for each project objective. For example, A- converted to 3.7, B+ to 3.3, etc. Figure 3 shows the averages for each objective by sprint, while Figure 4 shows the averages for each project objective overall.

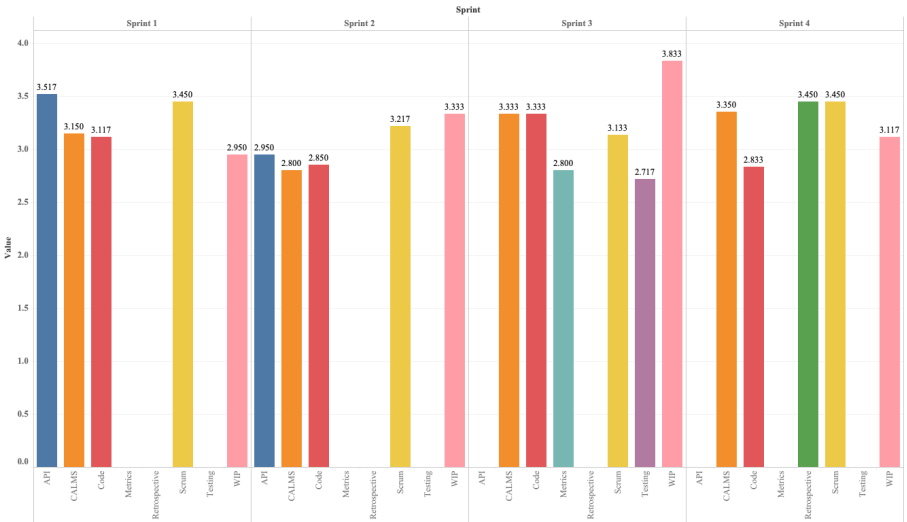


Figure 3: Project Objectives by Sprint

The four project objectives that crossed sprints were *CALMS*, *Code*, *WIP*, and *Scrum*. *CALMS* (overall average 3.16), showed a dip in Sprint 2, but a strong recovery by Sprint 4. This result indicates that students became better-adapted to the DevOps framework over time and were better able to reflect as time progressed. *Coding* (overall average 3.03), increased during Sprint 3, but fell to its lowest level in Sprint 4. This result likely indicates that most teams spent Sprint 4 addressing issues that occurred in previous sprints, and less product progression occurred. Tracking *WIP* (overall average 3.31) showed an upward trajectory that peaked during Sprint 3, with a slight decline in Sprint 4. This decline indicates that students became more accustomed to tracking their work over time using DevOps principles. It also agrees that the final sprint had less product progression. Finally, *Use of Scrum* (overall average 3.31)

remained nearly constant throughout the project. This consistency shows that students had a base understanding of the Scrum framework from the Software Engineering 1 course.

Other objectives showed some interesting trends. Work on *API development* (overall average 3.23) was strong during Sprint 1 but fell during Sprint 2. This decrease indicates that students had additional difficulties because they had to begin using their APIs during the second sprint. *Testing* (overall average 2.72) was the lowest-scoring objective. Many students struggle with setting up unit tests, but the added responsibility of automating those tests likely decreased this score. *Metrics* (overall average 2.80) also proved to be a problem area for students, who had to learn, implement, and use metrics during the semester. These problems indicate the need for additional practice with using metrics. The project *Retrospective* (overall average 3.45) was the highest-scoring objective, which indicates that students were good at honestly reflecting on the process throughout the semester.

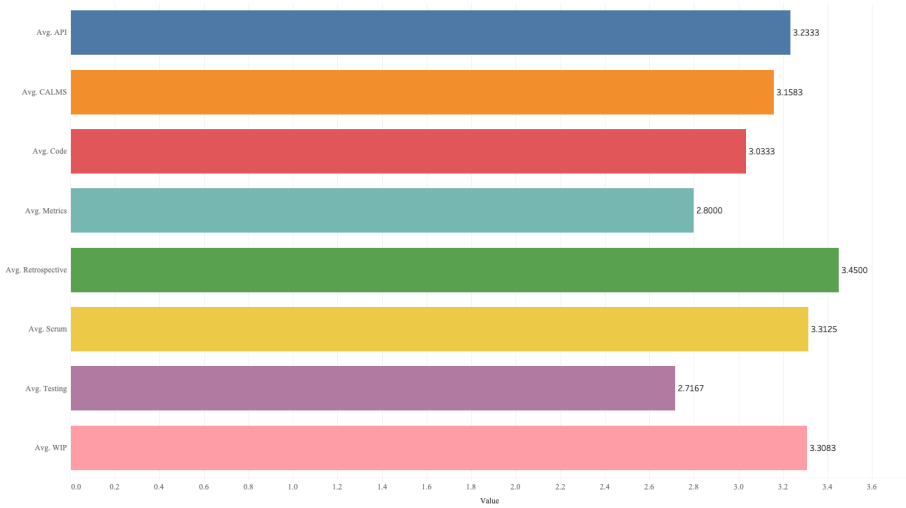


Figure 4: Project Objective Averages

5 Conclusion

This study shifted the focus of the second course in software engineering from traditional topics to DevOps. The experimental offering in Fall 2019 showed that some DevOps topics were well-received, including The Three Ways, particularly topics surrounding the Second Way (*Reviews and Inspections* and

Creating and Analyzing Telemetry). The use of Scrum practices remained constant throughout, but other classic software engineering topics were lower than expected. *Testing* seems to be a problem both in practice and in content retention and needs reinforcement in the curriculum through exercises or other hands-on assignments. *Metrics* was another topic that seemed problematic for students in both content retention and practice. Because most students had never used metrics, continuous reinforcement of their importance would improve student understanding. Instructors who adopt this methodology should ensure that DevOps topics do not overshadow important classical topics like metrics and testing. To ensure maximum effectiveness, a course in DevOps must balance content delivery with practical application to ensure student success.

References

[1] The calms model of devops. `https://www.devopsgroup.com/insights/resources/diagrams/all/calms-model-of-devops/`.

[2] Manifesto for agile software development. `https://agilemanifesto.org/`.

[3] Brian T. Bennett and Martin L. Barrett. Incorporating devops into undergraduate software engineering courses: A suggested framework. *J. Comput. Sci. Coll.*, 34(2):180–187, December 2018.

[4] Henrik Bærbak Christensen. Teaching devops and cloud computing using a cognitive apprenticeship and story-telling approach. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE ' 16, page 174–179, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Gene Kim, Patrick Debois, John Willis, and Jez Humble. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press, 2016.

[6] Kati Kuusinen and Sofus Albertsen. Industry-academy collaboration in teaching devops and continuous delivery to software engineering students: Towards improved industrial relevance in higher education. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET ' 19, page 23–27. IEEE Press, 2019.

Skin Cancer Detection Using Convolutional Neural Networks*

Mattia Galanti[1], *Gilliean Lee*[2], *Joshua John*[2]
[1] *School of Computing*
Clemson University
Clemson, SC 29634
{mattia.galanti}@clemson.edu
[2] *Department of Mathematics & Computing*
Lander University
Greenwood, SC 29649
{glee,joshua.john}@lander.edu

Abstract

This paper focuses on the development of classifiers capable of detecting a skin cancer(s) given dermoscopic images. The dataset used for the training is a part of the 2019 ISIC Challenge, and consists of more than 25,000 labeled dermoscopic images. Specifically, classifying dermoscopic images accounts for nine different diagnostic categories: melanoma, melanocytic nevus, basal cell carcinoma, actinic keratosis, benign keratosis, dermatofibroma, vascular lesion, and squamous cell carcinoma, some of which are benign. We have developed classifiers –a binary classifier and a multiclass classifier –on the Google Cloud Platform using Convolutional Neural Networks (CNNs). To prevent the classifiers from overfitting and to achieve higher accuracy even with the smaller training data size, we use image data augmentation. The binary classifier achieved an accuracy of 79% with 220 epochs of training, and the multiclass classifier' s accuracy is 72% with 200 epochs.

1 Introduction

Skin cancer, more than any other type of cancer, causes more death than heart disease in wealthy countries [3] and is increasingly the most common form of cancer in the United States [14]. In 2019, it is estimated that 7,230 deaths are attributed to melanoma alone. However, skin cancer is also one of the most treatable types of cancer. The five-year survival rate for melanoma patients is 99% if it is detected and treated before it spreads to the lymph nodes; thus, early detection is crucial. The warning signs of skin cancer include changes in size, shape, and color of moles or skin lesions [14]. Despite all this, it is very unlikely that during a routine check-up a skin lesion will be classified as a possible skin cancer. A fair number of patients diagnosed with melanoma had visited their regular doctors in the two years or so preceding the diagnosis [21]. In most cases, the melanoma was not diagnosed, and some patients were incorrectly cleared after having reported concern about skin lesions [21]. Over the last decade, the growing focus has been put on identifying screening obstacles in order to implement more tailored strategies that promote early detection efforts [21]. Locating skin cancer is not generally a part of the main practice for essential care. The chance of detecting a clinically imperative early skin cancer is low, thus practitioners are often not able to detect skin lesions at their earliest stages of development. Yet physicians never face the consequences of missing an early skin cancer diagnosis. Further, there are not many specialists who have the expertise to suitably diagnose potentially cancerous lesions [21].

Skin cancer detection is being revolutionized by deep learning. Deep learning is a subset of machine learning and a specific form of artificial neural network (ANN), similar to the multilayered human cognition system [12]. Deep learning methods have changed the investigation of natural images and videos, and similar examples are emerging with the investigation of biomedical data. Deep learning has been used to classify lesions and nodules, and to localize organs, regions, and landmarks [4]. In numerous areas of medicine, deep learning technology is used widely. One area of its application includes breast cancer detection. Here, traditional detection methods are often expensive, painful, and inaccurate with respect to measuring the location and size of the tumor. To overcome these deficiencies, neural network models for early breast cancer detection have been used [1]. Additionally, research on leukemia classification in later years has been primarily based on computer vision techniques. Machine learning techniques, such as K-means clustering, have been implemented in order to detect and classify blood cells in images. There are two advantages to using Convolutional Neural networks (CNNs): not only do they diminish the processing time by allowing us to skip most of the pre-processing steps, but they also are capable of extrapolating features that are more robust than the customary statistical features [19]. In recent years, deep learning has been

revolutionizing liver lesion segmentation, which is a fundamental step in the diagnosis of liver cancer. While manual segmentation is time-consuming and prone to error, scholars have designed a two-step U-Net approach in order to automatize such segmentation, and they have reported a very high accuracy rate (Dice score above 0.94) on their test data [7]. The Dice score, or Sørensen–Dice coefficient, is a common metrics for the evaluation of segmentation tasks in medical imaging [2]. All this is only a portion of the vast progress made in recent years in the biomedical field with regard to the detection of tumors.

Our research aims to develop a binary skin cancer classifier and a multi-class classifier using deep learning technology by training them with dermoscopic skin images from the 2019 ISIC Challenge dataset [9]. We designed and implemented the classifiers as CNNs using TensorFlow [18], and planned on reducing anticipated overfitting by expanding training images using an image augmentation technique.

2 Related Work

Over the years, new techniques that seek to improve the accuracy of skin cancer diagnosis and detection have emerged in the biomedical world. In 2012, Sheha et. al. [16] developed a system based on the use of two different multilayer perceptron (MLP) classifiers. The highest classification accuracy was achieved by using the traditional MLP classifier over an automatic one, with 92% and 76% classification accuracies, respectively [18].

Sharma and Srivastava [15] have proposed the use of Back-propagation neural networks (BNNs) and Auto-associative neural networks (AANNs) for accurate prevention and diagnosis of skin cancers. They developed a BNN-based classifier that achieved a 90.2% overall accuracy with three hidden layers of 40, 25 and 10 neurons in each hidden layer. In this case, the high number of neurons per hidden layer helped reduce the probability of overfitting. They also developed an AANN-based classifier that achieved overall accuracy of 81.5

It should be stated that the type of image you work with in some cases affects the performance of a certain diagnostic method [20]. In the past it has been observed that using dermoscopic image sets improves the accuracy with which CNNs classify malignant cases, while CNNs that were trained with close-up images were better at diagnosing benign cases [20]. Dermoscopic images are obtained through dermoscopy, an imaging technique which removes the reflection of the skin on the surface [8]. Thus, compared to close-up images, dermoscopic imaging provides an enhanced visualization of the skin [8].

3 Data Preparation

The 2019 ISIC Challenge dataset [10] contains 25,331 dermoscopic images for training with labels across 8 different categories. A part of the dataset is the HAM10000 dataset. Of these images, we use 80% as a training set and the rest as a validation set. All the images in the dataset come with different sizes, so we apply random cropping with a fixed resolution of 256x256 pixels. Random cropping allows our models to generalize better, for what we want them to learn is not always completely visible in the images or the same scale in our training data [17]. We also applied image augmentation technique to artificially expand the dataset and to increase the accuracy with fewer images. Specifically, we use some properties of the ImageDataGenerator class of Keras, such as rescale, rotation, shear range and zoom range for image augmentation. During the whole training, we had to deal with the fact that the initial dataset was heavily imbalanced, and the number of images in some classes was greater than in the rest, as reported in Table 1. For instance, for the Melanocytic Nevus class we have 12,875 images, almost half of the entire dataset, while the class Dermatofibroma consists of just 239 images. Precisely, four of these categories constitute 92% of the available data, while the rest make up only 8%. The ratio of malignant and benign images is about 1 to 2 in the training dataset. We organized the image data in subfolders according to the type of skin cancer or lesion to which they belong, so that the ImageDataGenerator class can make use of it.

Table 1: Number of Samples for Each Class in the Training Dataset

Diagnostic Class	Number of Images	Percentage
Melanoma - MEL	4522	17.9%
Melanocytic Nevus - NV	12875	50.8%
Basal Cell Carcinoma - BCC	3323	13.1%
Actinic Keratosis - AK	867	3.5%
Benign Keratosis - BKL	2624	10.4%
Dermatofibroma - DF	239	0.9%
Vascular Lesion - VASC	253	1%
Squamos Cell Carcinoma - SCC	628	2.4%
Total	25,331	/

4 Training and Testing

We built two models - a binary classifier for malignant and benign cases, and a multiclass classifier to classify the lesions and types of skin cancer reported in Table 1. Initially, we developed and trained our models on a local computer in Jupyter Notebook [11] environment, but the time to train the model by going through the whole training set just one time, also known as an epoch, was too long due to lack of computing power. Usually training a model takes well more than a hundred epochs, so training models on a local computer turned out to be an infeasible solution. Then, we updated our models and moved the development environment to Google Colaboratory [5] where users can develop and execute Python code in Jupyter Notebook environment with cloud computing free of charge. Still, the code in Google Colaboratory couldn' t run for more than about 30 minutes due to timeout.

As a consequence of receiving the Google Cloud Platform (GCP) Research Grant, we were able to set up virtual machines with multiple GPUs and train without time limit. This way, the time to run a single epoch went from about three and a half hours on a local machine to 4 minutes on a virtual machine with GPUs.

All training and tests ran using different configurations of GPUs to be able to decrease the time to run epochs as much as possible. For our binary classifier, we initially used a virtual machine with two NVIDIA Tesla T4 GPUs with 64GB of video memory, but in order to run more epochs in less time we took a step forward and upgraded to eight NVIDIA Tesla K80 GPUs with 128GB of video memory. For our multiclass classifier we used a virtual machine with two NVIDIA Tesla T4 GPUs and 64GB of video memory, without upgrading. In the multiclass classifier, the upgrade was unnecessary because we didn' t' need to train as many epochs since overfitting happened at an earlier epoch.

Convolutional Neural Networks (CNNs) emerged from the study of the brain' s visual cortex and are known to perform well in computer vision tasks such as image classification [6]. A CNN can be defined as a Deep Learning model which can analyze images, define weights and biases for different objects in those images, and be able to distinguish one from another. CNNs are made up of different layers, of which the convolution and pooling layers are very important. The convolution layers transform the input image so as to extract features from it. The pooling layers are used to reduce the size of the input image.

So as to achieve the best training model, we tried different configurations, changing the number of neurons per layer and the number of convolutional / pooling layers. The models that are presented are the ones that have obtained the best performances. Our CNN models are composed of a Sequential layer that includes four pairs of convolutional and pooling layers, a Flatten layer that

shapes 2d image data into single dimensional data, and a Dense layer which consists of densely connected layers. There is a Dropout layer, which applies the dropout regularization technique by randomly disabling neurons at certain percent during training to avoid overfitting in artificial neural networks, just before the dense layer. The dropout layer has a 50

Regarding the number of filters per convolution layer, we tried different combinations to see how the model would respond. There is no standardized way to choose that, so most of the time the combination depends on the data available and the results obtained. In the end, the number of filters in the four convolutional layers set to 32, 64, 64, and 128 obtained the best results. The configuration of our CNN is shown in Figure 1.

The Adam optimizer, which is known to converge to the global optimum quickly, is one of the most popular optimizers in deep learning [6], and was used for training our models. The last layer of the model is an output layer with the SoftMax activation function for our multi-class classifier, or with the Sigmoid activation function for our binary classifier. The design of the models is shown in Figure 1.

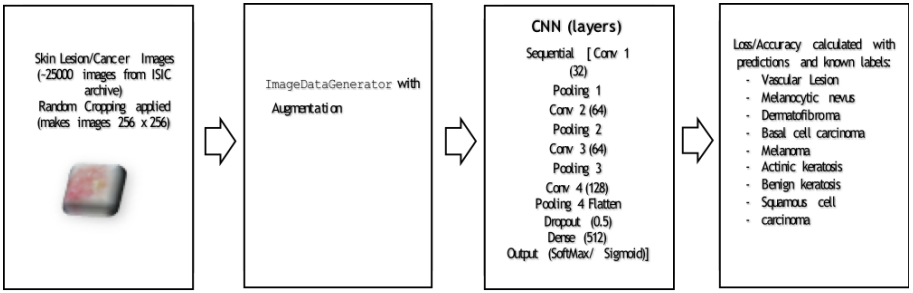


Figure 1: The Training Process of the Skin Cancer Classifier

With the training and test image data ready, we trained the models with the data, considering the loss and the accuracy, which are indicators of how the training process is going. We used a split ratio of 80:20 of training and validation data for both our binary classifier and multiclass classifier. After each epoch, the accuracy and loss of the models were logged and graphed. Accuracy is the portion of correct predictions, and loss is a measure of a model' s performance. We use the Cross-Entropy loss function for our models. More precisely, the Binary Cross-Entropy loss function for our binary classifier, and the Categorical Cross-Entropy one for out multiclass classifier. Both loss functions measure the performance of a model whose output is a probability value between 0 and 1 [13]. We have to use slightly different Cross-Entropy loss

functions because, based on the nature of our classifiers, we may have more than two classes. We run batches of 25 epochs to account for overfitting and we save it every 50 epochs of training to prevent data loss due to a connection timeout. After each epoch, training accuracy and validation accuracy are recorded. Training accuracy is the percent of correct predictions by the model being trained on the training data, and validation accuracy is calculated from the validation data. We use the validation set to compare the two obtained accuracies in order to correctly evaluate the results and observe the performance of the classifier; Figures 2 and 3 show training and validation accuracy curves of the models as training takes place.

Training has to stop when overfitting happens. Overfitting occurs when the model performs well on training data, but poorly on data that were never used during the training. We consider the occurrence of overfitting as the gap between training accuracy and validation accuracy gets bigger. In Figure 2, approximately at 100 epochs, overfitting is observed with the training accuracy at about 72%. In Figure 3, overfitting is not observed, even though the accuracy is plateaued at 79% at around 200 epochs.

5 Result

Our binary classifier' s accuracy approaches 79% with 220 epochs of training, and our multiclass classifier' s accuracy approaches 72% with 100 epochs, as described in Table 2. Evaluating the accuracy curves for each model, we observed that the multiclass classifier began to overfit around at 100 epochs, while the binary classifier performed well without overfitting until it plateaued, as shown in Figures 2 and 3.

Table 2: Classifier Accuracy

Classifier Type	Validation Accuracy
Binary (Malignant or not)	79%
Multiclass (Types of skin cancer/lesion)	72%

Figures 4 and 5 show the confusion matrix, and present the prevalence of misclassifications on each label for the classifiers. Basically, the bright blocks in the diagonal show high accuracy, and MEL, BCC, BKL and NV show high accuracy. The number of images per diagnostic category in Table 1 helps us understand why some of these present more misclassifications than others. As mentioned above, the initial dataset was heavily imbalanced, therefore affecting those diagnostic categories with fewer images. As a matter of fact, NV,

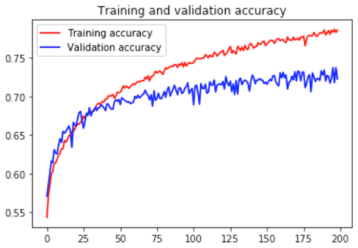


Figure 2: Accuracy Curve of the Skin Cancer Multiclass Classifier.

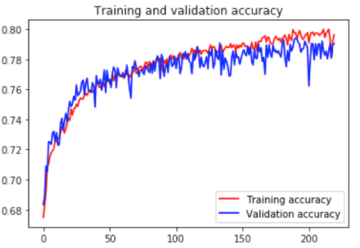


Figure 3: Accuracy Curve of the Skin Cancer Binary Classifier.

BCC, MEL, and BLK were likely to be classified correctly than others in the multiclass classifier due to the greater number of images in the training set.

Observing Figure 4 and 5, we notice from the confusion matrices that the number of true positives is quite low compared to the true negatives. This is another consequence of the heavily imbalanced training set and validation set. The ratio of images representing malignant skin cancers/lesions to benign ones is 8,473 to 17,058, roughly 1 to 2.

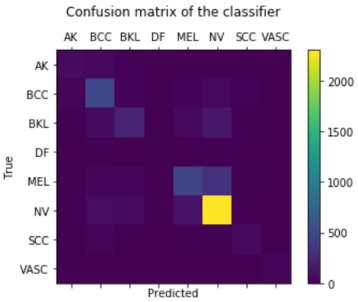


Figure 4: Confusion Matrix of the Multiclass Classifier

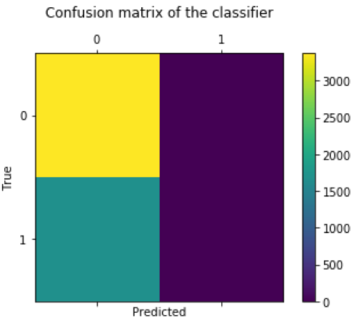


Figure 5: Confusion Matrix of the Binary Classifier

6 Conclusion

In this paper, we have presented a deep neural network using CNNs that we designed and developed to detect skin cancer and lesions. The results are promising with an accuracy of 72% (multiclass classifier) and 78% (binary classifier). It also confirmed that the accuracy of the model is heavily influenced by the size and ratio of the classes in the training set.

Future studies exploring new learning models, such as transfer learning that trains an existing well-performing model rather than from the scratch, may be needed to improve accuracy. A focus on improving the pre-processing, thus making it more effective so as to handle the imbalanced data during the skin lesion/cancer analysis, is another area of future research.

We believe research like ours plays an important role in the early detection and treatment of skin cancers thanks to its simplicity and accuracy. Furthermore, this kind of research can aid in developing other tools, such as apps for physicians and patients to use, further decreasing the time required to arrive at a correct diagnosis and, subsequently, expediting the treatment of what can be a debilitating disease.

References

[1] S. A. AlShehri and S. Khatun. Uwb imaging for breastr cancer detection using neural network. *Progress In Electromagnetics Research C*, 7:79–93, 2009. doi:10.2528/PIERC09031202, Last accessed on 2020-04-11.

[2] Jeroen Bertels, Tom Eelbode, Maxim Berman, Dirk Vandermeulen, Frederik Maes, Raf Bisschops, and Matthew B. Blaschko. Optimizing the dice score and jaccard index for medical image segmentation: Theory and practice. *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2019. doi:10.1007/978-3-030-32245-8_11, Last accessed on 2020-05-24.

[3] Sam Blanchard. Cancer is 'overtaking heart disease as wealthy countries' biggest killer'., 2019. "https://www.dailymail.co.uk/health/article-7421917/Cancer-overtaking-heart-disease-wealthy-countries-biggest-killer.html", Last accessed on 2020-02-10.

[4] Travers Ching, Daniel S. Himmelstein, Brett K. Beaulieu-Jones, Alexandr A. Kalinin, Brian T. Do, Gregory P. Way, Enrico Ferrero, Paul-Michael Agapow, Michael Zietz, Michael M. Hoffman, Wei Xie, Gail L. Rosen, Benjamin J. Lengerich, Johnny Israeli, Jack Lanchantin, Stephen

Woloszynek, Anne E. Carpenter, Avanti Shrikumar, inbo Xu, Evan M. Cofer, Christopher A. Lavender, Srinivas C. Turaga, Amr M. Alexandari, Zhiyong Lu, David J. Harris, Dave DeCaprio, Yanjun Qi, Anshul Kundaje, Yifan Peng, Laura K. Wiley, Marwin H. S. Segler, Simina M. Boca, S. Joshua Swamidass, Austin Huang, Anthony Gitter, and Casey S. Greene. Opportunities and obstacles for deep learning in biology and medicine. *The Royal Society*, 2018. doi:10.1098/rsif.2017.0387, Last accessed on 2020-03-19.

[5] Google. Google colaboratory. "https://colab.research.google.com/notebooks/intro.ipynb", Last accessed on 2020-07-09.

[6] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. OReilly, 2019. Last accessed on 2020-06-16.

[7] Xiao Han. Mrbased synthetic ct generation using a deep convolutional neural network method. *Medical Physics Journal*, 2017. doi:10.1002/mp.12155, Last accessed on 2020-02-20.

[8] Zalaudek I., Argenziano G., Di Stefani A., Ferrara G., Marghoob A.A., Hofmann-Wellenhof R., Soyer H.P., Braun R., and Kerl H. Dermoscopy in general dermatology. *Karger*, 212:7–18, 2006. doi:10.1159/000089015, Last accessed on 2020-06-19.

[9] ISIC. Isic 2019. "https://challenge2019.isic-archive.com", Last accessed on 2020-04-15.

[10] ISIC. Training data. "https://challenge2019.isic-archive.com/data.html", Last accessed on 2020-02-10.

[11] Project Jupyter. Project jupyter. "https://jupyter.org/about", Last accessed on 2020-07-09.

[12] June-Goo Lee, anghoon Jun, Young-Won Cho, Hyunna Lee, Guk Bae Kim, Joon Beom Seo, and NamkugS Kim. Deep learning in medical imaging: General overview. *Korean Journal of Radiology*, 2017. doi:10.3348/kjr.2017.18.4.570, Last accessed on 2020-03-12.

[13] Nielsen and Michael A. Improving the way neural networks learn. "https://neuralnetworksanddeeplearning.com/chap3.html", Last accessed on 2020-05-18.

[14] American Academy of Dermatology. Skin cancer. "https://www.aad.org/media/stats/conditions/skin-cancer", Last accessed on 2020-02-10.

[15] Deepti Sharma and Swati Srivastava. Automatically detection of skin cancer by classification of neural network. *International Journal of Engineering and Technical Research*, 4:15–18, 2016. issn:2321-0869, Last accessed on 2020-05-15.

[16] Mariam A. Sheha, Mai S.Mabrouk, and Amr Sharawy. Automatic detection of melanoma skin cancer using texture analysis. *International Journal of Computer Applications*, 42:22–26, 2012. doi:10.5120/5817-8129, Last accessed on 2020-05-15.

[17] Ryo Takahashi, Takashi Matsubara, and Kuniaki Uehara. Data augmentation using random image cropping and patching for deep cnns. *Journal of LaTeXClass Files*, 14:1–11, 2015. "https://arxiv.org/pdf/1811.09030v1.pdf", Last accessed on 2020-06-19.

[18] TensorFlow. Tensorflow. "https://www.tensorflow.org/", Last accessed on 2020-07-08.

[19] T. T. P. Thanh, Caleb Vununu, Sukhrob Atoev, Suk-Hwan Lee, , and Ki-Ryong Kwon. Leukemia blood cell image classification using convolutional neural network. *Internation Journal of Computer Theory and Engineering*, 10:54–58, 2018. doi:10.7763/IJCTE.2018.V10.1198, Last accessed on 2020-06-13.

[20] Philipp Tschandl, Cliff Rosendahl, and Bengu Nisa Akay. Expert-level diagnosis of nonpigmented skin cancer by combined convolutional neural networks. *JAMA Dermatology*, 155:58–65, 2019. doi:10.1001/jamadermatol.2018.4378, Last accessed on 2020-04-12.

[21] Richard C. Wender. Barriers to effective skin cancer detection. *American Cancer Society Journals*, 1995. "https://acsjournals.onlinelibrary.wiley.com/doi/epdf/10.1002/1097-0142%2819950115%2975%3A2%2B%3C691%3A%3AAID-CNCR2820751412%3E3.0.CO%3B2-G", Last accessed on 2020-03-12.

Hurricanes and Pandemics: An Experience Report on Adapting Software Engineering Courses to Ensure Continuity of Instruction*

Michael Verdicchio
Department of Cyber and Computer Sciences
The Citadel
Charleston, SC 29409
mv@citadel.edu

Abstract

Academic disruptions are an eventuality that must be anticipated and handled in an efficacious manner. While many schools have general plans, especially for hurricane and other weather-related disruptions, the COVID-19 pandemic has elevated the issue beyond regional risk mitigation. Team-based software engineering and capstone courses in computer science are especially vulnerable to these types of disruptions, and this in turn risks the attainment of key student and program outcomes near the culmination of the academic program. By practicing some of the processes and tools they preach, however, software engineering courses have a natural advantage and ability to adapt rapidly to changes. This paper recommends specific classroom adaptations for professional software engineering practices and tools, provides a sample rubric and assessment instruments, recommends instruction approaches that ensure continuity of instruction, and concludes with a brief experience report for recent hurricane and COVID-19 disruptions.

1 Introduction

Ensuring continuity of instruction during academic disruptions is paramount in higher education. While hurricanes are a frequent cause of academic disruptions in the American Southeast and along the Eastern seaboard, the COVID-19 global pandemic has brought the issue into sharp focus worldwide. Beyond institution-wide disruptions, students recovering from illness are often out of the classroom for more than one week, which can greatly hinder their ability to participate and keep up. In these situations, within computer science education, courses dependent upon teamwork (e.g. software engineering and capstone-style courses) create unique challenges and urgency. By implementing some best practices, leveraging institutional resources, and adapting software engineering tools and techniques to the classroom, instructors of these courses can ensure the continuity of instruction, maintain attainment of course and program outcomes, and most importantly, provide all students the best chances to learn and succeed.

2 Background

Since the Agile Manifesto[5], and well before the ACM/IEEE 2013 curricular recommendations for computer science programs[9], incorporation of agile software engineering practices into computer science coursework has been incrementally moving forward. Fox and Patterson argued, in 2013, how agile should be more than an additional chapter in textbooks and that it should be discussed alongside plan-and-document processes[6]. In recent years, many courses and books have shifted to embrace agile methods and their use in the classroom.

While this paper is not debating the pedagogical merits of agile software development in principle, it does make the case that courses featuring agile software development projects are well equipped to gracefully handle major academic disruptions, including mandatory evacuations for hurricanes and pandemics, and even dramatically shifted instructional modalities, if we indeed practice what we preach.

3 Courses

The recommendations in this paper are based on our program's two-course capstone sequence in software engineering. The first course includes material on software processes (both agile and plan-and-document) and culminates in a large team-based software engineering project ideally dictated by an external client. The second course is a practicum and is based on a full-semester

team software engineering project intended to be delivered to an external client. While plan-and-document software processes are good choices for some projects, due to the spread of agile practices and their applicability to small- and medium-scale projects, both courses encourage students to adopt an agile process incorporating elements of Scrum and Extreme Programming (XP).

Student teams are typically instructor-selected based on strengths and weaknesses known about individual students, combined with survey and preference information collected from students at the start of the term. Ergin describes instructor-formed team selection strategies in his 2019 paper[4]. Our teams typically contain 4-7 students depending on the size of the class.

4 Agile Processes

While it comes in many varieties, agile software development focuses on delivering working software incrementally while responding to change gracefully. Teams typically self-organize and work in iterations, or sprints, each lasting 1-4 weeks. Scrum is a popular agile process that organizes through particular roles, ceremonies, and artifacts[2]. The primary roles of Scrum Master and Product Owner, and their adaptation to the classroom, are detailed in the next section. Scrum ceremonies include sprint planning, review, and retrospective, as well as a daily "stand-up" meeting. Artifacts of this process include the product backlog (the overall to-do list for the project), the sprint backlog (a subset for the current iteration), and various tracking and burndown charts. Extreme Programming (XP) takes some of these agile ideas to the extreme, emphasizing short iterations, test-driven development, and pair programming[7]. Some of these practices are quite efficacious in the classroom[8]. The next sections describe how these agile practices, roles, and tools have been adapted to the classroom.

5 Adapted Scrum roles

Teams applying a Scrum (or Scrum-like) software development process typically have two key roles: the Scrum Master, who represents management to the project, and the Product Owner, who defines the features of the project. While it is useful to study (and even practice) Scrum and other software processes, it is also helpful to acknowledge that students in our courses work under a completely different reality than a trained, full-time development team. As such, a slight adaptation of these roles to serve their academic needs can enable the student teams to be more productive and successful. We encourage our teams to rotate the roles with each sprint/iteration so that each team member has the opportunity to learn the responsibilities. We further encourage either the

Scrum Master or the Product Owner to rotate into the other role as one new person steps in. This provides a continuity of leadership as one rotates on and one rotates off. Since few students like to be the teammate always dictating to others and "cracking the whip," letting each teammate play that role for a short time both avoids team conflict and ensures that the responsibilities of the roles are actually carried out regularly.

Table 1 compares typical professional roles[3] of the Scrum Master with adapted versions of these roles for the academic setting. The academic roles are flexible whether or not the course staff is functioning as the project's customer and if not, whether or not a team must also check in with course staff. In a hypothetical sprint, the Scrum Master may join the Product Owner at a customer meeting if needed. The Scrum Master will make sure the team meets for regular stand-up meetings and makes sure the Scrum ceremony formats are followed. As development proceeds, it is the job of the Scrum Master to make sure that developers are following best practices, including proper code organization and software architecture, proper branching and version control, and proper progress monitoring of user stories. Finally, the Scrum Master would be the one (perhaps with the Product Owner) to represent the team to the course staff for progress reports.

Table 1: Adapted Scrum Master Role

Professional	Academic
Represents management to the project	Bridge between course staff and the project team
Responsible for enacting Scrum values and practices	Responsible for enacting Scrum practices
Removes impediments	Helps team to work around other responsibilities
Ensure that the team is fully functional and productive	Enforce team rules, norms, and standards
Enable close cooperation across all roles and functions	Ensure teammates are working and communicating
Shield the team from external interference	Encourages team focus and advocates for resources

Table 2 compares typical professional roles[3] of the Product Owner with adapted versions of these roles for the academic setting. In a hypothetical iteration, the Product Owner would re-prioritize the product backlog after speaking with the customer for a check-in and demo (perhaps along with the Scrum Master). At the start of the sprint, the Product Owner will facilitate the story/task estimation process with the rest of the team, helping to clarify questions and represent the voice of the customer. As team members mark stories as complete, the Product Owner would verify that the described functionality was implemented, tested, and reviewed properly before marking the story as complete and merging the code into the mainline branch. When necessary, the Product Owner will add new stories to the backlog and re-prioritize.

Table 2: Adapted Product Owner Role

Professional	Academic
Define the features of the product	Takes responsibility for the product backlog
Decide on release date and content	Leads the maintenance of the sprint backlog
Be responsible for the profitability of the product (ROI)	Represents the voice of the customer in meetings
Prioritize features according to market value	Primary contact for meeting with the customer
Adjust features and priority every iteration	Adjust features and priority every iteration
Accept or reject work results	Accept or reject work results (enforce code review)

6 Adapted Agile Ceremonies

In the sprint/iteration planning meeting, the team selects user stories from the product backlog for completion in the upcoming sprint. These stories are deconstructed into tasks and estimated. For student teams with no experience, these estimates are quite coarse, but there is benefit to repeating this process even for just a few weeks of a term. In these meetings, team members also give their commitments to their work, providing an accountability mechanism for the Scrum Master and Product Owner for the week. The student version of this real-world ceremony is similar, but smaller in scope.

The daily stand-up meeting, or "scrum," is a key part of a full-time agile software engineer's workflow. The agenda for this meeting consists of three simple questions[2] for each team member: 1) What did you do yesterday? 2) What will you do today? 3) Is anything in your way? While daily is ideal for this ceremony, students have classes two or three days per week and may not see their teammates at other times, especially if they live off-campus or go to school part time. While student teams will need to meet regularly outside of the classroom, we have found that providing a short amount of time at the end of each class period gives teams a chance to hold at least two or three stand-up meetings per week, thereby facilitating momentum, accountability, and hopefully, further team communication.

The sprint review is where a team typically presents and demonstrates the accomplishments of the sprint [2]. The student version of this ceremony is likely less formal, but it provides an important accountability check for the team and reinforces the principle of always refining working software. Regularly demonstrating new features to an external client and/or course staff keeps student teams on the right schedule and increases their likelihood of delivering a good final product without requiring last-minute heroics. In the next section we discuss some tools that can enable this ceremony, especially for students.

The sprint retrospective provides a critical improvement opportunity for professionals and students alike, as teams reflect on what worked and what did not work during the last sprint. While it is good to give student teams autonomy, if possible, we have seen benefit in the earlier sprints from the course

staff facilitating the first of these short meetings. Asking team members in a group setting what we should start, stop, and/or continue doing encourages them to participate candidly in this process of continuous improvement.

7 Tools Supporting Agile Processes

Tool support for agile processes is not new, but helping one's students choose the right tools can enable learning and continuity of instruction in the event of an academic disruption.

7.1 Tracking User Stories

Perhaps the key artifact of Scrum and other agile processes is the product backlog. Using a physical tracking system such as Kanban[1] is a great way to visualize and collaborate, but it is difficult to implement for teams without a dedicated space for each team. Virtual systems provide a good alternative, accommodating full- and part-time student team members, and those who do not reside on campus. Some currently popular tools include Jira (jira.com), Pivotal Tracker (pivotaltracker.com), Trello (trello.com), and the Issues feature of GitHub (github.com). Some tools integrate the ability to note user story points and facilitate other estimation practices, and most offer free access for educational use. We have had success with Tracker from Pivotal. Most of these tools can also automatically generate tracking and burndown charts that are useful for measuring student progress throughout the project.

7.2 Version Control

Version control is critical for any software project's success, and as an industry standard practice, promoting its use by students helps equip them for the next steps in their careers. Git is arguably the most popular technology currently in use, and its decentralization and branching makes it an ideal fit for team projects. GitHub.com arguably provides the best tool support around Git, and builds in many features useful for professionals and students alike.

We encourage the "shared repository model" for team collaboration in GitHub, whereby all team members add new features to a project in a development branch, and then open a pull request to merge that branch back into the mainline branch. This pull request and its subsequent approval or rejection by the product owner further reinforce accountability and the code review process. Conveniently it also leaves a clear trail of activity for inspection by course staff. Over time, the branch and commit history of a team's repository is the plain and true evidence of participation from each team member.

7.3 Continuous Integration and Continuous Deployment

As they are able and as time permits, we encourage students to use web services to test, deploy, and monitor their applications in production. Tools like Heroku (Heroku.com) not only provide a production environment in addition to their development environment, but provide a convenient way to demonstrate the current version of the software to customers and of course staff, online. Other tools, such as TravisCI (travis-ci.org), Jenkins (jenkins.io), and others, provide a way for students to automate tests and measure code quality. Many of these tools cross-integrate and enable an academically disrupted or geographically distant team to feel like their project is still in motion despite circumstantial changes.

7.4 Team Communication

We require the use of Slack (Slack.com) for team communication as it (and similar technologies) are widely used in the industry. Slack in particular offers many integrations that can tie in useful process information from our other tools. Each team member is added to a private team channel with the course staff, and they are encouraged to download desktop and mobile applications and enable notifications. Students often want to default to group text messages or other options like Discord, but getting student buy-in for one system provides eventual communication benefits to them, and oversight abilities for the course staff.

We instruct students to link their Slack channel with their GitHub repository, which will automatically post whenever there is a commit, pull request, or other significant activity in the repository. This serves as a reminder that one's teammates are working (and maybe they should be as well), and that there may be important changes to the mainline code base that they should pull into their development branches. After a few iterations, we also encourage teams to connect their Slack channel to their production environment (e.g. Heroku for web applications), to their Continuous Integration tools (e.g. Jenkins or TravisCI). All of these recommendations make Slack the home base of communication for everything regarding the project. If Slack is being regularly used when students are together, in the event of a disruption, there can be a nearly seamless transition to a distributed team modality.

8 Student Assessment

Like any course, early and expedient instructor feedback to students is helpful and allows corrections and improvements not possible when only the final project output is assessed. The iterative nature of agile software development

lends itself well to instructor feedback by assessing student contributions to each sprint. For the first course in software engineering we recommend one-week sprints to allow this feedback and to avoid inevitable student procrastination. For second or capstone courses, teams should consider one or two weeks for a sprint length since they have prior experience. We make team expectations clear for each sprint in terms of the number of expected user stories to be delivered, test coverage to be achieved, etc. We also hold individuals accountable with a simple survey completed by each person. The survey questions are listed below, and allow the instructor to quickly intervene if red flags in participation or attitude are discovered.

- Select role: Scrum Master, Product Owner, or Team Member
- List all team meetings you attended
- List all team meetings you missed
- Describe the top two achievements by your team this sprint
- Describe your biggest personal contribution this sprint
- Who on your team deserves recognition for this this sprint?
- Who on your team was not doing their job this sprint?
- What letter grade does your team deserve for this sprint?
- What letter grade do you deserve for this sprint?

The answers to these questions can be quite telling. Some students are very self-aware and know they need to do better next time. Other students are a bit delusional and claim to be deserving of high grades when there is little or no evidence of activity. Weak team members and strong leaders become evident quickly. All of this information can be used to inform the strategic interventions of the course staff with the teams.

Additionally, the course staff at our institution schedules a weekly meeting with the Scrum Master and Product Owner of each team. This is a purely instructor-role meeting, so if the instructor is also serving as the team's customer, those discussions happen separately. This meeting is useful in obtaining a candid, leader's view of the team's progress and is a way for the course staff to indirectly influence the team and its members towards corrective actions. Rotating the Scrum Master out, the Product Owner into the Scrum Master role, and a new team member into the Product Owner role keeps a longer continuity of leadership and a better opportunity for instructor influence.

Assessing each sprint or iteration involves examining the agile project artifacts: version control history, product backlog, test coverage/code quality, etc., combined with the sprint evaluations from team members. For version control and user story tracking systems, we provide students with the following letter grade rubric shown in Table 3 to make clear what kind of activity is expected.

Table 3: Letter Grade Rubric for Version Control and User Story Tracking

Grade	Description
A	Organized with regular updates from all team members
B	Mostly organized with updates from all team members
C	Mostly organized with updates from some team members
D	Disorganized with sporadic updates
F	Disorganized with little to no activity, or missing

9 Experience Reports

In recent years, our institution was closed for at least one week due to hurricane evacuation orders in 2016 (Matthew), 2017 (Irma), 2018 (Florence), and 2019 (Dorian), each happening in early to mid-September, except Matthew, which was early October. In 2016, team projects had just begun and were highly disrupted since most of the recommendations in this paper were not yet in place. Storms in 2017, 2018, and 2019 came very early in the semester and had a smaller impact on projects since they'd not fully started at the time of evacuation. If we experience another hurricane evacuation in 2020 or beyond, we will have the recommendations in this paper in place to ensure continuity of instruction.

The spring semester of 2020 confirmed the efficacy of these recommendations when our institution broke for Spring Break and never returned for face-to-face instruction due to the global pandemic of COVID-19. Because students were already communicating via Slack, the team stayed engaged. The instructor moved weekly meetings with the Scrum Master and Product Owner to video conferencing and collected individual sprint reports online instead of in hard copy. As a result, a monumental academic disruption basically amounted to a switch to a distributed team – an incredibly real-world experience our students will be able to describe in job interviews. Some students had difficulty keeping up and participating with such an abrupt shift in academic circumstances, but red flags were easy to spot and the course staff intervened and accommodated as much as possible. In the end, despite all the challenges, projects were completed successfully. While not as satisfying as the in-person experience, teams presented final projects virtually and demonstrated working projects from their web-based production environments.

10 Conclusion

In this paper we have reviewed some agile software development processes, practices, and roles, and described their adaptations to the academic envi-

ronment. We have made recommendations for course and team organization, tools, and processes to ensure the ability of our software engineering courses to thrive face-to-face, online, or in a situation where they are forced to jump from one modality to another. Finally, we have described a rubric and assessment instruments and strategies for providing early and helpful feedback to team members and leaders to provide the best environment for students to succeed. If any type of course should be able to adapt quickly to disruptions and ensure continuity of instruction, it should be agile software engineering courses.

References

[1] E. Brechner. *Agile project management with Kanban.* Pearson Education, 2015.

[2] M. Cohn. Scrum. `mountaingoatsoftware.com/agile/scrum`.

[3] M. Cohn. *Succeeding with agile: software development using Scrum.* Pearson Education, 2010.

[4] Huseyin Ergin. Instructor-formed capstone teams based on interest and technical experience: the road to success. *Journal of Computing Sciences in Colleges*, 35(5):37–49, 2019.

[5] Martin Fowler, Jim Highsmith, et al. The agile manifesto. *Software Development*, 9(8):28–35, 2001.

[6] A. Fox and D. Patterson. Is the new software engineering curriculum agile? *IEEE Software*, 30(5):88–88, 2013.

[7] A. Fox, D. Patterson, and S. Joseph. *Engineering software as a service: an agile approach using cloud computing.* Strawberry Canyon LLC, 2013.

[8] An Ju, Adnan Hemani, Yannis Dimitriadis, and Armando Fox. What agile processes should we use in software engineering course projects? In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE ' 20, page 643–649, New York, NY, USA, 2020. Association for Computing Machinery.

[9] ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013: curriculum guidelines for undergraduate degree programs in computer science, 2013. `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`.

History of Technology and Discovery: A Study Away Experience in Computer Science*

Kevin Treu
Department of Computer Science
Furman University
Greenville, SC 29613
kevin.treu@furman.edu

Abstract

This paper describes a study away course in Computer Science on the history of technology and discovery, implemented in various locations in the United Kingdom and Ireland. The author intends for the course to be a possible template for replication as a short-term travel course, or as a component of a comprehensive semester-long program.

1 Introduction

Study away programs (alternatively referred to as "travel study" or "study abroad") have long been considered to be the near-exclusive domain of disciplines in the arts, humanities, and social sciences. Though there have been successful examples reported in the natural sciences [10, 13] and even some in computer science [2, 3, 4, 5, 12], proportionately few such experiences have been designed and implemented. Yet the benefits of studying computer science (CS) in the context of a travel experience are certainly no less than in other disciplines. Indeed, the history of CS over the last two decades suggests that broadening the worldview of our students is a particularly beneficial objective.

Though the academic discipline of CS is currently enjoying what some have called the "third surge" of enrollment spikes [14], the times between those

surges have prompted considerable reflection. It has been argued, for example, that CS has not always adequately emphasized its grand challenges –often focusing more on narrow applications than on enhancing the capabilities of humans and human society through machines and algorithms [11]. Others have demonstrated that the appeal of studying CS is closely tied to the marvelous diversity and versatility of the field [8], leading to significant emphasis on versatility, flexibility, and interconnectedness in the ACM/IEEE Computer Science curriculum guidelines –for example: "Computer science curricula should be designed to prepare graduates for a variety of professions, attracting the full range of talent to the field." [1]

"Are we about a machine or an idea?" [11] This paper describes the design and implementation of a study away course entitled *CSC-273: History of Technology and Discovery*, conducted in various locations in the British Isles, and argues for the effectiveness of the lesson that technology is about *ideas* more than *machines* when the lesson is taught on-site of some of history' s greatest technological discoveries. Following is a description of the course and the logistics of the travel element, along with a discussion of lessons learned. It is hoped that this experience –or certain elements of it –might be useful to others interested in designing a study away experience in computer science for a students from a broad spectrum of academic backgrounds and interests.

2 Course Components

History of Technology and Discovery was offered from August to December in the United Kingdom and Ireland in 2018, but was in the planning stages for three years prior to that. The semester-long program entitled "Study Away in the British Isles" is Furman University' s oldest established study away program. It partners a faculty member from the Department of English with a colleague from a different discipline, with the curriculum adjusted accordingly. Frequent collaborators have included the departments of Theatre Arts, Political Science, History, and Art. The partnership between English and Computer Science was implemented for the first time in 2018, with a recognition among both faculty directors that the combination of disciplines was both appropriate and advantageous, due to the interdisciplinary natures of both English and CS, and to the rich histories of both that are specifically connected to the British Isles.

The English/CS version of the program consisted of four courses in total:

- *Drama in Stratford and London*
- *Literature of Early Modern Britain*
- *History of Technology and Discovery*
- *Travel Study in the British Isles*

19 students were accepted for the program after an application process that began in the fall of 2017. All of the students were required to attend multiple orientation sessions in the spring of 2018, and were well versed on rules of deportment, details of the technology that would be employed, and course expectations. Structurally, the modules and assessment techniques for all of the courses were consistent with each other by design. The details of the *History of Technology and Discovery* course components are largely representative of those comprising the other courses, which had successfully been offered for many years.

It is worth noting that this course has been offered several times on campus. The components were significantly modified to take advantage of study away.

2.1 Summer Requirements

Prior to the departure of the group in August, the students were required to do a considerable amount of reading and writing during the summer break. This included several Shakespeare plays, and readings from British literature for the other classes in the program. For CSC-273, the workload was light. With a trip to Bletchley Park planned as a highlight of the trip, a full module was designed around Alan Turing. Thus, the students were assigned the biography *Alan Turing: Computing Genius and Wartime Codebreake* [6]. Students were given several discussion prompts and were each required to submit three one-page papers in response to these prompts. These were accompanied by three original discussion questions each. Representative prompts included:

- *Chapter 4 attributes two specific contributions that Turing made to code-breaking at Bletchley Park – the use of cribs (as dramatized in one of the best scenes in The Imitation Game), and analysis of probabilities. Describe to the best of your ability the workings of the Enigma machine and these two prominent techniques that Turing used to break the code.*
- *The previous reading discussed Turing's theoretical contributions to computers at length. Summarize the specific contributions that he made to the actual construction of real computers.*
- *One objection to the notion of machine intelligence in general is that "a machine is not conscious". Do you agree or disagree that this is a fundamental impediment to machine intelligence? Is it the only impediment? What was Turing's response to this objection?*
- *The final section of the book is entitled "Turing's Shrouded Legacy." What about his legacy is "shrouded," do you think? Give three reasons why Henderson (and other historians) might describe it this way.*

This preparatory work was designed to prime the group for subsequent conversation during the trip, when the Turing module was scheduled.

2.2 Readings and Site Visits

The heart of the course was found in the visits to various sites during our travels. Each site visit was preceded by a reading or viewing assignment. Articles, videos, and/or book chapters were provided for each, with students expected to prepare in advance for the visits. In addition to the Turing biography, required texts included *Science and Technology in World History*, by McClellan and Dorn [9] and *The Innovators*, by Isaacson [7]. Otherwise, articles and videos were generally provided as web links.

A comprehensive list of site visits follows, along with the theme or lesson associated with each. Reading and viewing assignments are omitted for space considerations, but are available upon request.

- Dublin — *Book of Kells* — ***Preservation, Conservation, and Digitization of Ancient Texts***
- Dublin — *Guiness Storehouse* — ***Technology of Brewing***
- Belfast — *Titanic Belfast* — ***Failures of Technology***
- Edinburgh — *Camera Obscura* — ***Tech of Cameras, Light, and Optical Illusions***
- Leicester — *King Richard III Visitor Centre* — ***Tech of Archaeology and Identification***
- Milton Keynes — *National Museum of Computing* — ***History of Computers***
- Milton Keynes — *Bletchley Park Trust* — ***Alan Turing and Codebreaking***
- Cambridge — *Cavendish Museum* — ***DNA: Discovery, Fingerprinting, and the Human Genome Project***
- Oxford — *Museum of the History of Science* — ***Women of Science***
- Salisbury — *Stonehenge Monument* — ***Ancient Computing and Math***
- London — *London Transport Museum* — ***(1) Tech and Big Data in Transportation; (2) Industrial Revolution***
- London — *Victoria Albert Museum* — ***Future Technology and Art***
- London — *Science Museum* — ***Darwin and the Origin of the Species***
- Greenwich — *Royal Observatory* — ***Invention of Mobile Timepieces***

2.3 Travel Journals

Simply participating in the site visits listed above was an enriching experience. However, to optimize the educational value some focused reflection was required. All students were to maintain an online travel journal recording their experiences at historic sites and museums as they relate to the history of technology. Each student was required to find at least one experience to record from ten of the technology-related historic sites or museums that we visited

during our tour. For example, when we visited Bletchley Park, multiple students posted an entry describing the strategy employed by Alan Turing to defeat the Enigma, while others wrote about the role played by women in the codebreaking effort, or the inner working of the Enigma itself. It was expected that journal entries would be heavily documented with photography or other graphics.

2.4 Papers

As noted previously, short one-page papers were required in response to summer reading assignments. During the actual semester, three papers of 1000-1500 words were assigned with staggered due dates, including one in early December after our return home. Topics were designed to be related to specific site visits, or to aggregate thematic ideas from multiple visits. The topics were:

- *When Good Technology Goes Bad (And We Allow It To)*
  This topic was assigned after our visit to the Titanic Museum in Belfast, but students were asked to reflect on the possible dangers of misuse or overreliance on technology from any of our visits, or from their own research. Topics included analysis of the dangers of gene editing with CRISPR, reliance on artificial intelligence (inspired by the Turing Test), and the space shuttle Challenger disaster.
- *The Historical Significance of Alan Turing*
  As mentioned earlier, the life and work of Alan Turing –along with a two-day visit to Bletchley Park where he helped break the German Enigma code –was a centerpiece of the class. Until very recently, however, he was an obscure historical figure at best. With the 2015 release of the film *The Imitation Game*, however, he has had something of a renaissance. Is it deserved? Students were asked to select one of two topics: the breaking of the Enigma code, or the invention of the computer. For their selection, they had to build a case either for or against Turing as a significant historical figure.
- *Prominent Women in the History of Technology*
  Throughout the trip we had the opportunity to study the often overlooked contributions of women to the history of technology. From an exhibit devoted to women in science at the Oxford Science Museum, to a lecture on Ada Lovelace that was fortuitously scheduled during the week we spent in Oxford, to a chance visit to the Herschel Museum of Astronomy in Bath (devoted to William Herschel but highlighting the work of his sister Caroline as well), there were ample opportunities to celebrate the significant roles that women have played. Student were given the opportunity to select one for their paper topic.

2.5 Lectures

Though there was significantly less emphasis placed on lectures than during a typical class, there was a lecture component. During the first half of the trip, as we frequently moved from location to location, we scheduled several lectures during evening hours in hotel meeting rooms. A notable lecture on the history of computing took place in the conference room of our hotel in Milton Keynes. Upon our arrival in London for the second part of our trip on October 1, we had a room reserved at the nearby University of London, Birkbeck campus, where lectures and discussions on various topics were held four days a week.

2.6 Student Assessment

Student assessment was very straightforward. Journal entries counted for 3% of the grade each, for 30% total. The three papers described above each counted for 20% of the grade, for a total of 60%. The last 10% of the grade was earned through deportment. This included the one-page summer papers, but also critically important elements such as class discussion participation, punctuality, and attentive behavior at site visits.

3 Conclusion

By most measures, the class (and the entire program) was a success. Quantitatively, 92.9% of the students *Strongly Agreed* that the experience of this class was an "enriching educational experience". 100% *Strongly Agreed* or *Agreed* that it "stimulated their desire to learn". Responses for other student evaluation questions were comparable.

The quantitative measures tell only part of the story, however. There are a number of lessons and conclusions to be drawn from the experience.

The history of technology did indeed fit beautifully together with the other topics covered in the program. It was particularly gratifying, for example, to observe the number of times references were made to Ada Lovelace outside of the expected context –in reference to her father Lord Byron, and also to her friendship with Charles Dickens. Additionally, we were all repeatedly reminded about how the history of technology impacts –and is impacted by –theatre and literature. Technology involved in the preservation and sharing of the Book of Kells is one obvious example. And quite frequently we talked about the effects that technology had on various theatrical productions that we saw.

Demonstrating to students the relevance that computing technology has to a variety of disciplines is a significant objective of the CS curriculum, and this course was a qualified success in that area. Students certainly benefited from integrating their domain knowledge with computational thinking skills, specific

techniques and tools, and a broad understanding of the role technology plays in society that I believe they are likely to find relevant.

However, this benefit was hard-earned. As noted previously, the history of study away at Furman –and of this particular program –is dominated by humanities, fine arts, and social sciences. It was a constant struggle to overcome student bias in the form of "Why do we have to take this course?" There was a pre-conceived notion of what CS entails (e.g., difficult, uninteresting programming) that was dispiriting at times. But anecdotal feedback provided since the end of the trip gives strong support for the idea that the interconnection of the material with the other courses was ultimately understood and appreciated.

It is a hypothesis of this paper that the experience of this study away class is potentially replicable by others. Though it may be unlikely for a semester-long program of this exact composition to be re-created elsewhere, perhaps the *History of Technology and Discovery* course could be an effective part of an immersive curriculum in partnership with Mathematics, Physics, or Biology – all topics that were touched upon in the class.

More likely is the possibility of teaching this course as a standalone experience in a short semester, such as a January or May term, or in the summer. These types of study away experiences have gradually become predominant at Furman, in fact. Should such a course be planned, one of the most encouraging lessons of this experience is that the travel itinerary doesn' t necessarily have to follow a strict list of sites dictated by a pre-determined list of topics. Several of the site visits listed above were added *after* the itinerary was set. The history of technology is such a rich field that it seems just about any major city has ample locations of educational value. This was demonstrated on our trip by the number of fortuitous opportunities the students found to explore the topic. The National Museum of Scotland, for example, has an entire science wing that several students explored on their own and reported on. Also in Edinburgh, a planned visit to the Royal Yacht Britannia quickly became an opportunity to learn about technology at sea. While in Bath, England to visit the Roman Baths, we came across the Herschel Astronomy Museum, celebrating the pioneering work of William Herschel and his sister Caroline. (Later, we saw a telescope invented by the pair at the Royal Observatory in Greenwich.)

Of course there were many lessons learned about the planning and logistics of such a trip, but those are outside the scope of this paper and available upon request.

References

[1] *Joint Task Force on Computing Curricula Association for Computing Machinery IEEE Computer Society, Curricula 2013: curriculum guidelines for undergraduate degree programs in computer science, 2013*, 2013.

[2] K. Abernethy, P. Gabbert, and H. Reichgelt. Information technology training in developing countries: A case study in jamaica. In Deryn Watson and Jane Andersen, editors, *Networking the Learner: Computers in Education*, pages 787–794. Kluwer Academic Publishers, Boston, MA, 2002.

[3] J. Butler and S. Khoja. Development and implementation of an information-based short-term study abroad course. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, page 1340, New York, NY, USA, 2020. ACM.

[4] P. Gabbert and C. Patterson. Interdisciplinary sponsored foreign studies in computing and business: Furman university' s winter term in jamaica. In *Proceedings of the 2002 International Applied Business Research Conference*, Puerto Vallarta, Mexico, 2002.

[5] M. Goldweber. Study abroad experiences in computer science (abstract from bof session). In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education. Association for Computing Machinery*, page 699, New York, NY, USA, 2015. ACM.

[6] H. Henderson. *Alan Turing: Computing Genius and Wartime Codebreaker*. Chelsea House Publishers, 2011.

[7] W. Isaacson. *The Innovators: How a Group of Hackers, Geniuses, and Geeks Created the Digital Revolution*. Simon Schuster, 2014.

[8] J. Margolis and A. Fisher. *Unlocking the Clubhouse*. The MIT Press, Cambridge, MA, 2001.

[9] J. McClellan and H. Dorn. *Science and Technology in World History*. Johns Hopkins University Press, 2015.

[10] J. McLaughlin and D.K. Johnson. Assessing the field course experiential learning model: Transforming collegiate short-term study abroad experiences into rich learning environments. *Frontiers: The Interdisciplinary Journal of Study Abroad*, 13:65–85, November 2006.

[11] J. Morris and P. Lee. The incredibly shrinking pipeline is not just for women anymore. *Computing Research News*, 16(3), May 2004.

[12] P. Pollock, J. Atlas, T. Bell, and T. Henderson. A computer science study abroad with service learning: Design and reflections. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 485–490, New York, NY, USA, 2018. ACM.

[13] J. Smieja, G. D' Ambruoso, and R. Richman. Art and chemistry: Designing a study-abroad course. *Journal of Chemical Education*, 87(10):1085–1088, 2010.

[14] T. Soper. *The exploding demand for computer science education, and why America needs to keep up*, June 2013.

Delivery of an Innovative Computer Programming Curriculum to Impoverished Middle School Learners in South Africa*

Robert Allen, William Darragh, Harrison Verhine
Computer Science Department
Mercer University
Macon, GA 31207
allen_r@mercer.edu,
wdarragh16@gmail.com,
harrison.m.verhine@live.mercer.edu

Abstract

In the summer of 2019, a group of Mercer University students and professors joined with South African educators for a unique Computer Science intervention. The cooperative efforts of these groups lead to the development of a curriculum particularly suited to teaching middle-school-aged children in impoverished areas of South Africa. The curriculum was designed to give these children the tools to be life-long learners of Computer Science (CS). Two professors and thirteen undergraduates, including computer science, engineering and humanities majors, traveled to South Africa to join with their South African counterparts to teach children over the course of three weeks. We present the key components of our innovative curriculum, reflections on the overall project, and ideas that may be of interest to the body of Computer Science educators at large. In particular, we believe that teaching children general problem-solving skills, and teaching them how to use tools for learning Computer Science, is greatly empowering. This paper explains our initial project development and implementation, and it outlines plans for similar future work.

1 Purpose with Clarifications

1.1 Commitment to Computer Science Education

The core agents in developing this curriculum and intervention are not formal researchers in the field of Computer Science Education (CSE). This project was built with a service-first perspective. Formal methods of evaluation were not implemented, and this paper stands as qualitative reflection of a CSE intervention, rather than a quantitative report of a CSE experiment. The intervention described in this paper was inspired by anecdotal experiences, and the qualitative findings may be of interest to the larger body of CS educators.

1.2 Concerns on Human Aspect

All references to the South African children have been made anonymous, and in no way are we attempting to present them as a part of a human study. Our community partners in South Africa (SA), the undergraduate student participants, and professors have all given their permission to have their names and pictures shared in any report that presents the findings of this project.

1.3 Terminology

We describe two groups of people who may be referred to as students, namely the South African middle school students, and the Mercer undergraduate students. We adopted the terminology commonly used in SA to distinguish between college-level students and children going through primary and secondary schools. The South African children are referred to as *learners* and the undergraduate students as simply *students*.

2 Intervention Origins

During the summer of 2018, undergraduate CS student William Darragh was a participant in an international service learning project working with small businesses around Cape Town, SA. While there, he met Filbert Mushiringi, an educator who works through his non-governmental organization (NGO) to empower children in the impoverished urban townships around Cape Town by teaching CS. Mushiringi feels that technical skills are particularly important, given that these learners are not exposed to any CS through their normal curriculum. Darragh and Mushiringi began correspondence, sharing ideas about best practices for teaching coding to children. Through their conversation, Mushiringi expressed that he had been struggling to teach children in such a way that they could move from completing guided tutorials to inventing

their own projects. He has also experienced working in schools with sufficient computer facilities, but without experienced CS educators. Seeing an opportunity to take advantage of Mercer' s unique resources, Darragh joined with Dr. Allen to design a project where undergraduate students would teach coding to middle-school learners in SA. Mercer has a rich history of service through their Mercer on Mission (MOM) program. This program funded the intervention that Darragh and Allen designed with Mushiringi. Figure 1 depicts the students with the learners in Durban.

3 Curriculum Development

3.1 Intervention Constraints

3.1.1 Timing

There were many complex components to consider when outlining a plan for this intervention. MOM funds three weeks of service in country. Based on recommendations from our SA community partners, we decided to work for one week in Durban and two weeks in Cape Town. We designed a plan to deliver one, week-long curriculum to three different sets of students, one in Durban, and two in Cape Town. So, one constraint was that this would only be a week-long course of instruction. Conveniently, we were able to schedule our three weeks to correspond with South Africa's three-week Winter break. Our community partners were able to arrange access to the respective schools' computer labs for five days each week, and four hours each day.

3.1.2 Hardware, Software, and Internet

Working in multiple classroom environments means working with different sets of hardware, software, and internet access. The tools used during this intervention needed to be flexible enough to function on these different sets of limitations. Through discussions with our community partners prior to the intervention, we found that we could rely on having internet access but had little information on specifications of the computers being used. Based on this, we decided to use cloud-based tools. This gave us the advantage of not being reliant on the hardware, as the processing of programs happens remotely. Additionally, cloud-based tools only require a browser, with no installations. In Cape Town, the school already had internet infrastructure prior to the intervention, while in Durban, we had to set up a cellular hot-spot WiFi router. We wanted the target learners to have the ability to use these tools after the intervention. This was not an issue for the learners in Cape Town; however, in Durban we left behind the wireless router to use for any future CSE endeavours.

3.2 Our Unique Pedagogical Approach

Our curriculum was designed to supplement existing Open Educational Resources (OERs) for CS by teaching students how to seek out additional information themselves. There are already many OERs available through a web browser online. Common examples include Code.org [4] and CodeCombat [3]. Mushiringi expressed a concern that these tools had little in terms of *exit strategy*. He described how once students finished tutorials on these sites, they were unsure of how to continue their learning. We sought to address that need. We did not set out to create an entirely new set of tools to teach coding. Instead, we combined existing OERs into an innovated curriculum that would turn already effective tools into starting points for life-long learning. Our idea for this extends from the same spirit of *learning to read* to *read to learn*. We wanted to show leaners that tools like Google-searching, reading documentation, and interacting with community pages is an effective way to problem solve and continue learning. Thus, we came up with the idea to split lessons into two parts: intro and explore. The intro section would use existing OER material and get students introduced to the tools. After spending a day with an introductory lesson, we transitioned into an explore lesson. Explore lessons focused on how to use the internet to find answers to questions, how to use documentation, and how to generally troubleshoot. Our goal was not to teach them everything about any one particular tool, but to show them how they could use resources readily available to learn as much as they desired.

3.3 A Three-Part Approach

With only one week with each set of learners, we wanted to maximize their chance of connecting with coding. Various approaches to problem solving will catch different students attention. We chose to present three approaches to solving problems, combined with an explore-on-your-own capstone component. Our curriculum includes problem solving in the physical world, with block-based coding, and with traditional text-based coding. This three-part approach was designed so that students would understand the basic fundamentals of programming, no matter the format, and to understand that coding is simply giving instructions.

3.4 Selection of Specific Languages and Tools

3.4.1 Problem Solving in the Physical World

We incorporated real-world models into our curriculum, and given the somewhat uncertainty on internet reliability, we prepared a collection of physical world lessons based upon *CS Unplugged* [1]. "CS Unplugged is a collection of

free teaching material that teaches Computer Science through engaging games and puzzles that use cards, string, crayons and lots of running around." Each Mercer student prepared to lead a CS Unplugged exercise as needed. We purposefully over-prepared with CS Unplugged in case we had days with poor internet connections. Our signature physical world exercise involved role-play with a *robot* standing on a sidewalk chalk grid. The other learners issued a series of commands written on note cards, which told the *robot* to do things like move forward or turn. The task was to have the robot move a ball from one cell to another. This activity was adapted from the CS Unplugged "Kidbots" activity [2]. Learners participating in this activity are shown in Figure 2.

3.4.2 Problem Solving with Graphical Code

An obvious candidate for graphical coding was the *Scratch* programming language [6]. Scratch is a free, web-based, visual programming language developed at MIT. Users program with drag and drop blocks that snap together. Scratch is well suited for making games and animations and is a great option for students first learning programming. The drag and drop blocks reduce syntax errors and the visual environment offers responsive feedback. The tutorials on the Scratch website provide great introductory lessons. We followed them up with exploration exercises, including some game development and animating green-screened photos of the learners, as shown in Figure 3. Verhine originally learned programming using Scratch and gave key insight into planning Scratch lessons, as well as serving as a leader in the classroom.

3.4.3 Problem Solving with Text-Based Code

A challenge with text-based programming is ensuring that students have access to a reliable code editor and proper compiler/interpreter. Recent advancements in online integrated development environments (IDE)s have mitigated this problem. We chose to work with a powerful tool called *Repl.it*. Repl.it is designed "to make programming more accessible. [They] build powerful yet simple tools and platforms for educators, learners, and developers."[5] Repl.it is an excellent education tool and programming environment for a host of reasons. First of all, the compiling and interpreting happens on remote servers, so it can be used effectively on any machine with a web browser and reasonable internet. Second, Repl.it has features that enforce fundamental ideas to coding, such as file management and publishing code. Third, Repl.it has the ability to create classrooms where it is easy to deliver assignments to the students.

Repl.it has the capacity to be used with a variety of languages. We chose to teach Python for its unique duality of being famously easy to pick up while still being used in real technical applications, and it will likely remain a relevant

Figure 1: Mercer Students with the learners in Durban, South Africa

Figure 2: Learners participating in CS Unplugged exercise

Figure 3: Learners animate green screen photos of themselves

language to CS education for years to come. We wanted to also play off the graphical nature of Scratch and our CS Unplugged lessons, so we chose to use the Turtle library to enhance the Python learning experience. The Turtle library allows the programmer to draw 2D shapes by using basic commands to move a cursor, or turtle, around the screen as shown in Figure 4.

3.5 Bringing It All Together

To add an element of cohesiveness to our curriculum, we focused on similar topics in all three approaches. All lessons focused around simple procedural programming, manipulating an object on a grid. In the case of CS Unplugged, it was a *robot* moving on a physical grid. For Scratch, it was manipulating sprites using simple block commands. And for Python, it was programming with the turtle module, using text commands to move a turtle around. Some common ideas in all three system were coordinates, angles, order of commands, and relative versus absolute positions. With the CS Unplugged activity, we drew a direct link between the physical world example and Python programming by using command note cards with plain English instructions on one side, and Python syntax equivalent commands on the other. See Figure 5.

4 Experience and Reflections

4.1 One Teacher, Many Helpers

Throughout this intervention, we employed what we think is a novel classroom setup. We used the large number of student teachers to our advantage. We had one student leader or professor in the front of the classroom teaching each new concept through demonstrations. This lecturer was the group's expert on the particular subject. The remaining students were divided among groups of learners. Although these students may not have been as knowledgeable as the lecturer about a particular subject, they were able to give special attention to each group of learners and provided instant technical support when necessary. These students also served as an intermediary between the learners and the teacher. Often, the learners felt more comfortable talking directly to the helper student teachers than the lecturer, so they were able to communicate to the lecturer that he or she needed so slow down or clarify a concept.

4.2 Local Support for Basic Concepts

We found that local support was integral to effectiveness in teaching. Students in Cape Town had been working with Mushiringi and other CS instructors prior to our intervention. This gave them a solid foundation for us to build on.

The students in Durban had a much weaker foundation. A local community member Brian Mtolo had been volunteer teaching some computer skills, but the Durban learners had significantly lower digital literacy than the learners in Cape Town. Many digital and mathematical concepts were foreign to them, especially when these concepts were explained in English, a second language to them. Mtolo was able to translate our explanations for especially tricky concepts, which had a significant impact on the learners' understandings.

4.3 Working with Non-technical Majors

Our humanity majors contributed greatly to the refinement of our program. During the team training sessions, prior to going to SA, our humanity majors provided valuable feedback on presentation, terminology, and flow of our lessons. Their non-technical perspective of learning this material taught all of our technical students a great lesson - don't preach to the choir. We adjusted the delivery and flow of our lessons accordingly and believe these changes improved the SA learner's understanding. Our humanity majors also brought a unique, added-value while teaching in SA. They related with the young SA learners on a very human level. They connected in a deeper, more personal way, and brought out many SA learner's willingness to be creative while coding our lessons. Figure 6 showcases some SA learners finding a way to put a pink hat on their Turtle-drawn character.

5 Intervention Summary and Final Thoughts

5.1 The Intervention in Summary

We have observed a Win, Win, Win situation from this intervention:

- The SA learners were excited about coding. They eagerly explored to create new projects. They demonstrated ability in continuing learning. Even months after the intervention, some learners were logging into our Repl.it classroom.
- The Mercer students experienced the joy of teaching. They witnessed growth and discovery by their learners, which empowered them, and many expressed the desire to continue work in this area. Tangentially, the Mercer students also broadened their world view by working in the SA impoverished township.
- The SA CS teachers benefited greatly as they, too, experienced our curriculum. There were many moments when we spent extra time teaching the teachers in SA a variety of tools and tricks in CS. Both teachers

Figure 4: Learner demonstrates a Python with Turtle Project

Plain English	Turn Left	Turn Right	One Step Forward
Python Syntax	turtle.left(90)	turtle.right(90)	turtle.forward(1)

Figure 5: Example note cards helped learners relate the CS Unplugged activity to Python code

Figure 6: Learners demonstrate an enhancement to their Python project where they have added a matching pink hat to the figure

expressed a gracious gratitude for our work and a sincere desire for our return.

5.2 Final Thoughts

There were numerous factors that lead to the success of the trip. Having local community partners participate in teaching helped build trust with the learners and supplemented our instruction in English with some instruction in the learners' native language. Despite only having access to computers with meager hardware resources, cloud-based OERs allowed for effective education. Encouraging students to search the internet for solutions to problems gave them great satisfaction and may have improved their overall problem-solving skills. These lessons were also taught with the help of non-technical undergraduate majors, who were able to provide valuable insight on learning programming with no previous experience.

We are pleased to report that our intervention created a total of 1300 student-learner contact hours. This means that each SA learner received about 15 hours of direct instruction. We were able to impact a total of 90 different learners spread out over two different primary schools. The relationships formed and insight gained will surely guide Mercer through future years of continued collaboration.

The Intervention in Numbers

13	Undergraduate students participating
2	South African primary schools
3	Weeks of teaching in country
90	South African learners participating
1300	Learner-contact hours
10	CS, IST and Engineering majors
3	Humanities majors

5.3 Final Points

- Having joint instruction with adults from the local community, who children already know, builds trust and helps to communicate in language the children are likely to understand.
- Cloud-based OERs are effective tools for teaching in impoverished areas with limited resources.
- Encouraging students to search for solutions on their own may help build problem solving skills.

- Non-technical majors can be helpful in designing CS curriculum as they provide critical perspective that technical majors may not have.
- Abundant learning can take place with meager resources.

5.4 Future Plans

Plans for a follow-up MOM to SA during the summer of 2021 are underway which include coordination with Mercer's College of Education. Getting education majors involved will prove beneficial all around. We are actively exploring ways to encourage more local teachers to prepare for and teach CS in Georgia. We have also begun to build a website to coordinate, and assist in delivering *intro* and *explore* lessons. One goal for the website is to make it easy for teachers to add new *explore* lessons associated with existing OER's *intro* lessons. We also envision including an anonymous chat feature where learners could assist each other. And long term, we are investigating ways to conduct research in learning differences between vastly different populations.

Future Work and Improvements

5.5 Next year's MOM

We were approved by Mercer University to return to South Africa to continue our work and build on relationships already formed. One of the greatest enhancements to next years MOM trip will be the inclusions of Mercer' s College of Education.

5.6 Work in Macon

Several students on the trip expressed a desire to continue teaching computer science to middle-schoolers in our local community in Macon, GA. We plan on looking into partnering with some existing programs to give students an opportunity to continue this work.

5.7 Continued Development of Mercoder

Although Mercoder was originally generated as a means of facilitating instruction in South Africa, we want to continue to develop the website to be a tool for educators all over the world. As we have mentioned before, we do not aim to create an entirely new toolset for teaching computer science. The goal of Mercoder is to compile these resources and to create some lessons that one, show off these great resources, and two, show how to move from using these resources to creating your own projects.

References

[1] Computer Science Education Research Group at the University of Canterbury. About - cs unplugged. https://csunplugged.org/en/about/, 2020.

[2] Computer Science Education Research Group at the University of Canterbury. Kidbots. https://csunplugged.org/en/at-home/kidbots/, 2020.

[3] CodeCombat. About codecombat - engaging students, empowering teachers, inspiring creation | codecombat. https://codecombat.com/about, 2019.

[4] Code.org. About us | code.org. https://code.org/about, 2020.

[5] Repl.it. Repl.it - about. https://repl.it/site/about, 2020.

[6] The Scratch Team. Scratch - about. https://scratch.mit.edu/about.

On the Correlation Between Homework Problems and Test Code-Completion Questions in a Data Structures Course*

Jose Cordova, Virginia Eaton, Tyler Greer, Lon Smith
Department of Computer Science and
Computer Information Systems
University of Louisiana at Monroe
Monroe, LA 71209
{cordova, eaton, greer, lsmith}@ulm.edu

Abstract

This paper describes a study designed to compare the performance of students on homework problems with their performance on similar exam questions in an Introduction to Data Structures course. It focuses on code selection/code completion problems and the performance of 132 students from Fall 2017 through Fall 2019. The results indicate that, with the exception of linked list traversal exercises, there is little or no correlation between performance on homework problems and scores on the corresponding exam questions. One should be careful in interpreting these results by characterizing code selection/completion homework questions as useless. Instead, the results should motivate future research to investigate the possible causes for the lack of correlation.

1 Introduction

The attitude of many students toward the correspondence between what they studied and what is on an exam can be summed up by the Venn Diagram shown in Figure 1 below. Of course, their professors would disagree. Most professors would argue that their exams accurately reflect what they taught

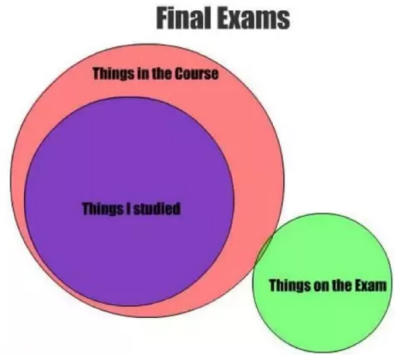


Figure 1: Common student perceptions

in the class, but do they? How do professors link their exams and what they teach in class? A major way of linking the material taught with exams is through homework. Ideally, the problems assigned for homework are models for similar problems on exams. Is this linkage successful? Do students who do better on homework do better on similar exam questions? This paper will examine these questions as they relate to teaching data structures to computer science majors. In particular, the paper will consider code selection and code completion questions. The paper will compare the performance of students on homework problems with their performance on similar exam questions. It will compare the performance of students from Fall 2017 through Fall 2019 for our Introduction to Data Structures course.

2 Background

Code completion problems on exams require students to have developed a degree of code comprehension through study and practice acquired in programming labs and homework assignments. How do professors ensure that their students are developing code comprehension? Cutts and his associates [1] examined this question through the development of Thinkathon activities for students in an introductory programming class. While they felt that their course design included the development of conceptual understanding and code comprehension, they also believed that students needed a more immersive experience to prepare them for the final exam and follow-on programming classes. All of the Thinkathon exercises were paper-based to break the dependence of students on computers. The researchers concluded that there is huge value in students thinking and working on programming exercises away from computers

and on their own without the help of their peers and their instructors.

Zingaro and Porter [5] also looked at ways to improve student performance on the final exam. In particular, they investigated the effect of in-class learning on success on the final exam. The researchers used isomorphic questions (i.e. questions that test the same concept) in class and on the final exam. For each lecture, the researchers chose a pair of isomorphic questions focused on a particular concept. For the first question, they used a Peer Instruction (PI) format. Following the PI format, they asked the class to answer the question individually and then discuss it with their classmates. The normal PI format would have them answer the same question after the discussion. Instead the researchers asked a second isomorphic question. Finally, a third isomorphic question was placed on the final exam. They concluded that isomorphic questions provide strong evidence that the in-class PI process helps students to learn and retain that learning for the final exam.

Petersen and his associates examined the contents of final examinations in the introductory programming course at fourteen North American institutions [3]. In their research, the authors considered different types of questions and attempted to identify the skills and concepts related to each question. As a result, they proposed a classification of exam questions into four categories: writing code, reading code, programming concepts, and non-programming. A key conclusion of their work is that code-writing questions -–which require understanding a larger fraction of concepts—were both more frequent and carried greater weight than any of the other question types.

Morrison and her associates examined the issue of data structure questions on final exams. In their first paper [4], they looked at 76 final exams from a variety of institutions around the world to see if there is a consensus on what should be taught in a data structures class. In the second paper [2], they looked at whether final exam questions actually required students to apply their knowledge of data structures. They looked at 59 data structures final exams and found that only 36 required the application of data structures. They concluded that questions that require the application of data structures are very important to assess student knowledge and should be based on similar homework problems.

3 Project Goals

The aim of this research is to do a comparative study of the performance of students from Fall 2017 through Fall 2019 in our Introduction to Data Structures course. The research is focused on specific types of questions that require students to apply their understanding of data structures by producing code segments that manipulate particular structures. The objective of this research

is to answer these questions:

- Is the linkage between homework problems and similar exam problems successful?
- Do students who do better on homework do better on similar exam questions?

4 Methodology

Our study is focused on students' performance on code completion/selection homework problems and corresponding test questions. For context, the Introduction to Data Structures course is the last in a three-course introductory programming sequence and is taught using Java as the implementation language. Homework questions are assigned after the corresponding material is discussed and similar exercises are solved in classroom sessions. Students typically have three days to submit homework solutions using our electronic learning management system. Test questions are administered electronically in a time-constrained, proctored environment using a secure browser. The primary method used to address our research questions relied on the computation of Pearson correlation coefficients to determine the strength of the relationship between two variables: scores obtained in assigned homework problems and scores obtained in the corresponding test questions.

As stated before, the course in question uses Java as its implementation language. Therefore, the homework problems and test questions involved in this study rely on standard Java Collections Framework interfaces, classes, and API. While a valid argument can be made advocating for language independence when teaching and testing knowledge of fundamental data structures concepts, we feel it is important for students to have the ability to test their solutions using an actual programming language.

Data was obtained by tabulating anonymized scores from 132 students who completed the course in five semesters (from Fall 2017 through Fall 2019). For the purposes of this study, six code completion/selection questions were chosen and numbered sequentially from 1 to 6. The selected questions, which were included in exams in each of the five semesters, cover standard material considered fundamental in courses at this level: linked lists, iterators, stacks, queues, and recursion. As summarized in Table 1 below, homework problems present the student with partially completed code segments and ask the student to enter missing expressions or statements (hereafter referred to as code completion questions), while test questions –with the exception of questions 3 and 4– ask students to select appropriate statements or expressions from a list of possible choices (hereafter referred to as code selection questions). It is our contention that use of code selection/completion questions contribute to the

assessment of important concepts while avoiding some of the issues identified in [3] when students are asked to write significant pieces of code in an examination, since errors caused by low-level concepts affect the ability to demonstrate understanding of higher-level concepts.

Table 1: Selected Questions Summary

Question	Topic	Homework Question Type	Test Question Type
1	Linked List Traversal	Code Completion	Code Selection
2	Traversal using Iterator	Code Completion	Code Selection
3	Stack Manipulation	Code Completion	Code Completion
4	Queue Manipulation	Code Completion	Code Completion
5	Recursive Method 1	Code Completion	Code Selection
6	Recursive Method 2	Code Completion	Code Selection

It is worth noting that homework questions are typically accompanied by starter code files in order to facilitate implementation and testing of the student' s solution. As shown in Figure 2, which contains a snapshot of homework question 4, students are encouraged to use the starter files to compile and test their code prior to entering the corresponding answers in the submitted solution.

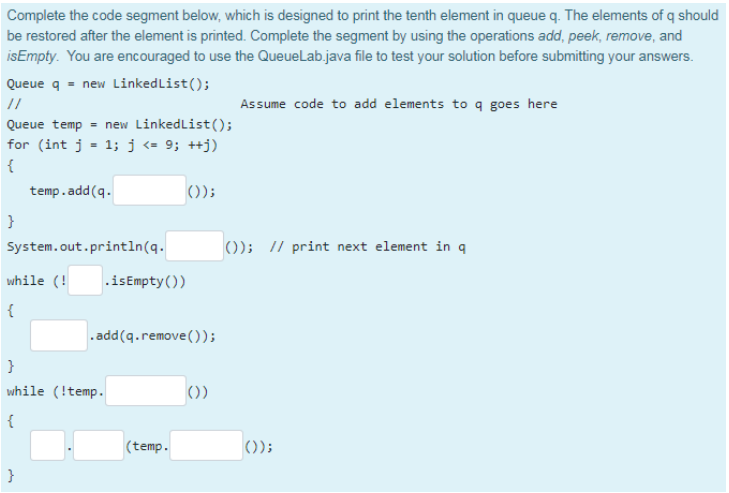


Figure 2: Homework Question 4

Figure 3 presents a snapshot of the corresponding test question. In contrast

to the homework question above, since students are working under time constraints without the ability to compile/test their solution, the question includes sets of choices for each required answer.

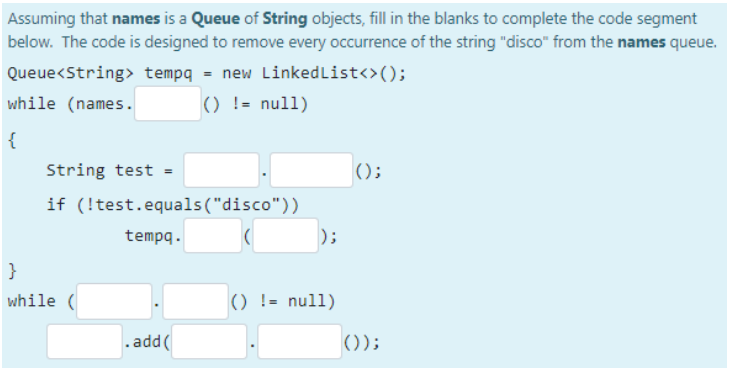


Figure 3: Test Question 4

Every effort has been made to link the concepts explored in homework problems with the corresponding test questions. As a second example, consider the code segment presented in Figure 4, corresponding to question 5 (Recursive Method 1).

As was the case with the first example, the corresponding test question (shown in Figure 5) is designed to assess the same skills while providing a set of possible choices for each required entry.

5 Results

After extracting the scores earned by each student in the selected questions and scaling them into a 0-1 range, a Pearson correlation coefficient was computed between the homework problem score and corresponding test question score for each topic, with n = 132 in all cases. The results are summarized in Table 2. As can be observed, homework scores and test question scores were found to be moderately positively correlated for question 1, corresponding to the linked list traversal exercises, $r(130) = .18$, $p = .0348$ (significant at $p < .05$). For questions 2-6, the correlation coefficients were extremely small –and in three cases negative—but not significant at $p < .05$.

Complete the recursive method below.

```
/**
This method determines whether the integer argument contains 7 as one of its digits.
@param number the value to be tested
@return true if the argument contains the digit 7, false otherwise
*/
public static boolean contains7(int number)
{
   if (number == 0)
      return          ;
   int lastDigit = number % 10;
   if (lastDigit ==      )
      return          ;
   else
      return contains7(                );
}
```

Figure 4: Homework Question 5

The method count7s accepts a non-negative integer n as argument and returns the number of times the digit 7 appears in n. Complete the method so that it accomplishes its task using recursion.

```
public static int count7s (int n)
{
    int count = 0;
    if (          )
        return 0;
    int digit =          ;
    if (digit == 7)
       count = 1;
    return          + count7s(          );
}
```

n == 0 | n > 0 | n >= 0 | n == 7

n / 10 | n % 10 | n % 7 | n / 7

count | 1 | digit | n

n | n / 7 | n % 10 | n / 10 | n % 7 | n - 1

Figure 5: Test Question 5

Table 2: Summary of Results

Question	Topic	Correlation Coefficient
1	Linked List Traversal	0.1839
2	Traversal using Iterator	-0.0230
3	Stack Manipulation	-0.0284
4	Queue Manipulation	0.0757
5	Recursive Method 1	0.0739
6	Recursive Method 2	-0.0521

6 Threats to Validity

There are any number of factors that must be taken into account when interpreting the results presented above. First, as any instructor knows, students' performance on test questions can be affected negatively for reasons such as test anxiety and time constraints, which may be largely unrelated to their understanding of the material. Second, when solving homework problems, students have the ability —and are probably expected to—consult legitimate sources such as notes, textbooks, and related internet sources. In contrast, the exams administered in our course expressly prohibit access to such resources during test taking sessions. In addition, the availability of starter code to accompany homework problems is intended to encourage students to use a compiler to implement and test their solutions before submitting homework answers. While we have considered ways to allow students' use of a compiler during exams, this was not the case during the period used as the basis for this study. However, it must be noted that the use of code selection questions —and to a lesser extent, code completion questions—in exams is partially designed to compensate for the lack of an available compiler. Lastly, the issue of academic dishonesty is one that must be considered, as students increasingly have answers to homework questions produced by their peers, by students who have taken the course in previous semesters, and by contributors to internet sites that profit from storing and making solutions available to their audiences.

7 Conclusion

The purpose of this study was to determine to what extent – if any – students' performance on homework problems correlates with their performance on corresponding test questions. In particular, this paper focused on code selection/code completion questions in the context of an introductory course in data structures. As shown in Table 2, with the exception of the linked list traversal

exercise, the most salient conclusion is that there is little or no correlation between performance on homework problems and scores on the corresponding homework questions.

One should be careful in interpreting these results by characterizing code completion homework questions as useless. Rather, we are motivated to dig more deeply and investigate the possible causes for the lack of correlation. Is this result unique to code selection/code completion problems, or are similar results to be expected with other questions types, such as code tracing or code writing? Can the lack of correlation be attributed to the fact that, in solving homework problems, students have a variety of resources available (compilers, books, internet sources, classmates, etc.) while none of these are available during exam sessions? While a thorough examination of these questions is beyond the scope of this paper, we believe that the answers are important for computer science educators and that further research in this area is warranted.

References

[1] Q. Cutts, M. Barr, M. Ada Bikanga, P. Donaldson, S. Draper, J. Parkinson, J. Singer, and L. Sundin. Experience report: thinkathon - countering an 'i got it working' mentality with pencil-and-paper exercises. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 203–209, New York, NY, USA, 2019. ACM.

[2] B. Morrison, M. Clancy, R. McCartney, B. Richards, and K. Sanders. Applying data structures in exams. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 353–358, New York, NY, USA, 2011. ACM.

[3] A. Petersen, M. Craig, and D. Zingaro. Reviewing cs1 exam question content. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 631–636, New York, NY, USA, 2011. ACM.

[4] B. Simon, M. Clancy, R. McCartney, B. Morrison, B. Richards, and K. Sanders. Making sense of data structures exams. In *Proceedings of the Sixth International Workshop on Computing Science Education*, pages 97–106, New York, NY, USA, 2010. ACM.

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

Searching for All Polygons in a Geometry Figure*

Chase Shaner, Chris Alvin
Computer Science Department
Furman University
Greenville, SC 29613
{chase.shaner, chris.alvin†}@furman.edu

Abstract

We consider the problem of identifying all convex and concave polygons in a geometric figure. We present two such algorithms. The first approach examines sets of segments in a powerset-style construction. The second approach takes a more holistic approach combining select pairs of polygons into new polygons. We then provide empirical evidence investigating the limitations of each approach as they search the space of polygons in a geometric figure. Our goal is to provide a large, finite space for further exploration in an Algorithms or an AI course.

1 Introduction

We consider the problem of identifying *all* polygons (both concave and convex) in a geometric figure. For example, in Figure 1, our goal is to identify all 12 triangles (e.g., $\triangle ABC$, $\triangle BCD$, etc.), 3 quadrilaterals ($DECB$, $ACXB$, and $AEXD$), and 6 concave pentagons ($DECBX$, $DEXCB$, etc.). We present and discuss two distinct search techniques to solving this problem as an example of the importance of reducing the size of a search space, even though, as we

†Corresponding author

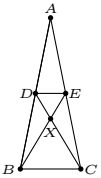


Figure 1: A motivational geometry figure.

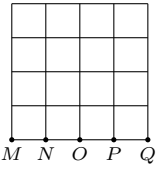


Figure 2: A complex geometry figure.

will show does not control potential unboundedness. Our intent is to present a real-world problem with two solutions that can be viewed as a search of a vast search space or as a search in a much-reduced space, respectively. We believe it is a critical part of artificial intelligence education (and other computational areas) to demonstrate reducing a search space while also demonstrating the limitations of computing on a single machine. In this case we are focusing on a computational geometry problem applicable to, for example, intelligent tutoring systems that feature problem generation in the spirit of [1, 2].

Our first approach identifies polygons using a semi-exhaustive combination of all segments. Valid polygons are identified by determining if the given set of segments makes a non-overlapping cycle. For example, AD, AE, and DE result in $\triangle AED$ while segments AE, BD, and AD do not result in a triangle. This combinatorial approach results in an exponential search space even for smaller geometric figures.

The second approach takes a more holistic computational geometry approach by identifying the smallest polygons in a given figure and then combining those basic polygons into larger polygons. In Figure 1, the first phase of the algorithm identifies triangles $\triangle AED$, $\triangle DEX$, $\triangle DXB$, $\triangle ECX$, and $\triangle XCB$. The second phase exhaustively combines polygon pairs that share segments or subsegments into one larger polygon. For example, $\triangle ECB$ will result from $\triangle XCB$ and $\triangle ECX$ since they share side XC. As a second example, $\triangle DEB$ and $\triangle ECX$ each share a subsegment of each triangle; combining results in concave pentagon $DECXB$.

2 Preliminaries

We begin with some terminology before describing our algorithms. A *minimal segment* is a segment which does not share internal points with any other segments. For example, Figure 1 contains 10 minimal segments (AD, DE, CX, etc.) whereas AB is not a minimal segment since point D lies between the endpoints.

We define a *polygon* as a Jordan curve [6] consisting of a finite set of segments. A *face polygon* (basis polygon in [7]) is a polygon that contains no other polygon. For example, in Figure 1, there are 5 face polygons: $\triangle AED$, $\triangle DEX$, $\triangle ECX$, $\triangle XCB$, $\triangle DXB$. Our goal is to identify the set of all polygons in a given figure, not simply face polygons. Therefore, we say that a polygon that consists of more than one face polygon is a *decomposable polygon*. For example, as shown in Figure 1, quadrilateral $AEXD$ is a *decomposable polygon* because it consists of $\triangle DEX$ and $\triangle AED$.

3 Algorithm 1: Combinatorial segment approach

One approach to identifying the set of all polygons is to consider combinations of segments in a figure. We begin with the set of all segments in a figure–a larger set of segments than may be expected. For example, in Figure 1, there are 10 minimal segments between individual vertices, but also include 4 super-segments built from the minimal segments (e.g., AB is a unique segment consisting of AD and DB). While these super-segments do not create an onerous search space, it is easy to surmise that a more complex geometric figure will result in many segments overall. Consider segment MQ in Figure 2. Along with subsegments created by internal collinear points N, O, and P, there are a total of $\binom{5}{2} = 10$ subsegments. Thus for Figure 2 there are a total of 100 segments to consider. If we identify the set of all triangles in Figure 2 we must consider $\binom{100}{3} = 161700$ different possible triangles even though it is clear that none exist. For quadrilaterals, the set of all possible quadrilaterals has cardinality $\binom{100}{4} = 3921225$ and in the case of Figure 2 we would need to consider the entire space. However, in cases such as Figure 1, we do not need to consider $\binom{14}{4} = 1001$ quadrilateral cases. This is due to the fact that if a set of segments forms a triangle, that same set of 3 segments cannot be used to form a single polygon of more than 3 sides. We generalize this idea of a minimal set of segments in Definition 1.

Definition 1. *Let S be a set of segments that form a polygon. We say S is* minimal *provided there does not exist any proper subset $T \subset S$ such that T forms a valid polygon.*

Algorithm 1 is an exhaustive bottom-up construction of polygons based on combining sets of segments. The algorithm begins by constructing all triangles using sets of three segments on Line 4. If a candidate set of three segments successfully forms a polygon, we save that set (in $\mathcal{P}$) otherwise we save the set as a 'failed' polygon in $\mathcal{F}_n$. Using the sets of segments that fail to form triangles, we inductively consider sets of segments of size 4, 5, etc. until (Line 9) we have identified all polygons or reached the limit on the constituent number

Algorithm 1 Polygon identification through sets of segments

```
1: S: Set of segments ; MS: Maximum number of sides for a polygon
2: function POLYGONS(S, MS)
3:     P ← ∅; F_n ← ∅          ▷ Polygons identified; Sets that are not polygons
4:     for each {s1, s2, s3} ⊆ S do          ▷ For each set of 3 segments
5:         if ISPOLYGON(s1, s2, s3) then P.add({s1, s2, s3})
6:         else F_n.add({s1, s2, s3})
7:     return INDUCTIVECONSTRUCT(P, S, F_n, ∅, MS)
8: function INDUCTIVECONSTRUCT(P, S, F_n, F_{n+1}, MS)
9:     if F_n = ∅ or |F_n[0]| ≥ MS then return P
10:    for each F ∈ F_n do                    ▷ For each non-polygon set
11:        for each s ∈ S do    ▷ Can adding a segment result in a polygon?
12:            if ISPOLYGON(F ∪ {s}) then P.add(F ∪ {s})
13:            else F_{n+1}.add(F ∪ {s})
14:    return INDUCTIVECONSTRUCT(P, S, F_{n+1}, ∅, MS)
```

of sides (MS) in the identified polygons. In the loop beginning on Line 10 we construct quadrilaterals (pentagons, etc.) by taking a failed sets of 3 segments and sequentially adding one more segment results in a polygon. If not, we add the set of 4 segments to a new list $\mathcal{F}_{n+1}$ on Line 13; 5-sided polygons are then constructed with a recursive call to INDUCTIVECONSTRUCT with sets of failed quadrilaterals in $\mathcal{F}_{n+1}$.

Lemma 3.1. *Algorithm 1 guarantees polygons consisting of minimal sets of segments.*

Proof. The loop starting on Line 4 constructs all 3-polygons using minimal sets of segments. INDUCTIVECONSTRUCT is then invoked with sets consisting of 3 segments that are not minimal; hence, each $F \in \mathcal{F}_n$ on Line 10 is not minimal. For each segment $s \in S$, if $F \cup \{s\}$ is not a minimal polygon, it is added to $\mathcal{F}_{n+1}$ on Line 13. Hence, all sets in $\mathcal{F}_{n+1}$ are not minimal and the recursive call maps $\mathcal{F}_{n+1} \to \mathcal{F}_n$ ensuring again $\mathcal{F}_n$ contains non-minimal sets. Thus, for $F \in \mathcal{F}_n$ consisting of n segments, the inductive process only considers polygons of minimal set size $n + 1$. □

4 Algorithm 2: Combining existing polygons to form polygons

Our second approach for identifying all polygons in a figure first identifies face polygons and second continues to combines those polygons into new polygons

Algorithm 2 Polygon identification using stitching

```
1: S: Set of segments
2: MS: Maximum number of sides for a polygon
3: function POLYGONS(S, MS)
4:     P<Segment, Polygons> ← ∅          ▷ All polygons containing a given segment
5:     pg ← BUILDPLANARGRAPH(S)          ▷ Graph corresponds to input segments
6:     FP ← FACEIDENTIFICATION(pg)       ▷ Identify 'basic', face polygons
7:     INITIALIZE(P, FP)   ▷ Initialize polygon dictionary with face polygons
8:     q ← FP              ▷ Combine polygons beginning with face polygons
9:     while q ≠ ∅ do
10:        p ← q.DEQUEUE()
11:        for each s ∈ p do                          ▷ For each side of a polygon
12:            for each p' ∈ P[s] ∪ P[SUBSEGMENTSOF(s)] do
13:                p'' ← STITCH(p, p')   ▷ Combine two polygons sharing a side
14:                if p'' ∉ P then   ▷ or subsegment of a side
15:                    P.ADD(p'')        ▷ Add new poly to set of polygons and worklist
16:                    q.ENQUEUE(p'')
17:    return COLLECT(P)       ▷ Extract and return unique polygons in the dictionary
```

by a process we call *stitching*. Algorithm 2 takes as input a geometry figure represented as a set of minimal segments S. We begin on Line 5 by constructing a planar graph pg corresponding directly to S. On Line 6, face identification is executed as described in [3, 5] and returns the set of face polygons FP.

To identify all *decomposable polygons* in a figure described by S, our algorithm exhaustively combines all polygons with all other adjacent *polygons*. We begin with the set of face polygons, FP, by initializing the set of identified polygons $\mathcal{P}$ (Line 7) as well as the queue of *polygons* to process (Line 8). Our goal is to exhaustively combine adjacent, non-overlapping polygons in $p \in q$ with all of the currently known polygons in $\mathcal{P}$ that share a side with p; we do so on Line 10 through Line 16. The process of combining two polygons together into a *decomposable polygon* we refer to as *stitching*; we describe this process via some examples below.

Stitching is the process of identifying all segments shared between two polygons, and creating a new polygon from the segments not shared between the two polygons. Two polygons are adjacent if they share a side or if a side of one polygon comprises a subsegment of a side of the other polygon. For ex-

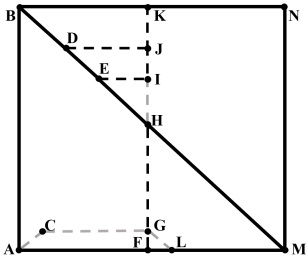


Figure 3: A demonstrative geometric figure for the *stitching* operation.

ample, $\triangle AED$ and $\triangle DEX$ in Figure 1 explicitly share segment DE. In the second case, $\triangle ECX$ consists of side segment XE while XE is a subsegment of $\triangle DEB$. In both cases, we may stitch the polygon pairs together: $\triangle AED$ and $\triangle DEX$ result in quadrilateral $AEXD$ while $\triangle ECX$ and $\triangle DEB$ result in concave pentagon $DECXB$.

We describe three cases for stitching combinations of face polygons and decomposable polygons as well as cases when the operation fails to produce a polygon.

Face-Face. Consider face polygons $KJDB$ and $JIED$ that share side JD in Figure 3. By combining the sides of $KJDB$ (KJ, KB, etc.) and $JIED$ (JI, IE, etc.), minus the shared side, JD, we find the *decomposable polygon* $KIEB$.

Face-Decomposable. In Figure 3, face polygon $BKIE$ is combined with decomposable polygon $KNMH$ since the two share segment KI. Stitching results in the concave, six-sided polygon $BNMHIE$. We observe that KI is an entire side of $BKIE$, but is a subsegment of side KH in $NMHK$.

Stitching is a process that may combine two 'complex' polygons into a 'simpler' polygon. In particular, stitching (1) may take two polygons with many sides and result in a polygon with fewer sides and (2) may eliminate more than the shared segment between two polygons. For example, in Figure 3 stitching the 7-sided concave polygon $BNMEIJD$ with quadrilateral $JIED$ results in $\triangle NMB$.

Decomposable-Decomposable. In Figure 3, quadrilateral $BHFA$ and $\triangle HMF$ are *decomposable polygons* that share side HG (and do not share any face polygons). By removing HG, these polygons stitch into *decomposable polygon* $\triangle MAB$ which encapsulates all of the *face polygons* of its predecessors ($CGFA$, $\triangle GLF$, etc.).

Stitching considerations. On its face, stitching is a simple operation. However, there are special cases for which we are not interested in stitching two polygons. Our first case is when two decomposable polygons overlap. In Figure 3, we would not stitch $\triangle BMA$ with pentagon $KNMLG$ because both

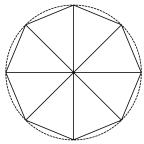


Figure 4: Sample circle-based construction of a geometry figure with increasing numbers of triangles within a circle.

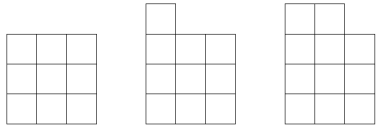


Figure 5: Progression of three geometric figures with 3 fixed columns; one square is added each iteration.

decomposable polygons share $\triangle HMF$. We identify this overlap by taking the intersection of the sets of constituent face polygons in a decomposable polygon; if the intersection is non-empty, we do not stitch. We argue the correctness of this approach since face polygons are minimal and must not contain any other polygons.

The second case we describe attempts to combine two polygons resulting in a shape that is not a polygon according to our definition of a polygon as a Jordan curve. Consider the concave polygon $BNMEIJD$ and the trapezoid $BHFA$. The two polygons share two distinct subsegments BD and EH. If we stitch the polygons together, the result is a shape with a 'hole' on the interior defined by quadrilateral $DJIE$. The resulting shape is not a Jordan curve and hence is not a polygon. Thus we must be mindful of stitching polygons when combining concave polygons.

5 Experimental Analyses

We implemented our algorithms in Java and executed our experiments on a 64-bit 1.2Ghz Intel processor with 16 GB running Windows 10. We will demonstrate empirically that Algorithm 2 is superior to Algorithm 1 in its efficiency and capacity for processing more complex figures, but also has its limitations. We used a corpus of 10 geometry figures to verify the correctness of the algorithms: both identified the same set of polygons in each figure, which we verified by hand. Since Algorithm 1 is prone to exceed memory limitations, our corpus of figures consisted of simpler geometry figures (e.g., Figure 1).

Geometric figures. We describe some of our geometric figures we used as a testbed for our analyses of Algorithm 1 and Algorithm 2. Our first geometric figure is a set of increasing triangles oriented around a circle. Figure 4 is a figure with 8 face polygons oriented around the center of a circle (the circle is dashed and provided as reference). Creating a geometric figure with an additional triangle only requires computing a central angle and the associated points on

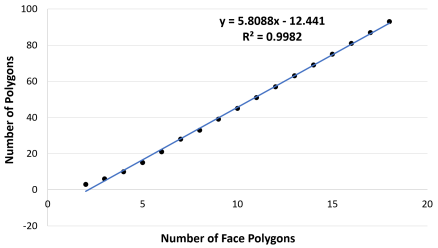


Figure 6: Total polygons compared to face polygons in circular sequence of triangles (a la Figure 4).

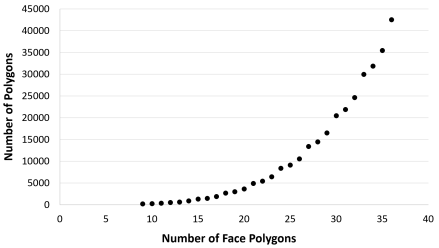


Figure 7: Total polygons compared to face polygons in an increasing grid of squares (a la Figure 5).

the circle. In this construction we can explicitly control the number of face polygons in our geometry figure.

Our second progressive geometry figure is based on a sequence of squares. In Figure 5 we observe a geometry figure consisting of a 3×3 grid. Again, controlling the number of face polygons, the next progressive geometry figure in Figure 5 has one additional square. The sequence of geometry figures progresses in this fashion adding a single square at a time, always maintaining three columns.

Controlling face polygons and total number of polygons. We argue that the complexity of a geometric figure is related to the number of face polygons and the corresponding number of total polygons. Before we put our algorithms through their paces, we consider the relationship between face polygons and total polygons with respect to our geometry figure progression from Figure 4 and Figure 5. We observe in Figure 6 a very strong linear correlation ($r^2 = 0.9982$) between the number of face polygons and the total number of polygons.

We consider the progressive grid of squares shown in Figure 5. There are two aspects to consider with this figure. We see a nonlinear relationship between the number of face polygons and total polygons: $r^2 = 0.9908$ for a quadratic curve. A construction of such a geometric figure provides a more stressful, but controllable testing grounds for our algorithms. In the second case, we ask the reader to consider every third point in Figure 7. We observe that the total number of polygons increases more when adding a face polygon completes a row compared to when adding a face polygon results in an 'incomplete' row. For example, the increase in total polygons from 29 face polygons to 30 face polygons, 3975 polygons, is greater than that found between 30 face polygons and 31 face polygons, 1441.

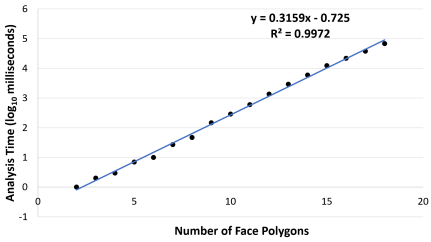


Figure 8: Execution times for Algorithm 1 of triangles arranged in a circle (a la Figure 4).

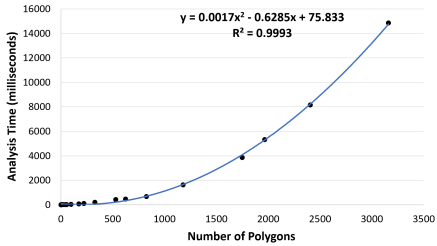


Figure 9: Execution times for Algorithm 2 *stitching* a sequence of square grids of square face polygons.

Algorithm 1. As Algorithm 1 is a powerset-based analysis of segments, stress testing was difficult due to excessive memory usage. Our initial stress test consisted of a square grid-based figure similar to that shown in Figure 2. However, such a figure does not contain any triangles and thus all possible sets of size 3 segments are considered when identifying the constituent quadrilaterals (and other even-sided polygons).

We executed Algorithm 1 setting the maximum number of sides (MS) to identify all polygons up to octagons on a sequence of triangles oriented around a circle (Figure 4). For 18 face polygons (containing 93 total polygons) in Figure 8, Algorithm 1 took approximately 69 seconds; we do not have data beyond that point due to heap space being exceeded. We argue that allocating more heap space simply delays the limitation of Algorithm 1. With an extremely strong linear correlation ($r^2 = 0.9972$) of log-transformed data in Figure 8, we are confident that our empirical data aligns with the theoretical notion that constructing and analyzing subsets is an exponential operation in the number of segments. Since the number of segments in a simple figure is large, Algorithm 1 is not a strong, general approach to identifying polygons. We note Algorithm 2 identified polygons in each of the test cases in less than one second.

Algorithm 2. We initially executed Algorithm 2 using an increasing sequence of square face polygons in a square configuration. That is, we started with a 1×1 face polygon *adding one square at a time*. After 3 iterations, the geometry figure would be a 2×2. After 5 more iterations, the geometry figure would be a 3×3, and so forth. Algorithm 2 was executed seeking polygons with 8 or fewer sides. We did not encounter memory problems executing Algorithm 2; time was our main restriction. We observe a quadratic polynomial relationship in Figure 9 with $r^2 = 0.9993$ as might be anticipated in a queue-based pairing algorithm. The largest grid in the experiment contained 19 face

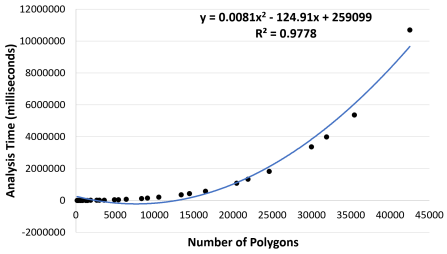


Figure 10: Time required for Algorithm 2 *stitching* on a sequence of grids with a fixed row length.

polygons, 3160 total polygons, and took *stitching* approximately 15 seconds. The next grid (4×5) containing 20 face polygons, could not be completely processed by Algorithm 2 after 4 hours of analysis. We attribute this steep increase in processing time to the fact that 20 face polygons would have resulted in a grid with a newly completed row similar to our discussion of Figure 5. Thus, the increase in total polygons to be stitched was disproportionately larger than any previous iteration.

Noting the ceiling from the previous experiment, we conducted a similar experiment using a grid construction as described in reference to Figure 5 fixing the number of columns at 3. The largest grid processed in this experiment was a 3×12 grid of squares containing 36 face polygons and 42510 total polygons taking approximately 178 minutes to complete *stitching*. We observe in Figure 10 processing time for Algorithm 2 growing as a quadratic ($r^2 = 0.9778$).

6 Related Works

Identifying faces and polygons in a geometric figure is well-studied, but identifying all such polygons is not. Laha, et al. [7] used face identification for two dimensional figures of lines by exhaustively cycling through points of intersection and creating convex chains of n-sides from the resulting segments; we can view these chains as cycles in a graph. Their exhaustive technique performs their construction in parallel compared to our approaches executing with a single thread of control.

While our approach to polygon identification focuses on a two-dimensional geometric figures, Dobkin et al. [4] considers multiple dimensions. Using face identification and a procedure similar to our stitching algorithm, Dobkin et al. identify all empty, convex polygons with multi-dimensional points using a 'Visibility Graph': cycling through all points in a set and constructing edge chains of less than or equal to n-sides between subsets of points which contain

no points within the covered area. After identifying two dimensional polygons in a visibility graph, their algorithm combines polygons which share an edge or vertex to identify empty convex polygons. While the work of Dobkin et al. is close to our work in spirit by identifying all convex polygons (and in multiple dimensions), our algorithms identify both convex and concave polygons, but in a two-dimensional setting.

7 Conclusions

We have presented two distinct algorithms to identify polygons in a geometric figure. Our first algorithm takes a powerset-style approach and encounters problems with exponential sizes of sets. Our second algorithm is a bit more robust, but was lacking with particular sets of geometric figures. Empirically, both algorithms are successful in exploring restricted, although not equivalent, search spaces. The utility of the algorithms in, for example, problem generation in an intelligent tutoring system may be fruitful, but beyond the confines of figures in high school geometry problems, the algorithms may not suffice for larger geometric figures or planar graphs.

References

[1] Chris Alvin, Sumit Gulwani, Rupak Majumdar, and Supratik Mukhopadhyay. Synthesis of geometry proof problems. In *AAAI 2014*, pages 245–252, 2014.

[2] Chris Alvin, Sumit Gulwani, Rupak Majumdar, and Supratik Mukhopadhyay. Synthesis of problems for shaded area geometry reasoning. In *AIED 2017*, volume 10331 of *Lecture Notes in Computer Science*, pages 455–458. Springer, 2017.

[3] Chris Alvin, Sumit Gulwani, Rupak Majumdar, and Supratik Mukhopadhyay. Synthesis of solutions for shaded area geometry problems. In *FLAIRS 2017*, pages 14–19, 2017.

[4] D. P. Dobkin, H. Edelsbrunner, and M. H. Overmars. Searching for empty convex polygons. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, SCG '88, page 224–228, New York, NY, USA, 1988. Association for Computing Machinery.

[5] David Eberly. The minimal cycle basis for a planar graph, 2015. http://www.geometrictools.com/Documentation/MinimalCycleBasis.pdf.

[6] Camille Jordan. *Cours D'analyse de L'École Polytechnique*. Gauthier-Villars et fils, 1893.

[7] Arijit Laha, Amitava Sen, and Bhabani P. Sinha. Parallel algorithms for identifying convex and non-convex basis polygons in an image. *Parallel Comput.*, 31(3-4):290–310, 2005.

Mutually Exclusive: A Survey of Ethical Decision Making in Technology*

Connor McPherson[1], Joe Dumas[1], Sumith Gunasekera[2], Claire McCullough[3]
[1] Department of Computer Science and Engineering
[2] Department of Mathematics
University of Tennessee at Chattanooga
Chattanooga, TN 37403
Connor-McPherson@mocs.utc.edu
{Joe-Dumas, Sumith-Gunasekera}@utc.edu
[3] Webb School of Engineering
High Point University
High Point, NC 27268
cmccullo@highpoint.edu

Abstract

Computing professionals make hundreds of decisions every day. Developers, security consultants, operations engineers, designers, and computer engineers all make small decisions that affect the final product. The values people choose to promote and ignore appear in the constraints and biases of the products they craft. This paper discusses the process of developing, distributing, and analyzing a values survey for computer professionals and computer science students in East Tennessee. We use advanced calculations of significance and beta for chi-squared tests to determine significance and discuss the ethical conclusions from the survey data.

1 Introduction

Ethics in programming is a hot topic in data privacy [11] and in machine learning [12], but programmers' values also play a significant role in product development and maintenance. When a developer, security consultant, or operations specialist makes any decision about a product, that decision, in part, comes from the programmer's values. Given a buildup of similar choices, developer values can affect the project. One example of this phenomenon is Huff and Cooper's empirical study of sex-bias in design. The researchers asked a group of designers to propose designs for educational software for students. In two test groups, the researchers explicitly specified that the students were male or female, while they told a third test group to propose designs for "students." Huff and Cooper found that the designs proposed by subjects in the gender-unspecified group were empirically similar to the designs proposed for boys and were different from the designs proposed for girls, even among designers who were female [9]. Huff and Cooper showed how preexisting biases and values implicitly influenced the design and gave rise to bias in the software.

Cory Knobel, CEO of RAW Consulting, and Geoffrey Bowker, Director of the Values in Design Laboratory at Bren, warned that "conversations and analyses of the values found in technologies are generally engaged after design and launch, and most users are faced with a daunting set of decisions already made on their behalf" [10].

Friedman and Nissenbaum, in their seminal essay, "Bias in Computer Systems", outlined three kinds of bias in software: preexisting, technical and emergent [6].

- Preexisting bias is societal, systematic bias held implicitly by consumers of a society that disseminates those biases. This includes gender bias, as in Cooper and Huff's research, and racial bias [6].
- Technical bias is exclusion by the constraints of software or hardware, and the design choices made as a result. As Friedman and Nissenbaum explain, "A technical constraint imposed by the size of [an airport monitor] screen forces a piecemeal presentation of flight options and, thus, makes the algorithm chosen to rank flight options critically important. Whatever ranking algorithm is used... the system will exhibit technical bias" [6].
- Emergent bias is the most difficult to spot during design, as it becomes apparent after development with changes to the software's environment after launch, creating scenarios or use cases that designers never had to consider during development [6]. This bias can unveil the values inherent in the designers. Programmers building software for coworkers will see emergent problems if that software is distributed to the public.

Friedman and Nissenbaum explain,

> Envision a hypothetical system designed for a group of airlines all of whom serve national routes. Consider what might occur if that system was extended to include international airlines. A flight-ranking algorithm that favors [flying with the same company for every flight segment] when applied in the original context with national airlines leads to no systematic unfairness. However, in the new context with international airlines, the automated system would place these airlines at a disadvantage and, thus, comprise a case of emergent bias [6].

Don Gotterbarn, current chair of the Association for Computing Machinery (ACM) Committee on Professional Ethics, said that,

> The changes in technology and the kinds and number of impacted stakeholders changed the fundamental nature of society. The development of the cell phone has changed people's access to information and to a wide variety of entertainment ···Computers impact all areas of our lives and many life preserving functions are relegated to a piece of computer guided machinery [7].

If programming as a discipline is to continue having as profound an impact, the decisions programmers think about should not end at the design and maintenance of a project. As Don Gotterbarn continued, "It is not sufficient to limit any computer discipline to addressing purely technical issues. As a profession, we must not retreat behind the obscurity and complexity of computing artifacts. We must acknowledge and embrace our role in shaping society and take responsibility for our part in those changes" [7].

It is because of these concerns that the ACM and other professional societies develop and publish codes of ethics. In the field of computer science, agencies such as the ACM and the British Computing Society release codes of ethics for their members. They promote integrity, professionalism, leadership, and the public good. The ACM describes the purpose of these codes as to "serve as a basis for ethical decision-making." Recent research shows that these codes do not affect programming habits [14]; however, the codes are a comprehensive view of the values programmers parse through in the decisions they make. Our research uses the professional codes of ethics as a lens for the values and internal biases of programmers and how those biases work their way into the products they create, support, and maintain.

This research aims to answer one question: "Is the way programmers respond to ethical scenarios dependent on their age, years of experience, or role as a student or a professional?" The goal of this research is to understand the

individual biases that influence programmers' choices during ethical dilemmas that plague modern programming, as well as to gain an understanding of how to correct those biases.

2 Related Work

To test how programmers respond to ethical situations, this research uses a scenario-based model based on past surveys. The two surveys discussed below use scenarios in their tests and use statistical regressions to interpret their data:

In 1996, Dr. Susan Harrington at Georgia State University studied codes of ethics and the influence on the denial of responsibility, with the conclusion that codes of ethics do have some small effect, especially for people who deny their responsibility to be ethical. While this helps build the case for the use of codes of ethics, it shows that codes are not strong enough to enforce ethical responsibility. Harrington concludes that "at minimum, managers must use a multifaceted approach to deterring computer abuse and not depend upon the simple solution of codes of ethics. The use of tactics, such as codes of ethics, for purposes of general deterrence should not be overstated but should not be discarded" [8].

In 2018, Andrew McNamara, Justin Smith, and Emerson Murphy-Hill surveyed software developers to determine whether the codes of ethics affect professional decision-making. The research concluded that they do not. McNamara's research shows that computer scientists are not significantly affected by codes of ethics; however, independent from the ACM code, the survey did not compare how respondents select responses according to their own biases [14].

Like these two studies, our research includes several ethical scenarios. Unlike these two studies, this research is entirely observational. It does not use a control group with a controlled stimulus. The goal of this research is to observe the ethical climate of programmers through the lens of the codes of ethics. Many of the scenarios used in this survey were adapted from the questions made by McNamara, Smith, and Murphy-Hill, making this research a continuation of their work. The questions in this survey place pairs of ethical values in a mutually exclusive scenario. This ensures that the respondents' results describe how they would react in making everyday decisions that force them to choose between two values. For example, one question says:

Question 0: The last customer meeting for your project was a disaster. Communication has been limited for the last month and the customer is expecting a full report from today's meeting. As you leave your office for the meeting, you overhear the administrative assistant saying,

> "If Joe calls in, please see that he calls home. His spouse says there is a mini-crisis."

You are to meet with Joe at the customer's office, and the two of you are to lead the meeting. Joe's participation is critical. Joe is quite nervous and often gives a bad impression if distracted. What do you do?

- Relay the information to Joe before the meeting
- Not relay the information to Joe before the meeting

This question is one of the scenarios adapted from McNamara, Murphy-Hill, and Smith's survey, except that this version adds the variable of the unhappy customer and how critical this meeting is for keeping the customer informed. This scenario forces a choice between helping one's coworker and one's responsibility to the client, two cornerstone values for many codes of ethics [1, 2, 3, 16].

3 Methodology

The survey categorized questions according to six values that are common to most ethical codes:

- Transparency, the principle of being open and honest to all stakeholders about everything that goes on before and during software production [1, 3, 16].
- Respect for Privacy, the principle of respecting other people's data, sensitive or otherwise [1, 2, 3, 16].
- Respect for Intellectual Property, the principle of honoring other people's work, property, and ideas [1, 2, 3, 16].
- Helping colleagues, the principle of helping one's fellow workers, and teaching them what they need to know to succeed [1, 2, 3, 16].
- Quality assurance, the principle of refusing to release software that falls short of what has been promised in terms of security, usability, and completeness [1, 2, 3, 16].
- Self-improvement, the principle of continual learning in computing, ethics, and the skills of communication [1, 3, 16].

These values were chosen based on their regular appearance through the above codes and their applicability in scenarios that force respondents to choose one over the other. Other values in the codes included competence, quality of life (of all people), social good, and security. These values were not chosen to keep the survey short and because many of them are dependent on, or covered by, the values already in the survey. For example, a programmer who values self-improvement will, by extension, become more competent.

Each scenario in the survey is a multiple-choice question, including two responses that favor one value more than the other and sometimes two other

responses that respect both equally. Respondents picked responses to each scenario based on how they would act in that situation. Along with these scenarios, respondents supplied their age, number of years of experience, student status (whether or not they were currently a student), and how highly they thought they held each value.

This survey was exclusively advertised to programmers in the East Tennessee area, specifically from the Chattanooga region, to get a geographically consistent sample. Southeast Tennessee is unique in the United States due to the region's strong startup support network. Significant contributors to this network are the economic development agency Launch Tennessee and entrepreneur centers such as the Company Lab in Chattanooga. This survey was distributed through these communities, making this survey not just a discussion on the values of programmers, but of programmers in the startup culture of Greater Chattanooga.

4 Results

Data collection officially closed February 13th, 2020, with a total of 90 responses from local professionals in computing. The respondent count, distributed by age, was: 57 respondents of 18-30 years of age; 21 respondents of 31-40 years of age; 12 respondents of 41+ years of age. The distribution of respondents by years of experience was: 38 respondents of 0-5 years of experience; 29 respondents of 6-15 years of experience; 23 respondents of 16+ years of experience. Student status was distributed as follows: 48 non-students; 42 students.

Results were analyzed for significance using the chi-squared test for independence. This is a test that determines whether two categorical factors are related based on a "contingency table" of the counts of each category [13]. The statistical test works by stating a "Null Hypothesis." The hypothesis assumes the categories have no relation to each other with the hope that the observed data will contradict this assumption. If it does, we can say that the categories are related.

The chi-squared test begins by comparing counts between several categories based on the assumed distribution of values if the categories were not related [13]. To do this, the data is compiled into contingency tables. In a contingency table, rows represent the number of responses for a specific question, while the columns represent respondent characteristics (student vs. non-student, experience range, age range). To get statistically significant findings, the chi-squared test uses a selected alpha, which represents the chance of getting a false result. A significant result should have a low alpha, such as 0.05, to lower the chance of a faulty result to 5%.

The chi-squared test assumes that the expected value ($E_{i,j}$) of each cell in a contingency table is greater than 5 for at least 80% of the cells and that all cells are greater than 1 [13]. This can usually be ensured by having more samples than 5 times the number of cells in any table. For our survey, the cell count never exceeds 10, which would make a sample size of 90 acceptable. Unfortunately, for four out of the ten questions, the results are skewed towards one or two responses. Experts advise combining the rows to remove the rows with too few counts [13], but while this validates the use of the chi-squared test, it limits the conclusions we can make on the relations after testing.

For these questions, combining two rows is sufficient, but one of the questions is skewed so far towards one response so that the chi-squared test cannot work, and may not even be needed to see a trend in user responses.

This research used a survey with 10 questions, each of which can be compared against respondent age, years of experience, and student status. Using an alpha of 0.05, we found only one significant result: with 95% certainty (1-alpha), we can say that whether a respondent is a student affects how they answer question zero. To try to find more significant results, we must vary alpha in order to increase the power of the test.

The Null Hypothesis assumes that the categories have no relation to each other, which the data either accepts or rejects. This test can be invalidated if the risk of mistakenly labeling a result becomes too high. This error can be visualized with the following table:

	Should be rejected (Null Hypothesis is false)	Should not be rejected (Null Hypothesis is true)
Test rejects the null hypothesis	True Positive Rate (TPR) = Sensitivity = Power = 1- β	False Positive Rate (FPR) = 1-Specificity = α
Test fails to reject the null hypothesis	False Negative Rate = 1-sensitivity = 1 –Power = β	True Negative Rate (FNR) = Specificity = 1 - size = 1 - α

Type I errors (α) represent seeing correlations where one does not exist. For the previous alpha of 0.05, there is only a 5% chance that our singular result is wrong. The sacrifice for this accuracy is that it increases β, the Type II error rate, resulting in missed significant results. This is especially true for the chi-squared test, which is a low-power test [4]. For this data, a better alpha must be calculated to increase power while keeping alpha acceptably low.

Beta can then be used to calculate the true positive rate, which is the "power" of a function. Power is equal to 1-β. A low beta results in a high power, which is good. To counteract the chi-squared test's naturally low power, this research uses the Youden index of a ROC Curve.

A ROC (Receiver Operating Characteristic) curve is a visual representation of the tradeoff between the true positive rate (power) and true negative rate (alpha) [5]. Figure 1 shows an example of a ROC curve (orange). The diagonal (blue) shows where power (true positive rate) and alpha (false positive rate) are equal. A point on the ROC is better when power is greater and alpha is smaller. This scoring of a point is its "Youden" (J) [15], where J = power – alpha. The point where the Youden is largest is the "optimal cut-point." The alpha value of this optimal cut point is the "calculated alpha," which is a superior value for alpha compared to the nominal alpha, 0.05.

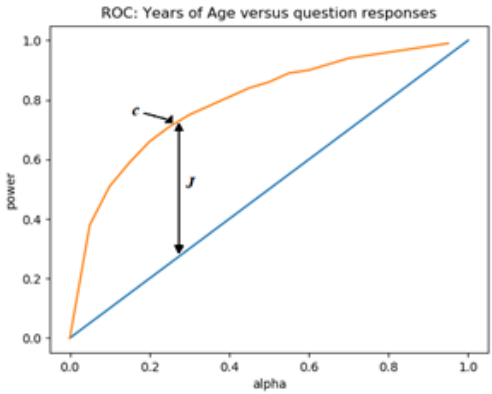


Figure 1: ROC curve depicting the Youden (J) and optimal cut-point (c)

Because power is a function of the chi-squared value of each contingency table, every contingency table has a different optimal cut-point and resulting alpha. Using Python's statistical packages, we calculated each table's optimal alpha before testing each chi-squared value for significance. Applying this approach to revisit every contingency table, we found the following results:

- The response to question 0 is related to whether the respondent is a student (alpha=0.15 power=0.809)
- The response to question 2 is related to whether the respondent is a student (alpha=0.262 power=0.535)
- The response to question 3 is related to whether the respondent is a student (alpha=0.162 power=0.807)
- The response to question 5 is related to whether the respondent is a student (alpha=0.162 power=0.806)
- The response to question 7 is related to whether the respondent is a student (alpha=0.25 power=0.651)

- The response to question 8 is related to whether the respondent is a student (alpha=0.175 power=0.779)
- The response to question 9 is related to whether the respondent is a student (alpha=0.15 power=0.801)
- The response to question 0 is related to the respondent's age (alpha=0.225 power=0.683)
- The response to question 3 is related to the respondent's age (alpha=0.15 power=0.817)
- The response to question 8 is related to the respondent's age (alpha=0.15 power=0.806)

Interestingly, the test failed to find any relation between question responses and the respondents' years of experience. Note that this does not prove that there is no relation between programmer values and years of experience. The chi-squared test works by rejecting the null hypothesis that there is no relation between two factors [13]. We cannot prove that there is no relation when we began the test with that assumption.

While these results are significant, the chi-squared test cannot tell us how they are significant, or what these results signify. The results from the chi-squared test must be studied to find the trends between the independent variable (age and student status) and the dependent variable (question responses). For example, question 0 is a question about valuing one's coworkers versus keeping good relations with one's client:

#5: Quality Assurance v Privacy: You and your coworkers have been working for the last year on an update to an already existing accessibility app to make texting on smartphones easier. The software is used in a wide variety of applications, and you believe there may be issues that haven't been found. The release deadline is approaching, and one coworker suggests configuring the initial release to send an error report of everything being done by the user whenever a system breakdown occurs. This data collection would keep track of all recent events, running apps and current texting channels. Data collection for the sake of improving the software is allowed in the company's privacy policy. What do you do?

- Begin development of the data collection software
- Request to push back the deadline and build a small group of users with whom to test the software
- Release the software without collecting data and wait for users to report errors
- Develop the data collection software to get information on customers for future use and begin work on the next update without checking for errors in the last update

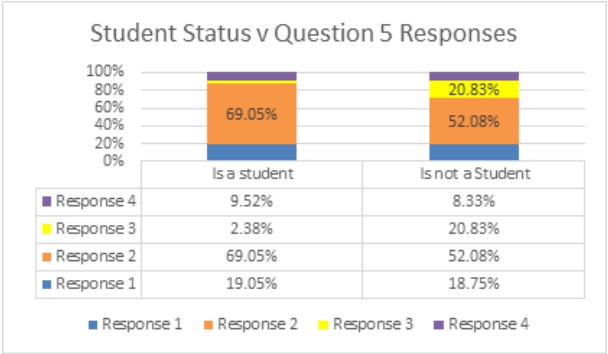


Figure 2: Stacked proportions of the Student Status v Question 5 contingency table

Respondents who are no longer students are more likely to value the privacy of users and quick release of a product over ensuring quality. Figure 2 visualizes this relationship. The relationships for all other tables show similar leanings, which are compiled into these results:

- *Student v 0*: Non-students value coworkers over clear communication with the client.
- *Student v 2*: Non-students are more likely to honor their Non-Disclosure Agreements, even if it means missing a project milestone.
- *Student v 3*: Non-students are more likely to contact customers about issues during project specification, while students are more likely to build the project even with the flawed requirements.
- *Student v 5*: Non-students are more likely to opt to release a product immediately without data-collection software even if it means project bugs go undiscovered.
- *Student v 7*: Non-students are more likely to sell data to third parties, while students are more likely to add an opt-out setting for customers.
- *Student v 8*: Non-students are more likely to value the licenses of privacy-invasive libraries and use them as-is, while students are more likely to attempt to find a different library.
- *Student v 9*: Non-students are more likely to tell employers, rather than customers, about valuable information about the risks a product may have, while students are more likely to tell customers.
- *Age v 0*: Users older than 18-30 years old are more likely to value coworkers over communication with the client.
- *Age v 3*: The older a user is, the more likely he/she is to contact customers about issues during project specification.

- *Age v 8*: The older a user is, the more likely he/she is to value the licenses of privacy-invasive libraries and use them as-is, while 18-30-year-olds are more likely to attempt to find a different library.

The last significant trend is from question 1, which was not calculated due to its uniform skew towards one response: Student v 1: Both students and non-students value helping coworkers over ensuring that a shipped product is usable. The conditional distribution table for Question 1 illustrates this:

Student v Question 1	Not a Student	Is a Student
Response 1	0.00%	0.00%
Response 2	12.50%	11.90%
Response 3	2.08%	2.38%
Response 4	85.41%	85.71%

Question 1 is a question comparing helping coworkers (with whom communications have broken down) and the quality of a product that is shipping tomorrow. Most respondents would ship the product even if it does not work and would instead take time to repair the relationship with their coworkers.

5 Conclusions

From these results, we can infer about the values of programmers in southeast Tennessee:

- Most programmers seem to value their coworkers more than the good of a singular project.
- Older programmers care even more about coworkers than younger respondents do.
- Respondents out of college are more likely to value releasing a project quickly, even at the expense of quality or privacy.
- Respondents out of college are more likely to value intellectual property, while students are more likely to respect privacy.
- Older programmers and non-students value clients more than younger students.

Most of these results match common understandings of programming (such as more experienced programmers being more beholden to their bosses), but it is significant that all programmers, especially older ones, value their coworkers, or at least treat it as the ethical thing to do. This shows that, contrary to the stereotype, programmers are not predisposed to live in solitude or to be antisocial. Programmers in the startup community of Southeast Tennessee feel a responsibility to their colleagues. Future iterations of this work would center

response collection in regions with different developer climates, such as the competitive atmosphere of Silicon Valley against the enterprise community of large tech companies, to get a complete snapshot of the computing ecosystem. Ethics is a complicated subject, and the ways programmers interact with ethics is even more complicated. This survey gives a glimpse into the choices computer professionals are forced to weigh and the values they rely on to make them.

References

[1] Aapc code of ethics, 2016.

[2] Acm code of ethics and professional conduct, 2018.

[3] Bcs code of conduct for bcs, 2015.

[4] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Academic press, 2013.

[5] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[6] Batya Friedman and Helen Nissenbaum. Bias in computer systems. *ACM Transactions on Information Systems (TOIS)*, 14(3):330–347, 1996.

[7] Don Gotterbarn. Thinking professionally codes of ethics— the conscience of a profession: connecting technology and society. *ACM Inroads*, 7(4):33–35, 2016.

[8] Susan J Harrington. The effect of codes of ethics and personal denial of responsibility on computer abuse judgments and intentions. *MIS Quarterly*, pages 257–278, 1996.

[9] Charles Huff and Joel Cooper. Sex bias in educational software: The effect of designers' stereotypes on the software they design 1. *Journal of Applied Social Psychology*, 17(6):519–532, 1987.

[10] Cory Knobel and Geoffrey C Bowker. Values in design. *Communications of the ACM*, 54(7):26–28, 2011.

[11] Christoph Lutz and Aurelia Tamò. Robocode-ethicists: Privacy-friendly robots, an ethical responsibility of engineers? In *Proceedings of the ACM Web Science Conference*, pages 1–12, 2015.

[12] Charru Malhotra, Vinod Kotwal, and Surabhi Dalal. Ethical framework for machine learning. In *2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K)*, pages 1–8. IEEE, 2018.

[13] Mary L McHugh. The chi-square test of independence. *Biochemia Medica: Biochemia Medica*, 23(2):143–149, 2013.

[14] Andrew McNamara, Justin Smith, and Emerson Murphy-Hill. Does acm' s code of ethics change ethical decision making in software development? In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 729–733, 2018.

[15] Enrique F Schisterman, Neil J Perkins, Aiyi Liu, and Howard Bondell. Optimal cut-point and its corresponding youden index to discriminate individuals using pooled blood samples. *Epidemiology*, pages 73–81, 2005.

[16] Software engineering code of ethics, 1999.

My CS1 Class Flipped over COVID-19*

Bonnie Achée
Computer Science Department
Southeastern Louisiana University
Hammond, LA 70401
bonnie.achee@southeastern.edu

Abstract

As the world was turned upside down by COVID-19 and the resulting decision to immediately transition all face-to-face classes to 100% online, the decision was made to turn CS1 upside down as well, and pivot to flipped class delivery. There is no shortage of documentation on the benefits of the flipped classroom model, specifically in introductory computer science courses, however, this model was not implemented in this particular CS1 classroom prior to COVID-19 [1][5][2][3][4][6]. The COVID-19 pandemic resulting in the immediate switch from 100% face-to-face to 100% online delivery served as the catalyst necessary to facilitate the change to a flipped classroom model. This paper serves as an experience report and details the steps taken to implement the switch to a flipped classroom, the students' reactions to such and lessons learned for moving forward. The unique nature of a single collection of students experiencing both traditional and flipped delivery in a single course during a single semester provides much insight as to the viability of this approach moving forward.

1 Introduction

Southeastern Louisiana University is a mid-sized public university with approximately 14,000 students. Approximately 450 students have declared computer science or information technology as their major course of study. CS1 is a required course for all majors, minors and concentrations in the Department of

Computer Science, as well as numerous other majors in the College of Science and Technology. This paper presents the details of flipping 3 sections of CS1 in the Spring of 2020, each section having a maximum of 24 students. The course teaches the basics of programming concepts using a late objects approach, the Eclipse IDE and the Java programming language, assuming no prior knowledge of IDEs or programming. At this point, all are well aware that the effects of COVID-19 have permeated every aspect of life and none more so than the university classroom. Both students and faculty who began the semester in a traditional face to face paradigm were thrust into distance learning with mere days to facilitate the transition. This paper serves as an experience report in transitioning a traditional face-to-face CS1 class mid-semester to a flipped class model with 100% online delivery in response to the COVID-19 pandemic. The transition to online delivery was abrupt and without instructor or student input, however, the decision to transition to a flipped delivery model was not cavalier. While the COVID-19 pandemic response provided the catalyst, sufficient impetus was a result of two factors: (1) ongoing research into flipped vs. traditional delivery and (2) previous student surveys indicating increased engagement and understanding provided when class time is used for coding and problem solving rather than traditional concept lectures [1][5][2][3][4][6].

2 Implementing the Flipped Online Delivery

2.1 Pre-COVID-19 Traditional Delivery

Prior to the COVID-19 pandemic, the class was delivered face-to-face on campus to 24 students per section in a computer lab classroom in two 75-minute traditional lectures a week. No lab hours were required, however students had access to computer labs with free tutoring during the week from 9am - 5pm. The initial 25% of lecture time was spent working concept review problems from previous lectures. Approximately 75% of the lecture time was spent on concept lectures using PowerPoint slides which were made available to students prior to class. There was minimal live coding in class meetings. Students accepted this approach since it was within their comfort zone and experience of class structure, however, student surveys repeatedly reported that they found the initial portion of the class spent solving concept problems and the live coding portion the most engaging and informative. The motivation to maximize the strengths of the current classroom (i.e. concept review problems at the beginning of class, regular meeting times, and live coding exercises in class) solidified the decision to flip.

2.2 Post-COVID-19 Online Flipped Delivery

Synchronous-only, asynchronous-only and hybrid approaches to content delivery were evaluated. The following benefits of the synchronous approach were considered: continuity between pre-pandemic and post-pandemic delivery; the ability for student-instructor and student-student interaction in an environment that otherwise did not foster such; the ability for the instructor to foster relationships and provide intervention where necessary. Drawbacks to synchronous delivery considered included technology issues beyond student control limiting the ability to attend class at a specific time, locational issues that may cause distraction or prevent the student from attending. Asynchronous delivery benefits that were considered included the availability of technology at times other than scheduled class time, the ability of students to access materials multiple times [1]. The only drawback of the asynchronous delivery considered was the possibility of students to disengage in the class. Because students did not register for online delivery, the decision was made to provide a hybrid approach of both synchronous and asynchronous elements, thereby utilizing the benefits of both delivery methods, minimizing the drawbacks and setting the environment for a flipped classroom.

2.2.1 Asynchronous Elements of the Online Flipped Delivery

To achieve a "blending of direct instruction with constructivist learning" [1] an asynchronous component is necessary. This serves not only to deliver direct instruction but also to provide a place where "content is permanently archived for review or remediation." [1] To this end, a YouTube channel was created for the class with two playlists to contain all asynchronous material: a PowerPoint lecture playlist and recorded Google Meet sessions playlist. All PowerPoint slides on programming concepts which originally comprised the in-class lectures were recorded with voice-over and uploaded to a dedicated playlist on the class channel. Accessibility issues were addressed by writing a script for the voice-over content and including it in the notes section of the slides and utilizing closed caption capabilities. As per the flipped class model, the PowerPoint presentation, which students were expected to view prior to the synchronous Google Meet, would inform the concept review problems and live coding in the class [2]. The Google Meet recordings of synchronous class delivery were uploaded to a dedicated playlist on the class channel. In addition to the standard, anticipated benefit that students were able to view the lecture portion multiple times and at their convenience [1], this provided the added benefit that students were able to view the Google Meet for class sections other than their own. This afforded the students access to the in-class discussion of other sections. Accessibility compliance was addressed using the speech recog-

nition feature, "Turn on captions" , in Google Meet that generates closed captioning as shown in Figure 1.

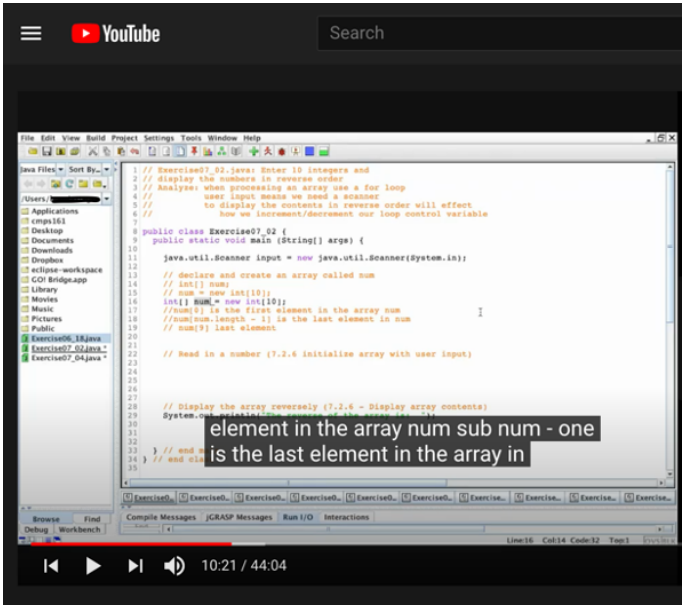


Figure 1: Google Meet Concept Problems

2.2.2 Synchronous Elements of the Online Flipped Delivery

The synchronous lecture was completely modified to fit the protocol of the flipped classroom to "increase interaction and personalized contact time between students and teachers" and to be a place where "all students are engaged in their learning." [1] This was achieved by structuring class time into two parts. At the beginning of class, students were given small problems that required the use of previously covered concepts to solve as shown in Figure 1. The instructor worked through the solutions interactively with the students, revisiting the major concepts required for solution. The students were encouraged to ask questions where they needed clarification. The remaining portion of the class was spent in "live coding" small programs as shown in Figure 2. The program requirements were introduced by the instructor and an algorithm was devised for a solution. The class worked as a group to code the algorithm to a running program. These programs served to reinforce the concepts covered in the corresponding asynchronous voice-over PowerPoint lecture that the students were

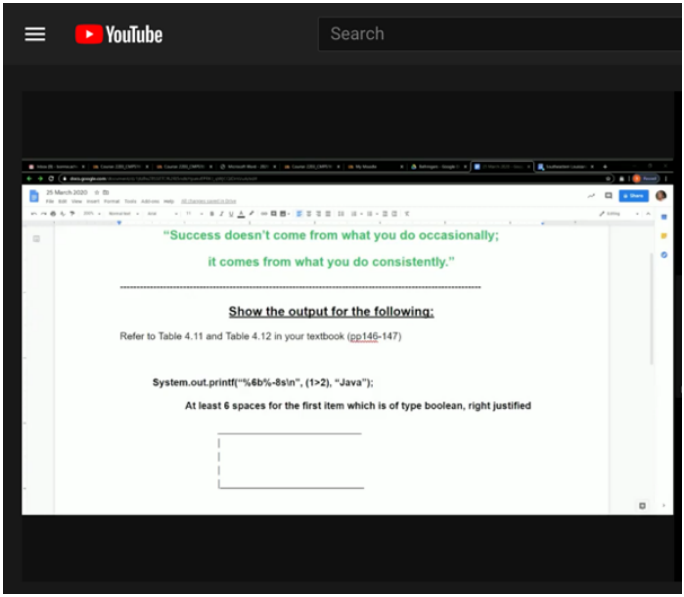


Figure 2: Google Meet live coding examples

expected to view before attending the live lecture.

3 Challenges

The transition to a flipped classroom with 100% online delivery mid-semester was not without challenges. While several of these challenges were anticipated, other challenges materialized. These challenges fall into two categories: instructional challenges and student challenges. The challenges associated with the immediate transition to online delivery were ubiquitous, ranging from technology issues to student mental health issues and everything in between. It is worth reporting that the issues surrounding the flipped delivery of the class, however, were both anticipated and minimal. It is these challenges that are the focus of this report. Instructional challenges in flipping the class were minimal and surrounded lesson planning and time constraints as predicted [4]. As previously discussed, class materials had been created for traditional delivery, therefore significant time had to be dedicated to creating both the synchronous and asynchronous content required for the flip. Synchronous content presented the least challenge as it consisted of in-class concept review problems (which were successfully incorporated prior to the flip) and coding exercises (which

were only minimally incorporated prior to the flip). Asynchronous content preparation proved more demanding as it required writing scripts and recording voice-over PowerPoint lectures. Thus the challenge was twofold: content creation and time requirement. Of the two, the time required to script, record and produce the asynchronous lectures proved far more extensive, particularly when coupled with a significantly increased amount of student and administrative emails. Student challenges to the flipped classroom model were time management and self-discipline related, causing them to not respond well to the flip initially, as predicted [6]. Students informally reported not viewing the assigned lectures prior to attending the class. It quickly became obvious which students had spent the requisite time in pre-class preparation. Because of external factors surrounding the flip, it is not possible to say how the same students would approach the pre-class preparation in a "normal" semester. With that being said, the in-class engagement was high and students reported high levels of satisfaction with the flipped class format.

4 Conclusions

Prior to COVID-19 the CS1 class at Southeastern Louisiana University was delivered face-to-face in a traditional lecture format, however, the immediate transition to 100% online delivery served as the catalyst to re-evaluate the format of the class. Based on ongoing research into the flipped classroom model and previous student surveys indicating increased engagement and understanding when such practices as concept problem solving and live coding were utilized, the decision was made to flip the class. The decision of synchronous, asynchronous and a hybrid approach to delivery was informed by the flipped classroom model and the class was delivered in the hybrid model. The traditional PowerPoint lectures moved to asynchronous delivery for pre-class viewing and valuable synchronous class time was spent solving concept problems and live coding. The transition was not without challenges, but the challenges from flipping the class were expected as they are well documented in the literature [3]. The student response was favorable and the class will proceed in the flipped model.

References

[1] J. Bergmann, J. Overmyer, and B. Willie. The flipped class: What it is and what it is not. `http://www.thedailyriff.com/articles/the-flipped-class-conversation-689.php`.

[2] M.D. Estes, R. Ingram, and J.C. Liu. A review of flipped classroom research, practice and technologies. *International HETL Review*, 4(7).

[3] M. Fetaji, B. Fetaji, and M Ebibi. Analyses of possibilities of flipped classroom in teaching computer science courses. In *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 747–752, 2019.

[4] M. Giannakos and J. Krogstie. Reviewing the flipped classroom research: Reflections for computer science education. In *Proceedings of CSERC '14*, pages 23–29. Association for Computing Machinery, 2014.

[5] EDUCAUSE Learning Initative. 7 things you should know about flipped classrooms. `http://www.educause.edu/library/resources/7-things-you-should-know-about-flipped-classrooms`.

[6] L. Ma, J. Hu, Y. Chen, X. Liu, and W. Li. Teaching reform and practice of the basic computer course based on flipped classroom. In *12th International Conference on Computer Science and Education*, ICCSE '17, pages 713 – 716, Piscataway, NJ, USA, 2017. IEEE.

A Student-based Software Development Team and Their Response to COVID-19

Bria Williams, Guillermo Cruz, Scott Heggen
Computer Science
Berea College
Berea, KY, 40403
williamsbri@bereaalumni.org, {cruzg,heggens}@berea.edu

Abstract

Often, higher educational institutions must purchase software to manage their operations. However, the cost to purchase some software is prohibitive, particularly for smaller institutions, resulting in software that can be difficult to use, poorly developed, or not fully-featured for the specific needs of the institution. An alternate solution is to hire a team of student developers, led by a faculty and minimal staff support, to create custom, institution-specific software that meets the college's exact needs, while providing students with valuable skill-building experience. In its sixth year of operation, our team has developed a framework to lead students who create software for an academic institution, resulting in nine software systems thus far. This case study details the student software development team framework, whose goal is to benefit students by emulating the software engineering industry. Lastly, the modifications made to continue pursuing this goal during the COVID-19 crisis is discussed.

1 Introduction

Academic institutions are unique in their software needs, and at times the right tools don't exist. Adapting existing software sometimes solves part of their needs, but is rarely a complete solution. Software modifications (i.e., custom features) are supported, for a cost, by software companies. Our institution took another approach, leveraging a team of students led by faculty and

staff with software engineering experience to craft custom, institution-specific software. The institution can avoid the high cost of software and maintenance, and get applications that are tailored specifically to meet their needs. More importantly, the program supports students in learning software engineering through the building of meaningful software, aligning with the educational mission of academic institutions. The students gain numerous technical and soft skills, most of which are valuable to employers [8] as previous work has shown [6], preparing them for the software engineering industry post-graduation.

There are plenty of examples of students creating real-world software (i.e., software that is used by the product owner to do their business) in courses [12], capstone experiences [2], and internships [9]. Kaminar [7] summarizes an instance where students developed software that was adopted by the institution. However, these adoptions occurred organically and were presumably one-off ventures with no long-term support. To our knowledge, no examples were found of institutions hiring students to develop custom software solutions, including maintaining the software throughout its lifetime, making this framework novel.

In March 2020, the program was challenged by the COVID-19 pandemic. Our institution made the decision to close its campus within one week of the WHO declaration [15], one of the earliest campuses to do so. As many academic institutions moved to remote-only classes and work, their dependency on software and supporting services became more significant. The change resulted in many adjustments to our student-led software development framework to continue maintaining software while students worked remotely.

This paper begins by describing one of the nine systems developed by the Student Software Development Team (SSDT). Next, the framework for the program is described, which blends the best parts of a software engineering course, a capstone experience, and an external experience like an internship. The paper will conclude with a discussion about how the program shifted to remote-only work during the COVID-19 crisis, demonstrating the adaptability of the program and how the students are benefitting from the framework being applied to a remote-only working environment.

2 Software Developed by Students

The challenge and experience of building entire software applications for the institution gives students the opportunity to grow outside of a traditional classroom setting, while benefitting the institution with custom applications to solve their unique needs. The software projects the SSDT undertake are uniquely specific to the institution. The team receives requests for help from departments across the institution. A number of factors determine which systems

will be implemented, including the request' s urgency, value to the institution, and the expected effort required to complete the project. The vast majority of the applications the SSDT build are web applications, using familiar web frameworks that are consistent across all of our applications.

In the six years since the creation of the SSDT, the team has undertaken twelve software systems, nine of which are still actively maintained by the team. Systems have been built to support the specific needs of one department, such as a tool for managing the entire Art program's archives, while others serve the entire institution, such as the labor status forms system described next.

2.1 The Labor Status Forms Application

There is not a large market for the creation of software to support the specific needs of work colleges, considering there are only nine in the United States [14, 4]. As a work college, our institution of approximately 1,600 students hires every student into a work-study program as part of their requirements for graduation. Each student works 10-20 hours a week, and each department at the college is allocated a specific number of students. The office in charge of regulating student labor was using paper forms to manage each student, resulting in significant errors. This process not only required additional student labor, it also expected students to walk across campus gathering signatures from multiple supervisors and academic advisors to complete their forms. A lean project estimated approximately 114 minutes per student to complete a hire.

The SSDT was asked to develop a custom application that would replace the antiquated process. A new web application was designed and developed to be used by every labor supervisor to hire and fire students. It was built with an administration interface, meaning all labor to be maintained within one web application, streamlining the process. The new web application, with its first version completed in 2015, provided the labor administrators with a much needed improvement to their previous process. The aforementioned lean project estimated a hire could be completed in 4 minutes, with significantly less errors.

After five years of developing this one system, a large backlog of requests had been received and were not getting integrated. This was largely due to the language selected when we began, as well as five years of code that had drastically improved over time. However, old code still lingered that occasionally didn't fit our new standards for quality. In 2019, the team decided to move the web application to a new framework that mirrored the other eight applications also managed by the SSDT, reducing the amount of relearning necessary when switching between software systems, as well as eliminating the many mistakes made by previous programmers that still lingered. The new system

also opened the door for new potential, as it introduced new processes. For example, sometimes a student needs proof of employment. Since paper forms no longer existed, a new proof of employment document had to be generated from the system.

This application began as an idea to eliminate the error prone paper-based process, and eventually evolved into an application that is now handling all aspects of student labor. The SSDT is driven to build software solutions for departments across the institution, with an emphasis on building maintainable software. This is why the SSDT has structured their software engineering process in a way that gives the team the necessary time needed to build applications from the ground up, while still being able to maintain all the systems.

3 Our Existing Software Engineering Framework

Over these last six years, an evolving process was created that informs the current framework presented here. Following best practices in lean thinking, the current framework is regularly evaluated and modified to meet the team's current needs. Thus, the framework presented in this section represents the current state and has been generalized as much as possible, but will certainly continue to evolve as the needs of the team change.

The SSDT follows Agile principles [5] using a modified version of Scrum [10]. The Scrum methodology identifies three key roles in a software development team: product owner, Scrum Master, and the development team. The product owner is the customer who made the software request and determines the vision and purpose of the software product. All work must be approved by the product owner before it is considered complete. The Scrum Master is typically a full-time employee of the college, managing the development team and the backlog of work, ensuring focus is maintained on the most important features of the application. Finally, the development team in this framework consists of undergraduate students who are "structured and empowered by the organization to organize and manage their own work" [10]. Scrum theory provides an iterative, yet incremental approach to cut down on risks and enhance the predictability of the project for the product owner.

Scrum also prescribes four events in the software engineering process: sprint planning, the sprint, sprint review, and sprint retrospective. These four events are implemented differently in the framework depending on the term, because the summer term has students working four times more than the regular term (i.e., 40 hours per week versus 10 hours per week). In the summer, the team follows a traditional Scrum model with a product backlog and work in progress determined during spring planning, daily Scrum meetings, and a Sprint Review and Retrospective happening every two to three weeks before starting the next

cycle. In a typical regular term, the Scrum timeframe is stretched out; Scrum meetings happen weekly, and sprints take four to six weeks. We often save new systems and major features for summer terms, while smaller features and bug fixes happen during regular terms. The next section details the framework during these two phases: the Summer Internship Phase, and the Maintenance and Customer Support Phase (i.e., the regular term).

3.1 The Summer Internship Phase

Starting in the summer term, students are hired into the SSDT based on a few metrics, including grades in core classes, their ability to work well with others (namely, pair programming [13]), and the perceived value they will attain from participating in the program (i.e., the strongest students are not always selected, as they may not benefit as much from the program).

A typical summer operates much like an internship, where students are employed for 40 hours per week for 8-10 weeks, resulting in up to 400 hours of software engineering experience. Their expectations mirror an internship in many ways: they are expected to be punctual, productive throughout the day, and regularly reporting their progress to supervisors. Each summer consists of six to ten students managed by one faculty and two staff (the leadership team). The students work in pairs as they design interfaces and develop code. Very rarely will a student have been explicitly trained on software engineering principles such as Agile methodologies, Model-View-Controller (MVC) or similar frameworks, or large-scale application development, prior to joining the team.

A key skill gained by our students very early in the summer internship is the ability to translate customers' needs into usable software (i.e., requirements gathering and design). Students begin by paper prototyping [11] a design of the application. This design process consists of many iterations of interfaces drawn on paper. Designs are critiqued and modified until there is a group consensus to move forward. This avoids the students spending significant hours writing code that doesn' t match the team' s understanding of the application or the product owner' s needs. After paper prototyping, the students begin to construct the models that support their prototype. As a group, they propose the underlying data structures of the application and the relationships between parts of the model. This process provides students with a better understanding of the model when they begin implementing their prototype, reducing confusion about how data is stored and retrieved. Having the front-end design and data model in place, the students begin building the application. Students are first given a virtual machine to develop on, which mirrors the production environment, but also requiring them to learn some basic Linux commands and usage (again, a new skill for most). Working from a template, pairs of students begin construction from a common git repository, also learning about

git workflow.

Throughout the summer, everyone meets for a daily Scrum. Students are encouraged to ask lots of questions early in the process. In fact, most of the training happens as a result of a question being asked, since no formal technical training is done. As the summer progresses, the leadership team begins expecting the students to ask each other for help first. This promotes sharing of knowledge among the students, as pairs begin to solve similar problems that others have already seen, and can help each other from making the same mistakes. To maintain organization and visibility about what features each pair is working on, we leverage Kanban boards [1]. Applying the Kanban model with the Scrum framework manages the overall flow of the project well.

As interfaces near completion, usability tests [3] are performed to test user interactions. The usability tests are run with another team, then the leadership team, and lastly with the product owner for final approval of the implementation. Then, the students' code gets tested for security, coding standards, accessibility guidelines, and bugs before being integrated into the production environment by the leadership team. Ideally, by the end of summer, the completed product is delivered to the customer. Because this is not always the case, constant communication with the product owner and clear expectation setting reduces surprises about delays in deployment. Any features that aren't completed migrate to the regular term, after critical bugs are resolved.

3.2 The Maintenance and Customer Support Phase

In the fall term, the team shifts to 10 hours per week, as the students begin attending classes. As expected, the productivity of the students reduces dramatically in the regular terms. The framework takes this into consideration. The team shifts its focus to maintaining existing software and responding to customer needs (e.g., bug fixes or small feature requests) before attempting to build new interfaces. Students gain valuable experience maintaining their software after deployment, a skill rarely taught in software engineering courses after projects are "completed" and delivered to customers.

Similar to summer, the Kanban board is instrumental to keeping the team organized and aware of each others' work, particularly since the students' schedules become more spread out. Students check the Kanban board for new issues, claim them, and keep record of their progress. This flow helps developers visualize their progress and estimate their time, which is essential given they only have 10 hours per week to work. When the issue is resolved and tested, a pull request is created and reviewed by the leadership team.

4 Shifting Process in Response to COVID-19

Following the COVID-19 campus closure, the SSDT program was put on hold midway through the Spring 2020 term. Only the leadership team continued to work on the software systems in our last sprint. The shutdown itself required a few changes to existing systems. For example, updated syllabi were needed for every course, outlining how classes would change during emergency online teaching. One of our software systems was currently being used to collect syllabi during regular terms; our team quickly modified the system to support an new "COVID-19" term, where new syllabi could be uploaded quickly.

The SSDT team worked with the administration to permit students to work on software in a remote-only capacity. The team quickly adapted its framework to this new work environment. One of the many considerations that had to be acknowledged during this process is that some students were now dealing with outside distractions that did not exist when working on-campus, including moving, balancing parenting and other duties, and unreliable technology at home.

The leadership team began by prioritizing issues in the queue. A thorough walkthrough of the current application was done to identify new issues created by unfinished work as students left campus. The final and most important step was deciding which tools the team would use that would most closely resemble the workflow while on-campus. Ultimately, the leadership team identified four tools that were key to facilitating student success in a remote SSDT environment: 1) a digital version of the Kanban board for organizing tasks; 2) a communication platform to facilitate online meetings; 3) a wireframing tool for creating low-fidelity prototypes, replacing our paper prototyping process; and 4) a collaborative code editor for modeling pair programming. The next section details how these tools were implemented.

4.1 New Tools and Processes

Consistent communication between the students and the leadership team was of the utmost importance in ensuring progress. An online Kanban board was used to digitize the physical board in our workspace. Figure 1 shows the old (left) and new (right) versions of the Kanban boards, indicating similar objects on each. The old version of the board required students to write their current issue on a "card" and place that card on "the wall." Cards moved up the wall as they got closer to completion, starting in the backlog and ending with a pull request, with percentage complete in between. This same organization was used in the digital version of the Kanban wall with the only major difference being that cards moved from left to right as they neared completion.

The team already utilized a platform for much of its online communication,

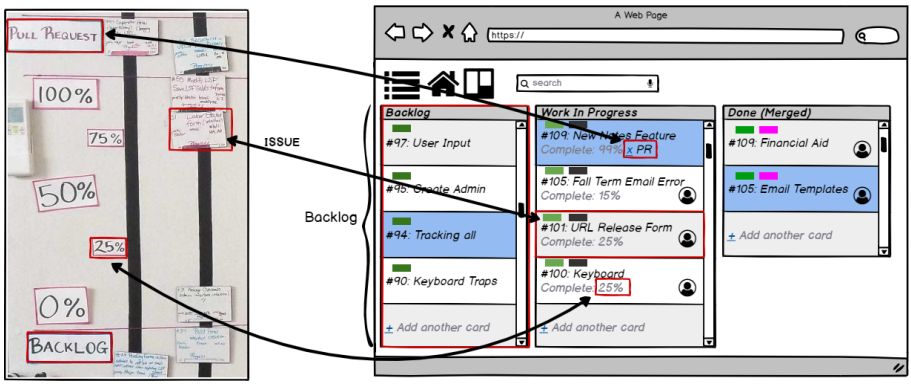


Figure 1: The physical Kanban board (left) and its virtual equivalent (right)

so it was decided to continue with a more significant reliance on this tool. The student programmers followed a "Did, Doing, Stuck" (DDS) structure, where the students would explain what they did the previous day, what they will be doing this work day, and if they are stuck on anything. The leadership team would check these updates daily to monitor progress and ensure students are not stuck for an extended amount of time, closely mirroring our process before the crisis. The communication platform also offered video calling capabilities, allowing the team to continue doing Scrum meetings as they did in the previous summers, just through video conferencing. Screensharing and virtual drawing tools were leveraged to simulate whiteboards, provide "just-in-time" training to newer students, and share interfaces for feedback.

Given the situation, students were not able to collaborate in-person on paper prototypes. Instead, an online collaborative wireframing tool was used to create mockups. The wireframing tool proved nearly as effective as paper prototyping, but also allowed designs to be responsive and perform like actual webpages (e.g., demonstrating "on click" events). This ensured that the team did not stray from our design standards, and introduced a tool that provides an efficient way to prototype interfaces that will likely be adopted post-pandemic.

Pair programming was particularly important to the leadership team, for many reasons [13]. Our solution was for the student programmers to video call during the entire work session, allowing them to discuss their work, similar to before the pandemic. However, pair programming from different terminals is highly ineffective. So, a plugin was added to their IDEs that allowed pairs to share code and see edits in real-time. The leadership team also used this plugin to perform code reviews and demonstrations with the student teams.

Anecdotal evidence suggests that the students are feeling productive, ef-

ficient, and are still learning from the experience despite the circumstances. The leadership team regularly asks the students in the morning scrum meetings to provide feedback about how to improve this process for them. They are mostly indicating they simply need practice and time to develop the skills needed, which is largely in line with what is expected of them when in person.

5 Conclusion

Higher educational institutions are often limited by proprietary software that can't meet their exact needs at a reasonable cost. The Student Software Developers Team is a framework for developing software, where the development team is composed primarily of undergraduate students who are not trained software engineers. Detractors are quick to indicate that trusting undergraduates with this task, even under the leadership of faculty or staff, is an unacceptable risk to the institution. However, the Student Software Development Team has developed nine software systems which are used broadly by the campus community in the last six years, proving its viability. The framework presented here represents one way in which this can be achieved.

The importance of remote work has been well known in the software engineering industry for years, but rarely is any effort spent in training students to work remotely. In the span of one month, we were able to move the SSDT to a remote-only process. The Summer 2020 cohort consisted of twelve students, seven of which had never participated in the program before. They were able to participate without any noticeable negative impact on efficiency and code quality. In fact, some new tools were added which we plan to adopt as part of the team's process as a result of the pandemic. The cohort was able to complete nearly all of the planned upgrades to the Labor Status Forms application described in Section 2.1, which is going into the production environment by the time of this writing. In fact, the Summer 2020 cohort was finished early enough that they were able to begin the planning phase for the next new application. Finally, the students are adding an additional important skill to their blossoming resumes: remote collaboration in a software engineering team.

References

[1] David J Anderson. *Kanban: successful evolutionary change for your technology business*. Blue Hole Press, 2010.

[2] AT Chamillard and Kim A Braun. The software engineering capstone: structure and tradeoffs. In *ACM SIGCSE Bulletin*, volume 34, pages 227–231. ACM, 2002.

[3] Joseph S Dumas, Joseph S Dumas, and Janice Redish. *A practical guide to usability testing*. Intellect books, 1999.

[4] Ecclesia College. Our Mission. https://ecollege.edu/about-us/., Accessed May 2020.

[5] Martin Fowler, Jim Highsmith, et al. The Agile manifesto. *Software Development*, 9(8):28–35, 2001.

[6] Scott Heggen and Cody Myers. Hiring millennial students as software engineers: a study in developing self-confidence and marketable skills. In *Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials*, pages 32–39, 2018.

[7] Ariel Kaminer. Student-built apps teach colleges a thing or two. *New York Times*, Aug 2014.

[8] Ilana Lavy and Aharon Yadin. Soft skills-an important key for employability in the "shift to a service driven economy" era. *International Journal of e-Education, e-Business, e-Management and e-Learning*, 3(5):416, 2013.

[9] Michael J Lutz, J Fernando Naveda, and James R Vallino. Undergraduate software engineering. *Commun. ACM*, 57(8):52–58, 2014.

[10] Ken Schwaber and Jeff Sutherland. The scrum guide. *Scrum Alliance*, 21:19, 2011.

[11] Carolyn Snyder. *Paper prototyping: The fast and easy way to design and refine user interfaces*. Morgan Kaufmann, 2003.

[12] Nasser Tadayon. Software engineering based on the team software process with a real world project. *Journal of Computing Sciences in Colleges*, 19(4):133–142, 2004.

[13] Laurie Williams, Eric Wiebe, Kai Yang, Miriam Ferzli, and Carol Miller. In support of pair programming in the introductory computer science course. *Computer Science Education*, 12(3):197–212, 2002.

[14] Work Colleges Consortium. Member colleges. https://www.workcolleges.org/member-colleges, Accessed May 2020.

[15] World Health Organization. WHO director-general's opening remarks at the media briefing on COVID-19. https://www.who.int/dg/speeches/detail/who-director-general-s-opening-remarks-at-the-media-briefing-on-covid-19—11-march-2020.

Improving Online Lectures in Distance Learning CS0 Course*

Conference Tutorial

Karen E. Works
Computer Science
Florida State University
Panama City, FL 32405
keworks@fsu.edu

CS0, the gateway course to most computer science degrees, is challenging and can be an obstacle to retaining students in the computer science majors [3]. Due to the COVID-19 crisis, many colleges are having to rapidly create online lecture materials for students aka "'emergency remote learning"' [4]. Clearly, given the nature of CS0 quality online materials are essential. Simply taping a face to face lecture has been shown to not effectively reach students [1].

In this tutorial, we explore different methods to create online videos that effectively support student's success in CS0. We will look at approaches of incorporating activities into lecture videos that promote active learning [5, 2, 6]. Participants will be given an opportunity to share both challenges and success stories of online materials for CS0. A variety of resources to support the creation of online video lectures will be presented. Specific examples included will be focused on content to support CS0.

This tutorial session will provide instructors with a broad range of ideas and approaches in how to create more effective online video lectures. Instructors will be given time to work in groups on creating short lecture videos using the approaches covered.

Biography

Dr. Works joined Florida State University Panama City in the summer of 2019. At FSU, she teaches courses for the distant learning computer science degree program. Prior to coming to FSU, she had 6 years university teaching experience in traditional face to face classroom settings.

In 2016, Dr. Works flipped her Intro to Programming course and won the Massachusetts Colleges Online Course of Distinction award. Flipping a course is where all lecture materials are provided online, and class time is devoted to working on hands on labs and projects. She is passionate about teaching computer science and always looking for innovate methods to best support her students.

References

[1] Terry Anderson. *The theory and practice of online learning.* Athabasca University Press, 2008.

[2] Jamie Costley, Mik Fanguy, Chris Lange, and Matthew Baldwin. The effects of video lecture viewing strategies on cognitive load. *Journal of Computing in Higher Education*, pages 1–20, 2020.

[3] Michael Haungs, Christopher Clark, John Clements, and David Janzen. Improving first-year success and retention through interest-based cs0 courses. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 589–594, 2012.

[4] Charles Hodges, Stephanie Moore, Barb Lockee, Torrey Trust, and Aaron Bond. The difference between emergency remote teaching and online learning. *Educause Review*, 27, 2020.

[5] Christopher Lange and Jamie Costley. Improving online video lectures: learning challenges created by media. *International Journal of Educational Technology in Higher Education*, 17:1–18, 2020.

[6] Anna Wood, Kate Symons, Jean-Benoit Falisse, Hazel Gray, and Albert Mkony. Lecture capture in online learning: A community of inquiry perspective. 2020.

Functional Take-Away*

Nifty Assignment

Steven Benzel
sbenzel@acm.org

Introduction

Combinatorial game theory [1] affords a rich collection of interesting problems for students to sharpen their coding skills. Impartial combinatorial games with complete information live on acyclic directed graphs and an optimal strategy for a player can be given using Grundy theory [2][3]. CGT was discussed in an introductory functional programming course and students were challenged to develop a game player procedure which takes a state of a game as input and perform a move by outputting the resulting new state. Developing a game player to efficiently implement a winning strategy requires students to implement a variety of abstract concepts including: subset generation, recursion over states, memoization, and nim addition.

The game of take-away is defined as follows: given a subset S of natural numbers (for example $S = \{1, 2, 4\}$) and a pile of tokens, two players alternate taking n tokens from the pile where n is an element of S. Play continues until the pile is empty and the last player that takes a token wins. S is known as the take-away set for the game. Multi-take-away is the extension of take-away to a multiple of piles where a player can only take n tokens from a single pile, where again n is an element of S. The last player to take a token (necessarily from the last remaining pile) wins.

Assignment

Students were assigned the task of providing a Scheme procedure with input S, the take-away set given as a List, and output a game player to play multi-take-away. The game player must itself be a Scheme procedure which given input a game state (the sizes of the piles given as a List) outputs the new state, the List of new pile sizes. The new state must derive from the old state

by a valid take-away move. An example of running and testing the procedure generate-player is as follows:

```
> (define S  '(1 2 4))
> (define player (generate-player S))
> (define old-state  '(3 4 5))
> (define new-state (player old-state))
> new-state
(3 4 1)
> (is-valid-move? old-state new-state S)
#t
```

For multi-take-away the grundy value of a state can be efficiently computed by first computing the grundy value for each pile of the state and then combining these values using the nim sum [3]. Thus, it is possible to produce an efficient player with perfect strategy. Although Grundy theory was discussed in detail, students were not required to produce a such a player and were only graded on correctness of their player and on their percentage of wins against the make-random-move player. For fun, a competition was run where student' s players were matched head-to-head for 100 random games and winning percentages were reported.

References

1. Winning Ways For Your Mathematical Plays, E. Berlekamp, J, H, Conway, R. Guy (1982) Academic Press
2. Mathematics and games, P. M. Grundy, Eureka, 2: 6-8 1939
3. `https://www.math.ucla.edu/~tom/Game_theory/Contents.html`

Goertzel: Lightweight DFT for Telephony, Morse Code and More*

Nifty Assignment

Robert Lutz and Evelyn Brannock
Georgia Gwinnett College, Lawrenceville, GA 30043
{rlutz,ebrannoc}@ggc.edu

About

If you only need to detect a few frequencies, a much faster (and computationally superior) method than the FFT, called the Goertzel algorithm [3], is available. This is a lightweight DFT computation that has the positives of superior simplicity, speed, and lower CPU requirements and therefore has many uses. One is the recognition of which key on a dial pad has been chosen –only a very few well-known well-described frequency ranges need to be detected. [5]

Students convert a prewritten Java Swing application that contains the implemented Goertzel algorithm to a JavaFX application. The Goertzel class can be used to detect if one or more predefined frequencies are present in a signal. If these frequencies are present, this identifies which key on the dialpad has been selected and presents this information via a GUI.

Can you reuse code by finding other creative applications of the Goertzel class?

Process

Students will begin with the Chitova263's Swing-based app [1] in Figure 1 and modify it to produce a JavaFX-based app shown in the same figure.

In both the original and derived app, processing is performed in a multi-step workflow. First, upon a keypress event, a waveform is generated for each of the dual frequencies, according to the DTMF standard [4]. Next, these signals are added into a composite waveform. Then, the combined signal is split into batches of samples. For each batch, an energy level is reported for each frequency of interest. After thresholding, tone detection is performed by comparing a detected frequency pair to a lookup table for the DTMF standard

Figure 1: Goertzel Assignment Flow

[4]. When two frequencies match the DTMF specification, the detected key is reflected in the UI. Algorithm details are provided in Goertzel's original paper [3], Kevin Banks' EETimes discussion [5] and wikipedia.org [2].

Metadata

Summary	• Setup project in IntelliJ w/JavaFX. (This is clumsy, worth documenting) • Digest and understand functionality of current, supplied code • Determine reusable code, and code that needs to be ported (Swing → FX) • Discover other uses for Goertzel's Algorithm
Topics	Programming, Computational Complexity, API Reading, Lit Review, GUI
Audience	Late CS2, Mobile App Development, Algorithms, Embedded Systems
Difficulty	Medium, due to mathematical underpinnings of Goertzel
Strengths	• **Global in nature**: Supports a variety of languages. • **Introduces the value of integration to other applications and tools:** Encourages learning another API (Swing to FX) • **Deep algorithmic and coding knowledge is not required**: Students are not required to understand DFT/ FFT • **Adaptability to multiple audiences:** Starter code can be provided to scaffold content areas of other courses • **Performance oriented:** Encourages discussion of computational complexity, embedded systems • **MVC:** Enforces importance of assignment of single responsibility to a class
Weaknesses	• DFTs / FFTs cognitive overhead
Dependencies	• IDE of choice (IntelliJ, Eclipse) and version of Java
Variants	• Adapt for Morse code receiver/ Morse code keyboard in Google Gboard • Adapt to Java Streams • Implement in other language (Python, Kotlin, Go, Dart/Flutter, Matlab) • Experiment with tri-tone multi-frequency, quad-tone multi-frequency • Build graphic visualizer • Use in networking classes to define and test custom protocols • UI variants: DTMF workbench, Morse Code workbench, etc. • Read/write audio data in binary formats: low-level or with libraries for mp3, wav, ogg etc. • Read onboard sensor data (microphone) in real-time and process

References

[1] Dual Tone Multi-Frequency Generator. https://github.com/Chitova263/Dual-Tone-Multi-Frequency-Generator.

[2] Goertzel Algorithm. https://en.wikipedia.org/wiki/Goertzel_algorithm.

[3] Goertzel paper (pp. 34-35). http://www.jstor.com/stable/2310304.

[4] Specification of Dual Tone Multi-Frequency (DTMF). https://www.anacom.pt/streaming/es_20123501v010101p.pdf?categoryId=115679&contentId=201604&field=ATTACHED_FILE.

[5] The Goertzel Algorithm. https://courses.cs.washington.edu/courses/cse466/12au/calendar/Goertzel-EETimes.pdf.