The Journal of Computing Sciences in Colleges

Papers of the 28th Annual CCSC Midwestern Conference

October 1-2, 2021
Ivy Tech Community College
Fort Wayne, IN

Baochuan Lu, Editor
Southwest Baptist University

Saleh Alnaeli, Regional Editor
University of Wisconsin-Stout

Volume 37, Number 4 **October 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 **7**

CCSC National Partners **9**

Welcome to the 2021 CCSC Midwestern Conference **10**

Regional Committees — 2021 CCSC Midwestern Region **11**

Reviewers — 2021 CCSC Midwestern Conference **13**

Conducting Survey Research in a Computing Topics Course: Phubbing and Being Phubbed **14**
Robert E. Beasley, Mohanid M. Akermawi, David G. Barnette, Nathanael D. Beasley, Franklin College

Automating Configuring Parallel Compute Environments for Students **25**
Bryan Dixon, California State University - Chico

Godot Engine and Checklist-Based Specifications: Revising a Game Programming Class for Asynchronous Online Teaching **30**
Paul Gestwicki, Ball State University

Visualizing Recursion Using Code and Stack Animation **41**
Y. Daniel Liang, Jireh Bethely, Gursimran Singh Walia, Georgia Southern University

A Machine Learning Approach to Understanding the Viability of Private 4-Year Higher-Education Institutions **50**
Kian L. Pokorny, McKendree University

Python Data Structures for Java Programmers – Conference Tutorial **58**
Bill Nicholson , Isaiah Dicrostoforo, University of Cincinnati

A Tutorial on Flutter – Conference Tutorial **59**
Michael P. Rogers, University of Wisconsin Oshkosh

Using GitHub Classroom for Assignment Management and Automated Feedback – Conference Tutorial **60**
Zachary Kurmas, Grand Valley State University

The Philosophies of CS 1 – Panel Discussion **62**
Cathy Bareiss, Bethel University, Jessen Havill, Denison University, Steve Bogaerts, DePauw University, Osvaldo Jimenez, University of the Pacific, John Trono, Saint Michael's College

IndianaComputes! Views of a K-12 Professional Development Program – Panel Discussion **62**
Karen M. Morris, University of Notre Dame

Teaching Heterogeneous Parallel Programming With CUDA – Conference Workshop **64**
David P. Bunde, Knox College

An Introduction to Tableau as a Data Visualization Tool – Conference Workshop **66**
Mary Jo Geise, University of Findlay

Building Regional Community for Computing Education Graduate Students – Conference Workshop **67**
Morgan M. Fong, Max Fowler, Seth Poulsen, Vidushi Ojha, Geoffrey L. Herman, University of Illinois Urbana-Champaign

Aspects of US-China Competition May Motivate Students – Work In Progress **69**
Pradip Peter Dey, Bhaskar Raj Sinha, National University

Developing a Cross-Platform Mobile Course Using a Multi-Paradigm Library – Work In Progress **71**
Alisa Neeman, Muskingum University

IndianaComputes! a K-12 Professional Development Program – Work In Progress **72**
Karen M. Morris, University of Notre Dame

Can OneUp Gamified Challenges Boost Undergrad Student Motivation Plus Engagement and Supplement Learning in An Online Introductory Cybersecurity Course? – Work In Progress **73**
Ankur Chattopadhyay, Meghyn Winslow, Momoka Kinder, Northern Kentucky University

Practical Program Verification with DAFNY – Work In Progress **75**
Ramachandra B. Abhyankar, Robert W. Sternfeld, Indiana State University

Spear phishing attack using Kali Linux – Work In Progress **77**
Imad Al Saeed, Saint Xavier University

Faculty-Advisor Relationship Impact on Student Pathways to IT Careers/Education – Work In Progress **79**
Matthew Cloud, Ivy Tech Community College of Indiana

The Consortium for Computing Sciences in Colleges Board of Directors

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 Midwestern Conference

Welcome to the 28th annual CCSC Midwestern Conference held at Ivy Tech Community College on October 1-2, 2021. With this year's hybrid conference format, we are looking forward to seeing everyone either in person in Fort Wayne, IN or virtually from anywhere.

The conference program includes paper sessions, speakers, pre- and post-conference workshops, tutorials, student activities and vendor sessions. We had a smaller number of paper submissions than usual, which is not surprising given the tumultuous year that we have all had. Nevertheless, the papers that were submitted were of high quality, and six out of eight submissions were accepted for a 75% acceptance rate.

We are honored to have Mr. James Weaver, Quantum Developer Advocate at IBM, as our keynote speaker. Mr. Weaver is a prominent developer, as well as an accomplished author and speaker. We are also very pleased to welcome Dr. Joe Czyzyk, Advanced Analytics Math Modeler at The Dow Chemical Company, as our banquet speaker. Dr. Czyzyk received his Ph.D. in Industrial Engineering / Management Science from Northwestern University and has worked at Argonne National Labs and Central Michigan University Research Corporation prior to joining The Dow Chemical Company. His talk will be on "How Data Science is Adding Value at The Dow Chemical Company".

We have three workshops planned: two pre-conference and one post-conference. The pre-conference workshops are titled "Building Regional Community for Computing Education Graduate Students" and "An Introduction to Tableau as a Data Visualization Tool". The post-conference workshop is "Teaching heterogeneous parallel programming with CUDA". Three exciting tutorial sessions are also planned, as well as multiple activities focused on students. These activities include the programming competition, the Student Showcase, and a popular panel session titled "What Students Need to Know About Industry".

Google, zyBooks and Codio are each hosting sessions. We sincerely thank them for their support of CCSC as National Partners. I encourage you to attend their sessions and speak with their company representatives.

In closing, I would like to express my sincere thanks to the Conference Committee and Paper Reviewers, whose efforts make this conference an outstanding forum for sharing ideas. We are always happy to welcome new volunteers, so please let us know if you are interested in joining us. Thank you for being part of the 28th annual meeting of CCSC Midwest.

Grace Mirsky
Benedictine University
Conference Chair

2021 CCSC Midwestern Conference Steering Committee

Regional Board — 2021 CCSC Midwestern Region

Reviewers — 2021 CCSC Midwestern Conference

Robert Adams	Grand Valley State University, Allendale, MI
Alisa Neeman	Muskingum University, New Concord, OH
Brian Howard	DePauw University, Greencastle, IN
Dave Surma	Indiana University South Bend, South Bend, IN
David Bunde	Knox College, Galesburg, IL
David Largent	Ball State University, Muncie, IN
Deborah Hwang	University of Evansville, Evansville, IN
Henry Walker	Grinnell College, Evansville, IN
Eugene Wallingford	University of Northern Iowa, Cedar Falls, IA
Henry Walker	Grinnell College, Grinnell, IA
James Teresco	Siena College, Loudonville, NY
James Vanderhyde	Saint Xavier University, Chicago, IL
Jeffrey Lehman	Huntington University, Huntington, IN
Kristofer Schlieper	Minnesota State University Moorhead, Moorhead, MN
Lawrence D'Antonio	Ramapo College, Mahwah, NJ
Michael Glass	Valparaiso University, Valparaiso, IN
Paul Gestwicki	Ball State University, Muncie, IN
Robert Adams	Grand Valley State University, Allendale, MI
Zachary Kurmas	Grand Valley State University, Allendale, MI

Conducting Survey Research in a Computing Topics Course: Phubbing and Being Phubbed[*]

Robert E. Beasley, Mohanid M. Akermawi,
David G. Barnette, Nathanael D. Beasley
Department of Mathematics & Computing
Franklin College
Franklin, IN 46131

{rbeasley,mohanid.akermawi}@franklincollege.edu

{david.barnette,nathanael.beasley}@franklincollege.edu

Abstract

This article describes the experience of using a Computing Topics course to teach students how to conduct survey research in the field of Computing. In the course, teams of three or four students were required to think of something they would like to investigate about computing technology using the scientific method. They were then required to articulate their research methodology, conduct their research, present the results of their research, and discuss the results of their research. It also reports on the findings of one team's research in the area of phubbing (i.e., the practice of ignoring one's companion or companions in order to pay attention to one's phone). The results of the study suggest that college students 1) often phub romantic partners, but they "kind of dislike" being phubbed by romantic partners, 2) frequently phub family members (other than romantic partners), and they "don't mind" being phubbed by family members, 3) rarely phub professional superiors, and they "pretty much dislike" being phubbed by professional superiors, and 4) sometimes phub professional peers, but they "kind of dislike" being phubbed by professional peers. Implications are discussed.

[*]Copyright ©2021 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

Introduction

Topics courses permit faculty members to teach subjects that are of current interest. Such courses are usually offered at the request of a faculty member and require institutional approval. For the past several years, the primary author of this article (i.e., the professor) has offered a topics course titled *Cognition in Computing*. A significant portion of this course requires teams of three or four students to think of something they would like to investigate about computing technology using the scientific method. They were then required to articulate their research methodology, conduct their research, present the results of their research, and discuss the results of their research. The only stipulation for the research projects was that they focus on phone use and that they utilize the survey research methodology. These were chosen so that all of the teams in the course were focused in the same general area and so that data could be easily gathered from a significant number of other students on the small college campus. The teams were evaluated by the professor and by peer review.

Before the research projects began, class time was dedicated to describing the Scientific Method. Although all of the students in the class had taken a course in basic applied statistics, additional class time was dedicated to reviewing basic applied statistical methods and the use of statistical software. In addition, class time was dedicated to reading and discussing various examples of survey research in the field of Computing. The students were tested over these topics via a traditional examination.

During the course, the professor worked closely with the teams to carefully state their hypotheses, develop their survey instruments, define their variables, describe their procedures, and select their statistical methods. This part of the course was a collaborative effort with the professor acting in a mentoring or coaching role with the teams to ensure that the research design was of high quality.

Toward the end of the course, all four teams, each of which had a different research topic, presented their research to the class for evaluation. Although all of the teams did a very good job, the results of their work were not quite in a publishable state due to minor data analysis issues, incomplete conclusions, and so on.

When the course was complete, the professor contacted the students in one of the teams whose data seemed particularly interesting. These students, who had just graduated, were asked if they'd like to continue with the research direction. They all agreed. The professor took the lead on the project, performing some additional data analyses, communicating the results of the analyses with the team via email, and meeting with the team in person to discuss the results of the analyses and their implications.

The topic of the study was *phubbing*. The *Oxford Living Dictionary* defines phubbing as the practice of ignoring one's companion or companions in order to pay attention to one's phone [1]. Behaviors associated with phubbing include leaving one's phone out in plain sight, letting one's phone alert them to calls and notifications, checking one's phone for calls and notifications, answering calls and notifications without explanation, and using one's phone when there is a break in the conversation [7].

In this study, the team set out to examine the frequency with which college students phub others and to determine if this frequency is in harmony with their own feelings about being phubbed. More specifically, the team attempted to determine whether or not phubbing frequency in college students is consistent with their affective (i.e., emotional) response to being phubbed in romantic relationships, familial relationships, professional relationships with superiors, and professional relationships with peers.

Research Methodology

Hypotheses

The four general null hypotheses were:

- $H1_0$: Phubbing frequency in college students is consistent with their affective response to being phubbed in romantic relationships.
- $H2_0$: Phubbing frequency in college students is consistent with their affective response to being phubbed in familial relationships.
- $H3_0$: Phubbing frequency in college students is consistent with their affective response to being phubbed in professional relationships with superiors.
- $H4_0$: Phubbing frequency in college students is consistent with their affective response to being phubbed in professional relationships with peers.

Instruments

The first instrument was used to elicit the frequency with which college students phub others. The questions used to elicit this frequency were: (Q1) I have my phone out in plain sight, (Q2) I let my phone alert me to calls/notifications, (Q3) I check my phone for calls/notifications, (Q4) I answer calls/notifications without explanation, and (Q5) I use my phone when there is a break in the conversation. The 5-point Likert-type responses to these questions were: never (0 points), rarely (1 point), sometimes (2 points), usually (3 points), always (4 points), and no answer.

The second instrument was used to elicit the affective responses of college students to being phubbed by others. The questions used to elicit these responses were (Q1) I like when they have their phone out in plain sight, (Q2) I like when they let their phone alert them to calls/notifications, (Q3) I like when they check their phone for calls/notifications, (Q4) I like when they answer calls/notifications without explanation, and (Q5) I like when they use their phone when there is a break in the conversation. The 5- point Likert-type responses to these questions were: strongly disagree (0 points), disagree (1 point), neutral (2 points), agree (3 points), strongly agree (4 points), and no answer.

The five questions on the two instruments were asked in the context of four scenarios. These were 1) when interacting with a romantic partner, 2) when interacting with a family member (other than a romantic partner), 3) when interacting with a professional superior (e.g., boss, professor), and 4) when interacting with a professional peer (e.g., coworker, classmate). Both surveys contained 20 questions. Both instruments were critiqued by the other teams during the course and where revised accordingly. This ensured that the questions were well written and free of ambiguities.

Variables

The first variable was *phubbing frequency.* This variable was measured by the responses to the questions on the first survey. For each student, the value of this variable was computed by averaging the frequency with which he or she leaves their phone out in plain sight, lets their phone alert them to calls and notifications, checks their phone for calls and notifications, answers calls and notifications without explanation, and uses their phone when there is a break in the conversation—when interacting with another person. The second variable was *affective response to being phubbed.* This variable was measured by the responses to the questions on the second survey. For each student, the value of this variable was computed by averaging how much he or she likes it when the person they are interacting with leaves their phone out in plain sight, lets their phone alert them to calls and notifications, checks their phone for calls and notifications, answers calls and notifications without explanation, and uses their phone when there is a break in the conversation.

Procedures

The data for the study was collected over a one-week period. The team collected survey responses from college students who were friends and acquaintances. Thus, convenience sampling was used. Before completing a survey, each student was required to read and sign an informed consent form approved by the academic institution's Institutional Review Board. Half of the students completed survey one, and half of the students completed survey two. Two groups of students were surveyed so that the responses to the questions on one survey would not influence the responses to the questions on the other survey. In addition, students were asked not to complete a second survey if they had already completed the analogous one. This was done to prevent the cross pollination of survey responses. And finally, the team attempted to gather survey responses from an equal number of males and females.

Statistical Methods

Since the data on the surveys were ordinal in nature, and since they were displayed to the respondents as increasing numeric values (i.e., 0-4), the analysis of means was considered appropriate and pragmatic for this research. Thus, two-sample t-tests were used to determine whether or not the mean phubbing frequency and the mean affective response to being phubbed differed significantly when interacting with a romantic partner, a family member (other than a romantic partner), a professional superior, or a professional peer. The alpha level for all two-sample t-tests was set at 0.05.

Results

Subjects and Sample Size

The subjects in the study consisted of 100 undergraduate students from a small, Midwestern, coeducational, liberal arts college. The demographic results were: Sex (Male 48%, Female 51%, Not Reported 1%) and Age ($\overline{x} = 20.49$).

Survey Response Means

Table 1 shows the results of the two surveys. As can be seen, when interacting with a romantic partner, the mean phubbing frequency was 2.36, and the mean affective response to being phubbed was 1.73. When interacting with a family member (other than a romantic partner), the mean phubbing frequency was 2.78, and the mean affective response to being phubbed was 2.05. When interacting with a professional superior, the mean phubbing frequency was 0.58, and the mean affective response to being phubbed was 1.28. And when interacting with a professional peer, the mean phubbing frequency was 2.13, and the mean affective response to being phubbed was 1.78.

Table 1: Results of the two surveys (*p $<$ 0.05, **p $<$ 0.01).

	Phubbing Frequency			Affective Response to Being Phubbed					
Interaction	**N**	**Mean**	**SD**	**N**	**Mean**	**SD**	**T-Value**	**P-Value**	**Signif**
Romantic Partner	49	2.36	0.81	49	1.73	0.50	4.61	0.00	**
Family Member	50	2.78	0.64	50	2.05	0.44	6.57	0.00	**
Professional Superior	50	0.58	0.55	50	1.28	0.61	-5.99	0.00	**
Professional Peer	50	2.13	0.84	50	1.78	0.60	2.42	0.02	*

Hypothesis Testing

In order to reject a stated null hypothesis, the mean phubbing frequency and the mean affective response to being phubbed had to be significantly different. Since the mean phubbing frequency ($\overline{x} = 2.36$) and the mean affective response to being phubbed ($\overline{x} = 1.73$) were significantly different ($p < 0.01$) when interacting with a romantic partner, null hypothesis H10 was rejected. Thus, phubbing frequency in college students is *inconsistent* with their affective response to being phubbed in romantic relationships. Since the mean phubbing frequency ($\overline{x} = 2.78$) and the mean affective response to being phubbed ($\overline{x} = 2.05$) were significantly different ($p < 0.01$) when interacting with a family member (other than a romantic partner), null hypothesis H20 was also rejected. Thus, phubbing frequency in college students is *inconsistent* with their affective response to being phubbed in familial relationships. Since the mean phubbing frequency ($\overline{x} = 0.58$) and the mean affective response to being phubbed ($\overline{x} = 1.28$) were significantly different ($p < 0.01$) when interacting with a professional superior, null hypothesis H30 was rejected as well. Thus, phubbing frequency in college students is *inconsistent* with their affective response to being phubbed in professional relationships with superiors. And finally, since the mean phubbing frequency ($\overline{x} = 2.13$) and the mean affective response to being phubbed ($\overline{x} = 1.78$) were significantly different ($p < 0.05$) when interacting with a professional peer, null hypothesis H40 was rejected. Thus, phubbing frequency in college students is *inconsistent* with their affective response to being phubbed in professional relationships with peers.

Reliability and Validity

Cronbach's Alpha was used to determine the reliability (internal consistency) of the questions on the two survey instruments. The outcomes of these analyses were interpreted using the following scale: < 0.50 (Unacceptable), 0.50 $<$ 0.60 (Poor), 0.60 $<$ 0.70 (Questionable), 0.70 $<$ 0.80 (Acceptable), 0.80 $<$ 0.90 (Good), 0.90 (Excellent). The reliability threshold on all measures was set at 0.70, which is typically viewed as acceptable reliability. The reliability of the individual measures on the first instrument were Q01-Q05 = 0.7771 (n=48), Q06-Q10 = 0.7738 (n=50), Q11-Q15 = 0.7128 (n=50), and Q16-Q20 = 0.8361 (n=49). The overall reliability of the first instrument was Q01-Q20 = 0.7928 (n=47), which is at the very top of the acceptable range (almost to the good range). The reliability of the individual measures on the second instrument were Q01-Q05 = 0.5182 (n=46), Q06-Q10 = 0.6708 (n=48), Q11-Q15 = 0.7906 (n=49), and Q16-Q20 = 0.8682 (n=50). The overall reliability of the second instrument was Q01-Q20 = 0.8371 (n=43), which is in the good range. Only the reliabilities of Q01-Q05 and Q06-Q10 should be viewed with caution.

The face validity of the two measures was good as most students in the age group studied would expect a survey on phubbing to include items about leaving one's phone out in plain sight, letting one's phone alert them to calls and notifications, checking one's phone for calls and notifications, answering calls and notifications without explanation, and using one's phone when there is a break in the conversation. In fact, the team generate the list of phubbing behaviors for this study. The content validity of the two measures was also good as those five behaviors are all commonly-observed signs of phubbing according to the students on the team.

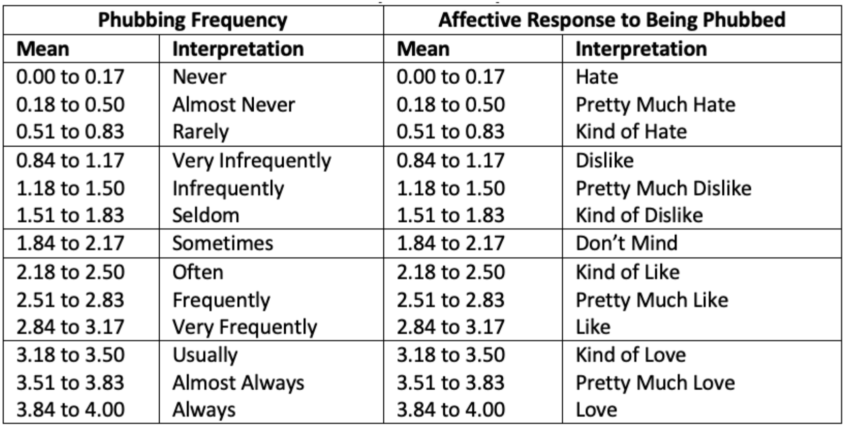
Discussion

Since the data collected in this study was ordinal in form, a table of numeric intervals was created to assist in the interpretation of the means displayed in Table 1 with finer granularity. Table 2 shows these numeric intervals and their respective interpretations. Notice that the numeric interval of 0.00 to 0.17 (interpreted as *Never* and *Hate*) and the numeric interval of 3.84 to 4.00 (interpreted as *Always* and *Love*) span one third of a point *combined*, whereas the remaining numeric intervals span one third of a point *each*. This was done so that the midpoint interpretations of *Sometimes* and *Don't Mind* straddle the midpoint of 2.00 on the 4-point scale. Note: Whenever an interpretation from the table below is used in this discussion, it will be displayed in italics.

Romantic Partners

The results of this study suggest that college students often phub romantic partners ($\overline{x}$ = 2.36), but they *kind of dislike* being phubbed by romantic partners themselves ($\overline{x}$ = 1.73). Thus, there is a moderate disconnect between college student phubbing behavior with romantic partners and how they themselves feel about being phubbed by them. This could be a cause of concern, since the results of a study of 400 adolescents and youths by Davey et al. [3]

Table 2: Numeric Intervals and Their Respective Interpretations.



Phubbing Frequency		Affective Response to Being Phubbed	
Mean	**Interpretation**	**Mean**	**Interpretation**
0.00 to 0.17	Never	0.00 to 0.17	Hate
0.18 to 0.50	Almost Never	0.18 to 0.50	Pretty Much Hate
0.51 to 0.83	Rarely	0.51 to 0.83	Kind of Hate
0.84 to 1.17	Very Infrequently	0.84 to 1.17	Dislike
1.18 to 1.50	Infrequently	1.18 to 1.50	Pretty Much Dislike
1.51 to 1.83	Seldom	1.51 to 1.83	Kind of Dislike
1.84 to 2.17	Sometimes	1.84 to 2.17	Don't Mind
2.18 to 2.50	Often	2.18 to 2.50	Kind of Like
2.51 to 2.83	Frequently	2.51 to 2.83	Pretty Much Like
2.84 to 3.17	Very Frequently	2.84 to 3.17	Like
3.18 to 3.50	Usually	3.18 to 3.50	Kind of Love
3.51 to 3.83	Almost Always	3.51 to 3.83	Pretty Much Love
3.84 to 4.00	Always	3.84 to 4.00	Love

suggest that phubbing others decreases relationship health and increases depression—the latter of which has its own negative effects on relationships. In addition, the results of a study of 153 college students by Chotpitayasunondh and Douglas [2] suggest a significant negative correlation between phubbing intensity and relationship satisfaction as well as between phubbing intensity and communication quality, which is vital to good romantic relationships. Perhaps the reason for these findings is that phubbing can create the perception that what is happening on the phubber's phone is more important than their romantic partner [4].

Family Members (Other Than Romantic Partners)

The results of this study also suggest that college students *frequently* phub family members (other than romantic partners) ($\overline{x} = 2.78$). However, they *don't mind* being phubbed by family members themselves ($\overline{x} = 2.05$). We should keep in mind, however, that what applies to romantic relationships likely applies to familial relationships—phubbing decreases relationship health, increases depression (which has its own negative effects on relationships), decreases relationship satisfaction, and decreases communication quality (which is vital to good familial relationships) [3, 2]. Perhaps the reason for these findings is that phubbing can create the perception that what is happening on the phubber's phone is more important than their family members [4]. In addition, Seppälä cites research which suggests that, when one uses one's phone while eating with family, their own enjoyment of the meal decreases, and their degree of personal engagement with their family members decreases [4]. Perhaps the reason students phub family members so frequently is that they take their relationships with them for granted. And perhaps the reason they don't mind being phubbed by family members is that they feel secure in their relationships with them.

Professional Superiors

The results of this study also suggest that college students rarely phub professional superiors ($\overline{x} = 0.58$), and they *pretty much dislike* being phubbed by professional superiors themselves ($\overline{x} = 1.28$). Perhaps the reason students don't make a habit of phubbing bosses and professors is that it might be perceived as disrespect and/or unprofessionalism, and it might harm their professional reputations. According to Gupta [5], phubbing does not demonstrate dedication to one's work, and students know this inherently when they are interacting with their professional superiors. Perhaps a reason students don't appreciate being phubbed by bosses and professors is that it makes them feel neglected professionally. Another reason might be that they think they should be shown the same courtesies that they show their professional superiors. Again, Gupta [5] asserts that phubbing does not demonstrate dedication to one's work, and this fact is not lost on students when they are interacting with their professional superiors.

Professional Peers

Finally, the results of this study suggest that college students sometimes phub professional peers ($\overline{x} = 2.13$), but they *kind of dislike* being phubbed by professional peers themselves ($\overline{x} = 1.78$). Thus, there is a moderate disconnect between student phubbing behavior with professional peers and how they themselves feel about being phubbed by them. As mentioned earlier, the results of the Chotpitayasunondh and Douglas [2] study suggest a significant negative correlation between phubbing intensity and communication quality. Thus, when phubbing occurs, communication quality (which is crucial when working with professional peers) decreases. Perhaps the reason students feel somewhat comfortable phubbing their professional peers is that they don't mind being phubbed by their professional peers that much.

Implications

Davey et al. [3] assert that students need guidance from family members, healthcare workers, teachers, and others to help them control their phubbing behaviors in an effort to promote better social, relational, and mental health. In addition, Roberts and David [8] assert that organizations need to create clear policies on phone use in the workplace and provide training to educate workers with regard to those policies. Fortunately, colleges and universities are in a great position to help educate students concerning the negative consequences of phubbing on social, relational, and mental health as well as the expectations that will be placed on them when they enter the work world.

Professors should institute and adhere to reasonable restrictions on phone use and should model proper phone use themselves. These can help students form life- and work- improving habits [4]. Permitting students to use their phones whenever they wish will likely contribute to the problems associated with the phubbing behaviors discussed in this article. A study of 413 corporate adults by Roberts and David [8] suggests that, when supervisors phub their subordinates, supervisory trust decreases. And when supervisory trust decreases, both psychological meaningfulness (i.e., feelings that one's work is valuable or conducive to one's professional growth) decreases and psychological availability (i.e., confidence in one's ability to carry out one's work) decreases. And when psychological meaningfulness and psychological availability decrease, employee engagement (i.e., the ability to focus on the task at hand) decreases. The implications for college faculty are clear. When professors phub their students, the students trust their professors less. When students trust their professors less, they feel like their work is less valuable or conducive to their professional growth, and they feel like they are less able to carry out their work. And when these conditions are present, students are less able to focus on their academic work.

As demonstrated earlier, college students don't (for the most part) like being phubbed by others. Thus, they are aware of the fact that phubbing has negative consequences. However, they continue to phub others often or frequently (with the exception of their professional superiors), which may be a symptom of deeper issue. One possibility is that students feel no moral obligation to treat others as they would like to be treated. Another possibility is that phone use is a behavioral addiction, which is characterized by behaviors that reward impulses despite their adverse consequences [6].

References

[1] Oxford living dictionary, "phubbing,". `https://en.oxforddictionaries.com/definition/phubbing`.

[2] Varoth Chotpitayasunondh and Karen M Douglas. The effects of "phubbing" on social interaction. *Journal of Applied Social Psychology*, 48(6):304–316, 2018.

[3] Sanjeev Davey, Anuradha Davey, Santosh K Raghav, Jai V Singh, Nirankar Singh, Agata Blachnio, and Aneta Przepiórkaa. Predictors and consequences of "phubbing" among adolescents and youth in india: An impact evaluation study. *Journal of family & community medicine*, 25(1):35, 2018.

[4] J. Ducharme. Phubbing is hurting your relationships. here's what it is. `http://time.com/5216853/what-is-phubbing/`.

[5] S. Gupta. Here's how you can be more professional at work. `https://www.entrepreneur.com/article/317946`.

[6] Engin Karadağ, Şule Betül Tosuntaş, Evren Erzen, Pinar Duru, Nalan Bostan, Berrak Mizrak Şahin, İlkay Çulha, and Burcu Babadağ. Determinants of phubbing, which is the sum of many virtual addictions: A structural equation model. *Journal of behavioral addictions*, 4(2):60–74, 2015.

[7] Newport Academy. Phubbing and why it's bad for us. `https://www.newportacademy.com/resources/mental-health/phubbing-why-its-bad-for-us/`.

[8] James A Roberts and Meredith E David. Put down your phone and listen to me: How boss phubbing undermines the psychological conditions necessary for employee engagement. *Computers in Human Behavior*, 75:206–217, 2017.

Automating Configuring Parallel Compute Environments for Students*

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

Abstract

In this research, we used Jetson Nano boards and Ansible playbooks to simplify the configuration of small clusters for students to use in a parallel programming course. Along the way, we discovered some unexpected use cases and outcomes when students repurposed the playbooks. We will also discuss how these methods could be used in the future, especially as less expensive Jetson Nano boards become a more readily available option.

1 Introduction

When approaching teaching our Numerical Methods and Parallel Programming course, a common complaint gathered from students over the years was that the assignments required using a shared computing environment, which often resulted in extra noise in their performance metric measurements and made it harder for them to create conclusions about their parallel performance observations confidently. In some cases, students had to wait for days before their code could run, resulting in them not getting runs completed before the assignment due dates.

There are quite a few examples of using small embedded boards, like Raspberry Pis, to build clusters for computing and education[13, 15, 10]. They are not limited to small scale deployments either, as at least one US research lab

*Copyright ©2021 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

built a 750 node Raspberry Pi cluster, and there were talks about expanding it to 10000 nodes[17, 5]. A CCSC paper from a few years ago even explored how to have students build such a setup on the cheap[16].

These were all examples of homogenous compute clusters, and the current top US supercomputers are being built on heterogenous compute nodes consisting of compute processors and GPUs[8]. Furthermore, the future of supercomputers is moving towards these heterogeneous architectures as well[7]. Given that the future of parallel programming in supercomputing is moving toward a hybrid CPU and GPU compute node configuration, it makes sense to give students the ability to work with such a system on a small scale. It happened that in March 2019, NVIDIA launched its Jetson Nano embedded boards that retailed for $99 each[2]. NVIDIA has since launched a new cheaper model for $59. This platform makes for an even more attractive board for building a small cluster with GPUs on each node that can replicate a small scale version of these heterogeneous architectures.

2 Parallel Course

Our university has a Numerical Methods and Parallel Programming course, and it would be instrumental for the students to develop on such a cluster. In most cases, a simple two-node cluster is more than adequate to learn some of the basics of multi-node programming that we explore in the class. When teaching this course, the focus was on covering a breadth of topics related to parallel, in contrast to the previous approach of taking a deep dive into MPI[9]. As such, it was beneficial for the cluster configured to support Apache Spark in addition to Open MPI[1].

3 Guide and Playbooks

To facilitate the cluster deployment and make it easier for the students, we built a guide and a set of Ansible playbooks that the students could use to build their own small scale clusters[11, 14]. The guide included a few things: a part list, an initial setup, and how to run the Ansible playbooks.

The part list included all the parts they would need to purchase to build a small two node Jetson cluster, with links to places they could buy them if needed. There was no requirement students had to buy the Jetson boards for the class; however, with the cheaper $59 nano board option, it would be more reasonable to request that students purchase at least a single board and work with a partner to make a small cluster. There was no textbook required for the class, which minimize the expenditures required for these assignments. Students who opted not to build their own Jetson clusters ran into the same shared environment issues that motivated the idea of providing this set of tools to help students run their small clusters.

The guide then included a set of steps to do the following:

1. Statically assign IPs to the Jetson boards
2. Generate RSA keys and copy the keys from their local computers to each of the boards
3. Get Ansible installed in a local Python virtual environment
4. Modify the Ansible config file provided with their board IPs and the username used on the boards
5. Test their Ansible config by pinging their inventory

Ansible is a configuration management tool that simplifies software deployment to numerous systems. Such tools are beneficial and common for managing large clusters. We chose to create several separate playbooks instead of putting all configuration steps into a single Ansible playbook. These separate, distinct playbooks allow the students, and potentially anyone else that may use the playbooks, to focus on the parts they require. The separate playbooks help prevent users from installing packages and tools that are unnecessary for their use case. The guide currently has the following playbooks:

- all.yml - Runs all of the playbooks at once
- initial.yml - To install updates and the basic GCC, G++, BLAS, and OpenMPI libraries
- nfs.yml - Configure NFS, shared home, and share SSH keys
- hosts.yml - Configure the /etc/hosts on the nodes to match the inventory names
- python.yml - Configure Python3, pip3, OpenMPI for Python3 bindings, Numpy for Python3
- spark.yml - Install and configure Apache Spark on the nodes
- rust.yml - Currently only installs Rust on the primary node
- clang.yml - Installs Clang 7 on the cluster and is necessary for Rust
- zsh.yml - Installs Zsh on the cluster but does not force it as the default shell

For students who did not want to use the playbook for simplifying things, the guide also includes some step by step manual install instructions for just getting MPI and the shared home directory set up. Moreover, it included directions to run the Linpack HPL Benchmark if students were curious about how their cluster would benchmark compared to a real super computer[6].

3.1 Use in Class

The parallel programming course has been taught with the Ansible playbooks and Jetson Cluster guide one time so far. We did not require students to buy a cluster. We did recommend they share clusters to cut down on the costs per student. The feedback was overwhelmingly positive for the students who did purchase a cluster or paired up to make a cluster.

3.2 Other Uses

One of the unexpected outcomes was students choosing to combine nodes to create larger than two-node clusters. When students add additional worker nodes to the Ansible configuration, the playbooks will scale to configure the more massive cluster correctly. Allowing the playbooks to scale was a planned design, as there was hope students might combine resources to run a more massive cluster for their final projects. We tested the scaling playbook functionality by updating a cluster designated for teaching from a two-node to a four-node cluster. One enterprising student made a large cluster with nodes from other students and even gave remote SSH access to the cluster he was hosting to other students in the class.

We also had a couple of students indicate that they could use the playbooks for pretty much any set of Debian nodes they wanted. There were examples of the playbooks being used to configure two Google Cloud compute instances as a small cluster and another instance where the playbooks aided in the configuration of two local VMware Virtual Machine Instances as a small cluster[4, 3].

The GP-GPU features of the Jetson Nano boards are baked into the Jetson Nano hardware and Ubuntu instance already. Since the Jetson Nano and Jetson Nano 2G both feature a 128-core Maxwell GPU support CUDA programming, the CUDA drivers come installed by default. The playbooks do not need to assist in the installation; it is just a matter of leveraging those cores in the code running on each node. The use beyond the Jetson boards was a neat takeaway, as it means that the playbooks can help anyone easily configure a small cluster regardless of the system, as long as it is running Ubuntu or a Debian Linux environment[12].

4 Conclusion

The students appreciated being able to develop cluster code for a system they could see and touch. The playbooks and guide made it far less painful for them to get started with parallel programming and get a cluster up and running for themselves. They also liked how Ansible made setting up such a cluster for personal use relatively straightforward. The playbooks served as a good starting point that the students could build upon when developing their Jetson board projects with their local configurations. Students could set up the cluster with minimal effort since the playbooks took care of all the hard or tedious setup. Now that newer, less expensive Jetson boards are available, the students might be even more open to the idea of purchasing such boards. A cluster created for parallel programming could find use in future courses or personal projects.

If we can get funding, we also plan to take these playbooks and scale them out to help manage a larger set of hosted Jetson Nano boards, as this would allow students who do not want to host their own to have access to a cluster.

References

[1] `https://spark.apache.org/`.

[2] Buy the latest Jetson products. `https://developer.nvidia.com/buy-jetson`.

[3] Delivering a digital foundation for businesses. `https://www.vmware.com/`.

[4] Google cloud. `https://cloud.google.com`.

[5] LANL Turns to Raspberry Pi for Supercomputing Solution. `https://top500.org/news/lanl-turns-to-raspberry-pi-for-supercomputing-solution/`.

[6] The LINPACK benchmark. `https://www.top500.org/project/linpack/`.

[7] LLNL and HPE to partner with AMD on El Capitan, projected as world's fastest supercomputer. `https://www.llnl.gov/news/llnl-and-hpe-partner-amd-el-capitan-projected-worlds-fastest-supercomputer`.

[8] LLNL sierra. `https://computing.llnl.gov/computers/sierra`.

[9] Open MPI. `https://www.open-mpi.org/`.

[10] Raspberry Pi. `https://www.raspberrypi.org/`.

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

[12] Ubuntu Linux. `https://ubuntu.com/`.

[13] Pekka Abrahamsson, Sven Helmer, Nattakarn Phaphoom, Lorenzo Nicolodi, Nick Preda, Lorenzo Miori, Matteo Angriman, Juha Rikkilä, Xiaofeng Wang, Karim Hamily, et al. Affordable and energy-efficient cloud computing clusters: The bolzano Raspberry Pi cloud cluster experiment. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, volume 2, pages 170–175. IEEE, 2013.

[14] Bryan Dixon. Jetson cluster. `https://github.com/csuchico-csci551/JetsonCluster`.

[15] Kevin Doucet and Jian Zhang. Learning cluster computing by creating a Raspberry Pi cluster. In *Proceedings of the SouthEast Conference*, pages 191–194, 2017.

[16] Samuel Holt, Andrew Meaux, Jacob Roth, and David Toth. Making the one cluster per student method of teaching parallel computing financially practical. *Journal of Computing Sciences in Colleges*, 33(4):106–113, 2018.

[17] Bruce Tulloch. Raspberry Pi clusters come of age. `https://www.raspberrypi.org/blog/raspberry-pi-clusters-come-of-age/`.

Godot Engine and Checklist-Based Specifications: Revising a Game Programming Class for Asynchronous Online Teaching[*]

Paul Gestwicki
Computer Science Department
Ball State University
Muncie, IN 47306
pvgestwicki@bsu.edu

Abstract

This experience report describes the revisions to an undergraduate elective game programming course that were made in response to the COVID-19 pandemic. The course transitioned from a lab-based, in-person class to an online, asynchronous one. This required a change in teaching technology from Unreal Engine 4 to Godot Engine. The course expanded its use of checklist-based specifications grading in order to facilitate student autonomy with minimal reduction in creativity and motivation. The course revisions required significant time investment, but the results were positive.

1 Introduction

The pandemic surprised everyone in Spring 2020, and institutions of higher education were pressed to make radical changes for the coming Fall. Many faculty put in extraordinary effort to accommodate these changes into our course plans. This experience report describes changes made to one such course: an upper-level Computer Science department elective on game programming. The

[*]Copyright ©2021 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

course was traditionally offered face-to-face in a lab environment, but due to the pandemic, it transitioned to asynchronous and online. This necessitated a cascading change in the instructional technology, switching from Unreal Engine to Godot Engine.

This report covers the course changes and their impact. The course itself is described, followed by discussions of engine selection and assessment strategy. The complete course plan, which includes all of the project specifications and assessment instructions, can be found online[1] and is provided under a Creative Commons Attribution ShareAlike license.

2 Context

The game programming course under discussion is a three credit-hour elective. The prerequisites include three semesters of programming-focused courses as well as a course on discrete mathematics. Most students who take the course are junior or senior Computer Science majors, although it can also be the final course taken by Computer Science minors. Formally, the course is not itself a prerequisite for any other course, although some students use it as preparation for working on game projects in capstones or multidisciplinary team projects. The learning objectives of the course cover fundamental techniques and processes of game programming, emphasizing technical concerns over matters of game design. That is, the course is about how to program games, not what games to make.

The technology, language, and platform are left to the instructor's discretion. For several years, the course used PlayN [5], a low-level game programming library for Java. When funding from the college allowed for an update to lab computers a few years ago, the course switched to Unreal Engine (UE4). UE4 is a "AAA" engine used for games, architectural visualization, realtime video production, and simulations; it is perhaps most well known to students as being the engine that runs the popular game *Fortnite*[2] and for being used in the production of *The Mandalorian* [3].

When teaching in-person or on-campus, students have access to Computer Science department lab machines that meet the significant hardware requirements for developing with UE4. However, not all students have access to such hardware when taking online courses. Hence, it would not have been feasible to continue teaching with UE4 in the online Fall 2020 semester, and a platform with more modest hardware requirements was needed.

[1]`https://www.cs.bsu.edu/homepages/pvgestwicki/courses/cs315Fa20`
[2]`https://www.epicgames.com/fortnite`

3 Modality

It was not just the hardware that had to change but also the mode of instruction. The Game Programming course had always been taught in a traditional, face-to-face format, even before the lab was equipped with specialized hardware for game development. The university's response to the pandemic was to move many courses into an online asynchronous mode, including the Game Programming course along with almost all of the department's non-introductory courses.

Many of the traditional teaching methods used in the course were made infeasible by the transition to an asynchronous online modality. For several years, the course design had drawn inspiration from studio-based learning (SBL), which is an approach in which students create artifacts that represent their learning and share those, formally and informally, with peers and experts [13]. SBL is built on assumptions of shared space and shared time, neither of which were available for Fall 2020. The shared space can potentially be simulated via remote meeting technologies, but working asynchronously means that the rapid, face-to-face feedback required for authentic SBL is impossible.

In the absence of SBL, let alone any face-to-face or synchronous interaction, an approach had to be adopted that would maximize student autonomy. What could normally be done with on-the-fly adaptation and intervention had to become codified expectations and presentation. However, execution ought not impinge upon student creativity, lest it violate both the spirit of game development and the necessary motivational aspects that allow students to get through the challenging material. From the perspective of self-determination theory [15], one needs to recognize the importance of intrinsic motivation to student success.

4 Godot Engine

A *game engine* is a software system that combines several tools and libraries that are useful for game development. These are often brought together into one interface in the same way that an integrated development environment provides for general software development. Unity, Unreal Engine, and GameMaker are three popular game engines.

Godot Engine is a game engine with several noteworthy features, including: supporting both 2D and 3D game development; running on multiple platforms; building for multiple platforms, including HTML5 and mobile; tooling for 2D and 3D animation; and a custom scripting language—GDScript—that is heavily inspired by Python. Godot Engine has significantly more modest hardware requirements than UE4, satisfying this important constraint. Additionally, the project website at `godotengine.org` includes extensive documentation specifically to help beginners.

Godot Engine is free and open source software (FOSS), licensed under the Expat ("MIT") license. The roots of the engine go back to 2001 according to a visual history provided by Juan Linietsky [11], and in an interview, co-creator Ariel Manzur has described the real development as starting in 2007 [18]. In 2014, Godot Engine was released as free and open source software (FOSS). Software freedom is an important factor when selecting instructional technology since it preserves the rights of the users to learn and share [17] while also promoting a culture of creativity and invention [10].

5 Engine Popularity

Godot Engine is one of many that might satisfy the requirements for the course. Engine popularity is worth considering in addition to fitness for pedagogic purpose. This issue should be familiar to computing educators: we teach in Python and Java despite having countless books and papers explaining how to do it in Pascal.

Unfortunately, popularity is hard to measure. In the absence of any established, rigorous analysis of what engines are being used in practice, one can build an understanding of the field by looking at engines used in game jams. Kultima [8] conducted a meta-analysis of game jam scholarship and based on that, defines a game jam as "an accelerated opportunistic game creation event where a game is created in a relatively short timeframe exploring given design constraint(s), and end results are shared publicly." Since the publication of this meta-analysis, the number of jams appears to have grown dramatically due to the popularity of `itch.io` as a hub for hobbyist and amateur game developers. There had not previously been such a hub, but now, `itch.io/jams` regularly lists over a hundred simultaneous jams at any time. It is worth being aware of these jams since they represent a complex learning environment that is emerging online outside the influence of computer science educators. Ito *et al.* [7] provide a good introduction to informal "new media" learning, while Faas *et al.* [2] have described how the emergence of streaming game development provides new and powerful affordances for learning programming in particular.

Global Game Jam (GGJ) is the world's largest annual game development event. Interested readers should refer to Fowler *et al.* [4] for a history of GGJ. User-reported data about jam entries is publicly available from `globalgamejam.org`. Table 1 shows the relative popularity of game engines used in GGJ submissions in the years 2019–2021. The years 2019, 2020, and 2021 had 9000, 9598, and 6381 submissions, respectively. The dramatic drop in submissions in 2021 is undoubtedly an effect of the pandemic.

Table 1: Game Engine Popularity at Global Game Jam (2019–2021)

Engine or Library	**2021**		**2020**		**2019**	
GameMaker	160	2.5%	281	2.9%	337	3.7%
Godot Engine	312	4.6%	358	3.7%	191	2.1%
libGDX	9	0.2%	12	0.1%	20	0.2%
Processing	8	0.1%	13	0.1%	23	0.3%
Scratch	16	0.3%	33	0.3%	23	0.3%
SDL	16	0.3%	31	0.3%	25	0.3%
Unity	4104	64.3%	6246	65.0%	5852	65.0%
Unreal Engine	547	8.6%	666	6.9%	606	6.7%

Data about popular engines Unity, Unreal Engine, GameMaker, and Godot Engine are provided along with those for other potentially interesting engines. While PlayN—the previous library used in the game programming course—is not tracked in the GGJ database, it is closely related to libGDX, another low-level Java-based game development library whose data are included. Other notable engines and libraries include: Processing, which is often used to introduce programming concepts in art schools and visual arts programs, and it is sometimes found in Computer Science programs as well; Scratch, the venerable block-based programming environment for beginners [12]; and SDL, which is not an engine at all but is a low-level, cross-platform library for game programming, often used by students and hobbyists to make games and engines.

In summary, then, while Unity is the dominant force, Godot Engine's popularity in jams is on par with other competitors and significantly higher than some other CS-friendly options. Being FOSS, it manifests the value of software freedom, which is important as a matter of access and also to prevent unnecessary constraints from being imposed on our student learners. Godot Engine is also used by Indiana University's game design major, which gives credence to its fitness for purpose in higher education. With Godot Engine being a good candidate for a platform, attention must now be paid to the asynchronous online pedagogy.

6 Checklist-based specifications grading

The course kept its traditional structure of being divided into two roughly equal parts. The first half of the semester was divided into weekly assignments that helped students to build fundamental skills. The second half of the semester was devoted to a final project that was completed in three iterations. Each iteration involved submitting a formally evaluated increment. In a normal semester, students would share these results in the lab with their classmates following SBL. For the online asynchronous class, the games were posted to the Canvas discussion board, where classmates were encouraged to give feedback.

For the past three years, the class has used specifications grading as a primary evaluation technique. Specifications grading is an implementation of contract-based grading in which students' grades are determined by the extent to which they meet clear, objective specifications [14]. Largent [9] has reported on the efficacy of specifications grading for Computer Science education. Following this approach, then, the requirements for each project were divided into criteria for grades `A`, `B`, `C`, and `D`. Meeting all of the `D` criteria earned a `D`, additionally meeting all the `C` criteria earned a `C`, and so on. The specifications were designed so as to encourage autonomous practice and maintain intrinsic motivation. The increased specificity of the instructions for asynchronous instruction is manifest in the word count of the project descriptions, which ballooned from 3,673 in Fall 2019 to 11,632 in Fall 2020.

Checklist-based specification is an original extension of specifications-based grading in which students self-report their completion of the specifications. Figure 1 shows the checklist-based specification for the first week's assignment. Students found the checklist after a detailed, prose explanation of the week's tasks. The technical work for this specific week included completing a standard Godot Engine tutorial.[3] The week also familiarized students with other aspects of the class, including submitting work via URL to Canvas, using the discussion board to establish a sense of community, proper use of version control, and the inclusion of project reports with each submission.

While checklist-based grading was used in the past, Fall 2020 also saw the addition of final project "stars," which were required for for `A` and `B` grades. Students could earn stars in various ways, rather than being tied to fixed criteria as shown in Figure 1. Earning an `A` over a `B`, then, was a matter of earning more stars. Readers familiar with Nilson [14] will recognize this as embedding the "more hurdles" philosophy within the broader context of "higher hurdles".

Each submission required a project report in which the student provided a self-assessment and reflection. The self-assessment comprises a completed checklist along with a statement of what grade is earned. Given the project report, then, the instructor's task is verification of students' claims rather than grading each submission from scratch. The reflection has the students share at least one paragraph describing something that they found interesting or challenging.

The reflection requirement has two clear benefits. First, it provides some

[3]`https://docs.godotengine.org/en/stable/getting_started/step_by_step/your_first_game.html`

Figure 1: Sample checklist-based specification for grading

Meeting the following criteria earns a `D` or better grade; failing to meet these criteria earns an `F` grade:

- ❏ D-1: The repository link is submitted before the project deadline.

Meeting all previous criteria as well as the following earns a `C` or better grade:

- ❏ C-1: The tutorial has been completed.
- ❏ C-2: You have introduced yourself on the discussion board.

Meeting all previous criteria as well as the following earns a `B` or better grade:

- ❏ B-1: Your `.gitignore` file is correctly specified in the repository.

Meeting all previous criteria as well as the following earns an `A` grade:

- ❏ A-1: The project report is complete as per the instructions, with a reflection and self-assessment.

feedback to the instructor about the challenges students have faced and where their interests are taking them. This is important because, in an online asynchronous environment, there are few affordances to "learn by looking around." This impacts SBL practice as previously mentioned, but it has even broader implications for software development: Cockburn [1] wrote about our capacity to process ambient information as a major contribution to the success of agile software development projects, but the affordance for such activity is the first that is lost when making a course asynchronous. The reflection therefore addresses, in part, a major complication of the modality. The second purpose for the requirement is to inculcate an affinity for reflective practice [16]. That is, it sends the message that getting the project working is only part of the activity: they should be thinking about what they are doing while they are doing it.

7 Observations

Checklist-based grading has been used in the course since Fall 2018, and these continued to work as intended despite the change in technology. The comprehensive documentation of expectations, rather than the combination of documentation and extemporaneous in-person discussion, appeared to yield less student confusion around weekly requirements. This clarity was especially useful in the absence of shared lab time. Having large weekly projects meant that there was a significant amount of material to consider while grading, but the students' self-assessments meant that grading was focused on verification rather than comprehensive review. For example, if a student was satisfied with submitting `C` work, there is no need to check whether they have completed the requirements for an `A` or `B`. Furthermore, if a student had checked a requirement that was not satisfied, feedback could unambiguously indicate this.

Despite the overall successes of the assignment format, a minority of students struggled with the seemingly simple requirement that they state what grade they have earned. There were inevitably multiple students who made comments indicating that they *hoped* for a certain grade rather than having done the work of *verifying* that they have demonstrated competence. It is not clear what, if anything, helps them recognize that it is their capacity and responsibility to do the latter rather than the former. This points to a potential area for future qualitative research, to better understand the mindset of the students regarding the balance of responsibilities in higher education.

The "star" system yielded mixed results. It was intended to show students the variety of valuable activities that they could pursue, but it was clear that most students simply settled for the ones that required the least effort. It makes the endeavor seem like a presentation of false choices, which, ironically, is a common design flaw in game design. Some students did pursue unconventional paths, however, and it's possible that even those who took the easier path gained some perspective or insight by reading the options. Again, further research is necessary here in order to understand all the variables at play. In the meantime, it seems that little is in presenting more options, even if engagement is less than ideal.

The course plan is clear about the course requiring nine effort-hours per week. Successful students clearly abode by this advice, although it broke down at the end of the semester. The learners made clear progress through the first and second iterations of their final projects. The final iteration, which spanned the two weeks after Thanksgiving break, showed markedly less advancement. Reviewing project history through version control showed that almost all the project activity took place in the 72 hours before the deadline, which pattern was not the case in the previous two iterations.

Reviewing students' reflection essays revealed that this was not a simple matter of putting off work until the deadline. The majority of students seemed to recognize that this was not supposed to be work due at the end of two weeks, but rather, a sustained effort that culminated in a deliverable. However, it seems the latter is unconventional, and many students seem to default to a due-date-driven management style: do the work that is due next. Once again, this merits future qualitative research to better understand students' perspective. Separately from personal management, however, many students reported that their reason for lack of progress on the final iteration was actually the mismanagement of the faculty of their other courses. That is, many reported that their other courses, rather than expecting steady progress throughout, suddenly assigned additional work in the final two weeks of the semester, which required inordinate attention. This has not been verified, but it merits reporting on here, especially in the spirit of providing a snapshot of teaching during the pandemic. Even if the students' stories are completely inaccurate representations of reality, they still represent a widely-experienced perspective.

Finally, it is worth noting that while the course design expected consistent effort to be applied during the semester, the pedagogy of the final project did not require it. During in-person sessions, social pressure is enough to keep students making progress on their projects, since they do not wish to be embarrassed in front of their classmates. There was no replacement for this during the final project. Labor logs, such as those described by Inoue [6] for college writing courses, could be employed in figure courses. They provide both a record of student activity as well as an instrument for reflection.

8 Acknowledgments

David Largent encouraged me to first explore specifications grading and provided valuable feedback on this aspect of the course design. I am also grateful to the reviewers who helped strengthen the presentation of this work.

References

[1] Alistair Cockburn. *Agile Software Development: The Cooperative Game.* Addison-Wesley, Boston, 2006.

[2] Travis Faas, Lynn Dombrowski, Alyson Young, and Andrew D. Miller. Watch me code: Programming mentorship communities on twitch.tv. *Proc. ACM Hum.-Comput. Interact.*, 2(CSCW), November 2018.

[3] Jeff Farris. Forging new paths for filmmakers on "The Mandalorian", 2020. `https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian`, retrieved March 25, 2021.

[4] Allan Fowler, Foaad Khosmood, and Ali Arya. The evolution and significance of the global game jam. In *Proc. of the Foundations of Digital Games Conference*, 2013.

[5] Paul Gestwicki. Teaching game programming with PlayN. *J. Comput. Sci. Coll.*, 31(1):90–97, October 2015.

[6] Asao B. Inoue. *Labor-Based Grading Contracts.* WAC Clearinghouse, Fort Collins, CO, 2019.

[7] Mizuko Ito, Sonja Baumer, Matteo Bittanti, danah boyd, Rachel Cody, Becky Herr Stephenson, Heather A. Horst, Patricia G. Lange, Dilan Mahendran, Katynka Z. Martínez, C. J. Pascoe, Dan Perkel, Laura Robinson, Christo Sims, and Lisa Tripp. *Hanging Out, Messing Around, and Geeking Out.* MIT Press, Cambridge, MA, tenth anniversary edition edition, 2019.

[8] Annakaisa Kultima. Defining game jam. In *Proceedings of Foundations of Digital Games (FDG) 2015*, 2015.

[9] David Largent. My exploration of specifications grading in a discussion-based course. *J. Comput. Sci. Coll.*, 33(1):89, October 2017.

[10] Lawrence Lessig. *Free Culture.* Penguin Press, London, 2004.

[11] Juan Linietsky. Godot history in images!, 2014. `https://godotengine.org/article/godot-history-images`, retrieved March 23, 2021.

[12] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The Scratch programming language and environment. *ACM Trans. Comput. Educ.*, 10(4), November 2010.

[13] N. Hari Narayanan, Christopher Hundhausen, Dean Hendrix, and Martha Crosby. Transforming the cs classroom with studio-based learning. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, page 165–166, New York, NY, USA, 2012. Association for Computing Machinery.

[14] Linda B. Nilson. *Specifications Grading.* Stylus Publishing, Sterling, VA, 2014.

[15] Richard M. Ryan and Edward L. Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist*, 55(1):68–78, 2000.

[16] Donald Schön. *The Reflective Practitioner.* Basic Books, New York, 1983.

[17] Richard M. Stallman. *Free Software, Free Society: Selected Essays of Richard M. Stallman.* GNU Press, Boston, 2nd edition edition, 2004.

[18] SteamLUG Cast, 2016. Interview with Ariel Manzur, Season 4, Episode 5, archived at `https://steamlug.org/cast/s04e05`, retrieved March 23, 2021.

Visualizing Recursion Using Code and Stack Animation[*]

Y. Daniel Liang, Jireh Bethely, Gursimran Singh Walia
Department of Computer Science
Georgia Southern University
Statesboro, GA 30458
{yliang,jb24628,gwalia}@georgiasouthern.edu

Abstract

Recursion is an important programming technique, but it is difficult to learn. Students often struggle to understand how recursion works and why a tail recursion is efficient. We created a code and stack animation to demonstrate how a recursive method is executed and another code and stack animation to demonstrate how a tail recursive method is executed. Our animations show the call stacks and illustrate interactively how the call stacks grow and shrink during the execution of a recursive method and how an activation frame is shared for a tail recursive method. These code animations enable students to trace the code step-by-step visually and help students understand the recursion and tail recursion. This paper presents these two code animations.

1 Introduction

Code animation is a visual tool for tracing the execution of the code. We have developed more than 240 code animations for Java, C++, and Python. The code animation have been integrated in the Pearson's Revel interactive ebooks [8, 5, 6], which have received positive reviews [1, 2]. In a study [1] conducted at Central Michigan University in 2015, *"62 percent of students agreed or strongly agreed that the animations in Revel that stepped line by line through code, showing what was happening in the program, helped them better understand how to replicate the coding on their own."* In another study [2] conducted

[*]Copyright ©2021 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

at the University of Louisiana in 2016, *"87 percent of students who used the animations strongly agreed or agreed that they helped them better understand how to replicate coding on their own."* The second study was conducted after we added more code animations in the book.

Code animation is effective to help instructors and students to teach and learn programming. In [7], we demonstrated how the code animation can help students to learn variables, selection statements, loops, methods, pass by value, and arrays effectively. Code animation helps students comprehend the code. More importantly, it illustrates important programming concepts using animation. Frequently, we have been asked by instructors to develop code animation for recursion. We listened, responded, and created two code animations for recursion: one for non-tail recursion and the other for tail recursion.

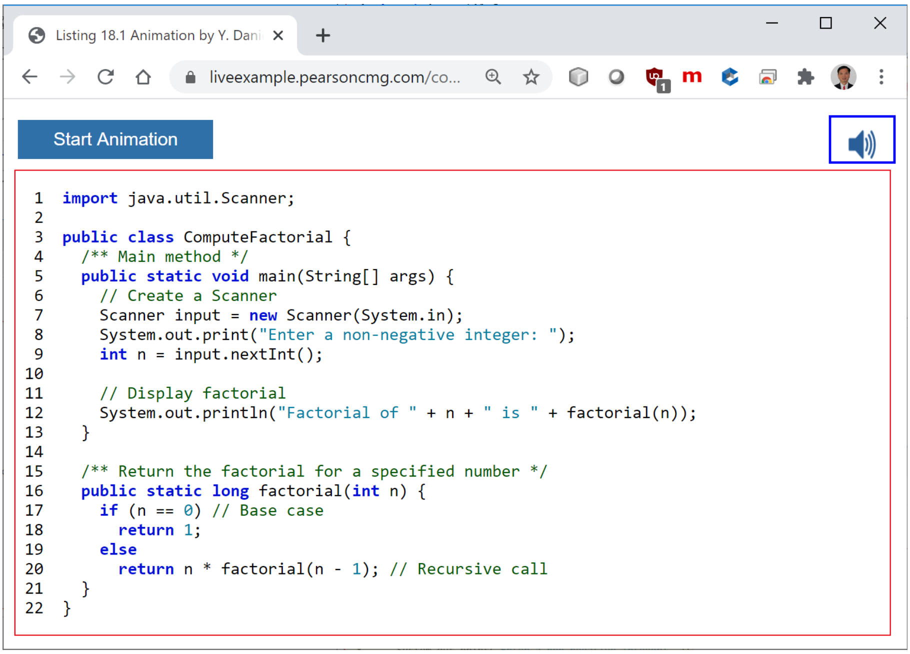
Recursion is an important programming technique. It is ideal for solving inherently recursive problems that would be otherwise difficult to solve without using recursion. Recursion is difficult to teach because students have no idea how recursion is executed behind the scenes. Many researches have been conducted on teaching recursion [3, 11, 10]. The various approaches for teaching recursion are summarized by Dann, Cooper, and Pausch [3]. Three main approaches are algorithm animation, program visualization, and using special software tools. An example of algorithm animation was the GAIGS (Generalized Algorithm Illustration through Graphical Software) package developed by Naps [9]. Using this system, students can run some prepared animations. However, this system is not Web-based and the animation does not show the call stacks. The program visualization is similar to our code animation that directly shows the execution of the code in the program. However, it does not show the change of the stack as the method is called or returned. The special software tools for teaching recursion use a software such as Alice. These special software tools do not show the call stacks. All the existing approaches are not Web-based. Our code animation is developed using HTML, CSS, and JavaScript. It is platform independent and can be viewed on any device.

It is worth to mention that Guo [4] has created code animation for several programming languages including Java, Python, and C++. However, his tool does not read input for Java and C++, does not show the explanation for each statement being executed, and does not show how a tail recursive method is called. So, we created the code animation to visualize recursion and tail recursion.

In the following sections, we will present a code animation to demonstrate how recursion works and another code animation to demonstrate how a tail recursion works. We will then present a class test report. The report shows that these animations help students better understand recursion and tail recursion.

2 Recursion Animation

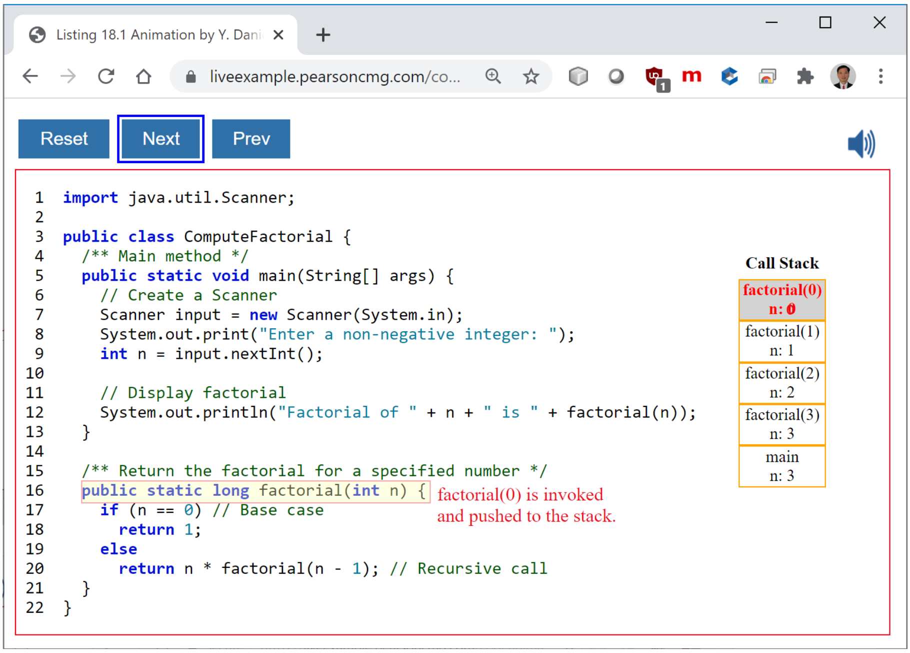
To show how recursion works, we created an animation for computing factorial using a recursive method. The factorial problem is not a good candidate for recursion, because a non-recursive solution for this problem is more efficient than using recursion. Nevertheless, it is a simple and intuitive problem for introducing the concept of recursion and for demonstrating the techniques of writing a recursive method. Because of this reason, many textbooks use the factorial problem for introducing students to recursion. You can access the code animation for this recursive program from `https://liveexample.pearsoncmg.com/codeanimation/ComputeFactorial.html`, as shown in Figure 1.


Listing 18.1 Animation by Y. Dani
liveexample.pearsoncmg.com/co...
Start Animation

```java
 1 import java.util.Scanner;
 2
 3 public class ComputeFactorial {
 4   /** Main method */
 5   public static void main(String[] args) {
 6     // Create a Scanner
 7     Scanner input = new Scanner(System.in);
 8     System.out.print("Enter a non-negative integer: ");
 9     int n = input.nextInt();
10
11     // Display factorial
12     System.out.println("Factorial of " + n + " is " + factorial(n));
13   }
14
15   /** Return the factorial for a specified number */
16   public static long factorial(int n) {
17     if (n == 0) // Base case
18       return 1;
19     else
20       return n * factorial(n - 1); // Recursive call
21   }
22 }
```

Figure 1: Code animation for recursion

The animation shows how a recursive method is executed with a stack step-by-step visually. When the program executes, the main method is invoked (line 5). The animation shows that an activation record is created and pushed to the stack for the main method. After declaring and reading n, the animation shows the value of n is stored in the activation record. When calling factorial(n) in line 12, the animation shows that a new activation record is created. To compute factorial(3), factorial(2) is recursively invoked in line 20. The animation shows that the activation record for factorial(2) is pushed to the stack. To compute factorial(2), factorial(1) is recursively invoked in line 20. The animation shows that the activation record for factorial(1) is pushed to the stack. To compute factorial(1), factorial(0) is recursively invoked in line 20. The animation shows that the activation record for factorial(0) is pushed to the stack, as shown in Figure 2. When n is 0, factorial(0) returns 1 in line 18. The animation shows that the activation record for factorial(0) is removed from the stack. Now factorial(1) is on the top of the stack, it returns 1 * factorial(0). The activation record for factorial(1) is removed from the stack. Now the animation shows that factorial(2) is on the top of the stack, it returns 2* factorial(1). The activation record for factorial(2) is removed from the stack. Now the animation shows that factorial(3) is on the top of the stack, it returns 3* factorial(2). The activation record for factorial(3) is removed from the stack.



```
 1  import java.util.Scanner;
 2
 3  public class ComputeFactorial {
 4    /** Main method */
 5    public static void main(String[] args) {
 6      // Create a Scanner
 7      Scanner input = new Scanner(System.in);
 8      System.out.print("Enter a non-negative integer: ");
 9      int n = input.nextInt();
10
11      // Display factorial
12      System.out.println("Factorial of " + n + " is " + factorial(n));
13    }
14
15    /** Return the factorial for a specified number */
16    public static long factorial(int n) {
17      if (n == 0) // Base case
18        return 1;
19      else
20        return n * factorial(n - 1); // Recursive call
21    }
22  }
```

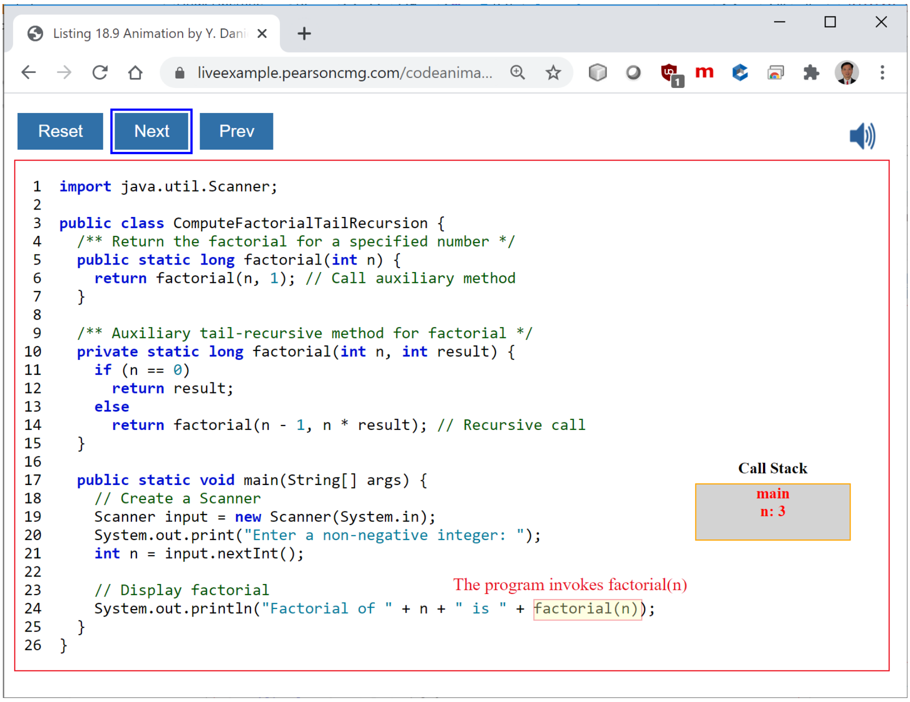
factorial(0) is invoked and pushed to the stack.

Figure 2: The activation record for factorial(0) is pushed to stack

Finally, when the main method exits, the animation shows that the call stack is empty. With the help of this animation, students can see how a recursive method is called and tracked using activation records in the call stack.

3 Tail Recursion Animation

To show how a tail recursion works, we created an animation for computing factorial using a tail recursive method. You can access the code animation for tail recursion from `https://liveexample.pearsoncmg.com/codeanimation/ComputeFactorialTailRecursion.html`, as shown in Figure 3. factorial(n, result) in lines 10-15 is a tail recursive method. The main method invokes factorial(n) in line 24. Suppose that n is entered as 3, the animation shows that factorial(n) invokes factorial(n, 1) in line 6. factorial(n, 1) is a tail recursive method. The animation shows that it invokes factorial(n – 1, n * result) in line 14. factorial(3, 1) invokes factorial(2, 3). factorial(2, 3) invokes factorial(1, 6). factorial(1, 6) invokes factorial(0, 6). Since factorial(n – 1, n * result) is a tail recursive, when a new factorial(2, 3) is invoked, the compiler does not create a new activation record. It will simply reuse the current activation record on the top of a stack for a tail recursive method. When the tail recursive method returns result in line 12, the animation shows that the activation record for the tail recursive method on the stack is removed, as shown in Figure 4.


Listing 18.9 Animation by Y. Dani
liveexample.pearsoncmg.com/codeanima...
Reset
Next
Prev

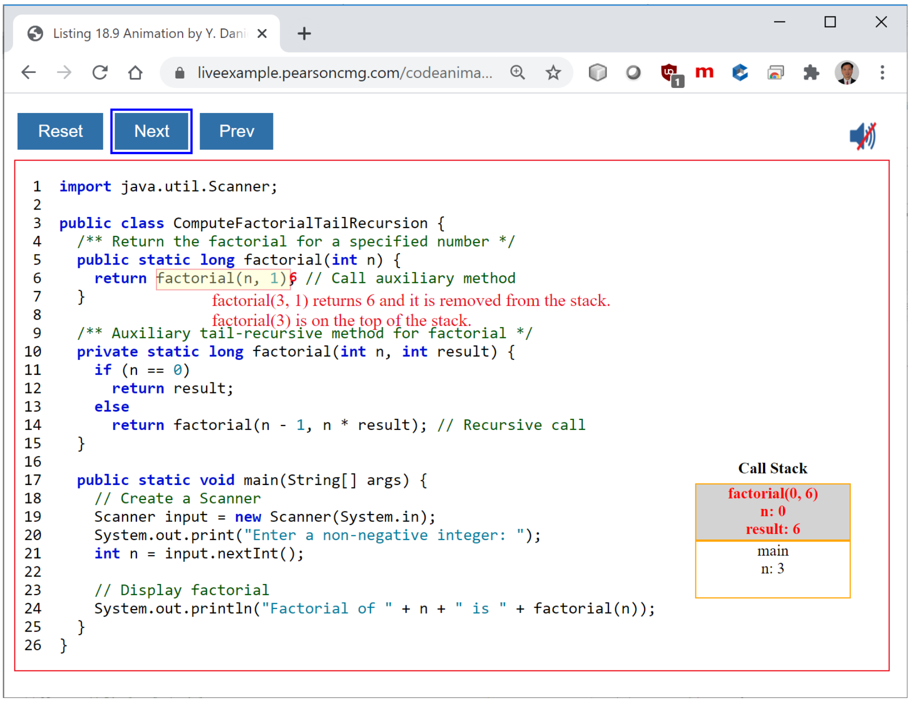
```
1  import java.util.Scanner;
2
3  public class ComputeFactorialTailRecursion {
4    /** Return the factorial for a specified number */
5    public static long factorial(int n) {
6      return factorial(n, 1); // Call auxiliary method
7    }
8
9    /** Auxiliary tail-recursive method for factorial */
10   private static long factorial(int n, int result) {
11     if (n == 0)
12       return result;
13     else
14       return factorial(n - 1, n * result); // Recursive call
15   }
16
17   public static void main(String[] args) {
18     // Create a Scanner
19     Scanner input = new Scanner(System.in);
20     System.out.print("Enter a non-negative integer: ");
21     int n = input.nextInt();
22
23     // Display factorial
24     System.out.println("Factorial of " + n + " is " + factorial(n));
25   }
26 }
```

Call Stack

main
n: 3

The program invokes factorial(n)

Figure 3: Code animation for tail recursion


Listing 18.9 Animation by Y. Dani
liveexample.pearsoncmg.com/codeanima...
Reset
Next
Prev

```java
import java.util.Scanner;

public class ComputeFactorialTailRecursion {
  /** Return the factorial for a specified number */
  public static long factorial(int n) {
    return factorial(n, 1)6 // Call auxiliary method
  }

  /** Auxiliary tail-recursive method for factorial */
  private static long factorial(int n, int result) {
    if (n == 0)
      return result;
    else
      return factorial(n - 1, n * result); // Recursive call
  }

  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a non-negative integer: ");
    int n = input.nextInt();

    // Display factorial
    System.out.println("Factorial of " + n + " is " + factorial(n));
  }
}
```

factorial(3, 1) returns 6 and it is removed from the stack.
factorial(3) is on the top of the stack.

Call Stack

factorial(0, 6)
n: 0
result: 6

main
n: 3

Figure 4: The activation record for the tail recursive method is removed when the method is returned

With the help of this animation, students can see why a tail recursive method is efficient in space and time, because the activation record is created only once for a tail recursive method.

4 Evaluation

The idea of creating code animations for recursion was suggested by an instructor who has used our code animation for non-recursive programs. We have created the code animation for recursion and tail recursion in the summer of 2018 for Java, C++, and Python. In the Fall 2018, we surveyed the students in a class of 9 students. We used a scale of 1 to 10 for answers, where 1 is poor and 10 is excellent. The result is as follows:

1. Does the ComputeFactorial.java animation help you understand how a recursive method is called using activation records? 8.66

2. Does the ComputeFactorialTailRecursion.java animation help you understand how a tail recursive method is called using activation records? 8.44
3. Do the animations for ComputeFactorialTailRecursion.java and ComputeFactorial.java help you understand why a tail recursive method is more efficient than a non-tail recursive method? 8.44

The survey shows that students agree strongly that the animations for recursion and tail recursion help them understand recursion and tail recursion.

We conducted further studies in the Spring 2021 in two classes. Right after we introduced recursion using ComputeFactorial and ComputeFactorialTailRecursion, we tested students in two classes with the following six questions:

Q1: To compute factorial(3), how many times is the factorial method invoked? Count invoking factorial(3) as the first time. (1 point. Answer: 4 times).

Q2; To compute factorial(3), how many times is the tail recursive factorial(n, result) method invoked? Count invoking factorial(3, 1) as the first time. (1 point. Answer: 4 times).

Q3: What is the maximum number of the activation records when running ComputeFactorial with input 3? Count invoking the main method as 1 activation record. (1 point. Answer: 5).

Q4: What is the maximum number of the activation records when running ComputeFactorial with input 4? Count invoking the main method as 1 activation record. (1 point. Answer: 6).

Q5: What is the maximum number of the activation records when running ComputeFactorialTailRecursion with input 3? Count invoking the main method as 1 activation record. (1 point. Answer: 3).

Q6: What is the maximum number of the activation records when running ComputeFactorialTailRecursion with input 4? Count invoking the main method as 1 activation record. (1 point. Answer: 3).

In Class 1, 17 students studied the materials using the code animation. In Class 2, 13 students studied the materials without using the code animation. Table 1 shows the result from the two classes. The table shows that the average on the preceding six questions is 0.696 for the class using code animation and 0.5 for the class without using code animation.

The test was given immediately after the lectures on recursion and tail recursion. So, most of the scores are low, because students were just introduced to recursion. The score for question Q5 is higher than the rest for Class 1, because we used the question as the example in the code animation for Class 1. Students have fresh memories of that question. Q1, Q3, and Q4 are on non-tail recursion in Table 1. Q2, Q5, and Q6 are on the tail recursion in bold in Table 1. Class 1 has a slightly better result than Class 2 on the non-tail

Table 1: Test Question Result from Two Classes (tail recursion in bold)

	Q1	**Q2**	Q3	Q4	**Q5**	**Q6**	Average
Class 1 (using animation)	0.765	**0.587**	0.647	0.647	**0.882**	**0.647**	0.696
Class 2 (without using animation)	0.538	**0.615**	0.692	0.615	**0.307**	**0.231**	0.5

recursion (Q1, Q3, and Q4) and has a much better result than Class 2 on tail recursion (Q5, Q6).

The test result shows that the performance for Class 1 is better than Class 2 on average. In particular, Class 1's performance is much better than Class 2's performance on tail recursion. The animation for recursion and tail recursion was used for Class 1. The animation helps students learn recursion and tail recursion.

5 Conclusions

This paper presented the code animations for recursion and tail recursion. These are the effective tools for helping students to learn how recursion and tail recursion work behind the scenes. Our future work is to establish a framework for creating code animation for recursive programs and create more code animation for recursive programs.

References

[1] Revel educator study assesses quiz, exam, and final course grades at central michigan university, Fall 2015. http://www.pearsoned.com/results/revel-educator-study-assesses-quiz-exam-final-course-grades-central-michigan-university/.

[2] Revel™ educator study observes homework and exam grades at university of louisiana, Spring 2016. http://www.pearsoned.com/results/revel-educator-study-observes-homework-exam-grades-university-louisiana/.

[3] Wanda Dann, Stephen Cooper, and Randy Pausch. Using visualization to teach novices recursion. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pages 109–112, 2001.

[4] Philip J Guo. Online Python tutor: embeddable web-based program visualization for CS education. In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 579–584, 2013.

[5] Y Daniel Liang. *REVEL™ for Introduction C++ Programming and Data Structures. 4e. ISBN-13: 978-0134669854.* Pearson Education, 2018.

[6] Y Daniel Liang. *REVEL™ for Introduction Python Programming and Data Structures. 2e. ISBN-13: 978-0135187753.* Pearson Education, 2018.

[7] Y Daniel Liang. Teaching and learning programming using code animation. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, pages 24–30, 2018.

[8] Y Daniel Liang. *REVEL™ for Introduction Java Programming and Data Structures. 12e. ISBN-13: 978-0135945476.* Pearson Education, 2020.

[9] Thomas L Naps and Brian Swander. An object-oriented approach to algorithm visualization—easy, extensible, and dynamic. In *Proceedings of the twenty-fifth SIGCSE symposium on Computer science education*, pages 46–50, 1994.

[10] E. Roberts. *Thinking Recursively with Java ISBN-13: 978-0471701460.* John Wiley & Sons, Inc., 2005.

[11] Cheng-Chih Wu, Nell B Dale, and Lowell J Bethel. Conceptual models and cognitive learning styles in teaching recursion. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 292–296, 1998.

A Machine Learning Approach to Understanding the Viability of Private 4-Year Higher-Education Institutions[*]

Kian L. Pokorny
Division of Computing
McKendree University
Lebanon, IL 62254
klpokorny@mckendree.edu

Abstract

For several years there have been warnings of an impending higher-education crisis in the United States. Clayton Christensen's 2013 suggestion that half of colleges will close in the next decade still looms [4]. His disruptive innovation theory [2] which considers the impact of online education coupled with the reduced number of college age students is now heightened by the COVID-19 pandemic. These factors all play a role in the economic stress of US higher education institutions. In this paper, we consider a machine learning approach to investigating trends in higher education that lead a school to close, be consolidated with another university, or for some reason to no longer exist. The goal is to understand overall trends and consider factors that indicate an unhealthy status of a school which precipitates its closure.

Using the IPEDS publicly available data, Private Not-For-Profit 4-year institutions are examined. The data is sparse in its dimensional space and highly imbalanced in the classes. Six learning algorithms are applied, and results are presented. Using an ensemble of four of the algorithms are shown to provide the best results.

[*]Copyright ©2021 by the Consortium for Computing Sciences in Colleges. Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the CCSC copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Consortium for Computing Sciences in Colleges. To copy otherwise, or to republish, requires a fee and/or specific permission.

1 Introduction

College enrollment has declined for the last 10 years in the United States. In 2011 there were 21,011,000 students enrolled in U.S. higher education. By spring of 2020, that number had fallen to approximately 19,744,000 [3]. The decline in the number of "college age" students is expected to continue. Higher Education in the United States is a competitive business. Schools compete for a declining number of students and their tuition dollars.

Reduced number of college age students is not the only challenge U.S. universities face. Due to the COVID-19 pandemic it is estimated that college enrollment dropped by 2.5% from fall 2019 to fall 2020 [1]. Vedder [5] suggests seven reasons for the decline of American universities. Declining enrollments, rising costs of attendance, lower cost vocational training, less government support for basic research, fewer traditional college age (18-22) people, foreign universities' higher prestige and lower cost, and a view of US universities as being politically left are all suggested as contributing factors to the long-term decline of U.S. universities [5].

In the book, The College Stress Test: Tracking Institutional Futures Across a Crowded Market [6] it is recognized that stratifying across sectors (Public 4-year, Private Not-for-Profit 4-year, Private For-Profit 4-year, and Public 2-Year) is appropriate. Zemsky et.al. [6] found different but similar factors play a role in scoring the financial health of schools in each sector. Public universities rarely fail because state and federal government support helps when enrollments decline. Thus, for calculating a stress score for public institutions, state appropriations are used as part of the financial health of the institution. Whereas Private For-Profit institutions are tuition dependent. These institutions have different "missions" than Private Not-For-Profits, and they can come and go quickly. Private Not-For-Profit institutions can be self-reliant or may depend on church support. For these institutions endowment plays a role in calculating the financial component of the stress score.

In this study we identify factors that indicate a university's potential closure, merger, or consolidation. For this paper all these situations are classified as a closure (closed). All other schools are classified as open. We then utilize several algorithms to predict a school closure. In the section 2, we consider the data including a description of the data set, data engineering, and feature creation. Section 3 describes the analysis and results achieved. Section 4 provides conclusions of this initial study and directions of further investigations.

2 Data Source

Here the focus is on Private Not-for-Profit 4-year higher education institutions. All publicly available data from the Integrated Postsecondary Education Data System (IPEDS) for the years 2004 through 2018 is obtained. Each year has approximately 40 tables containing approximately 1800 variables. Private Not-for-Profit 4-Year schools are extracted for this study. The data includes 933 Private Not-for-Profit 4-year institutions; 39 of these institutions closed between 2004 and 2018, 8 closed after 2018, and 885 remain open at the time of writing this paper. These schools represent 2,951,201 undergraduate students in 2018.

It is determined that the size of the student body at an institution plays an important role in determining success of the institution. Small schools that are tuition dependent struggle financially when faced with declining enrollments. We stratify our schools on size and address the problem with schools that had undergraduate enrollments of 3500 or less, two years prior to closing. This is based on 2017 enrollments for schools that remained open at the end of our study period. The resulting dataset contains 717 schools. 37 of these schools closed between 2006 and 20018, and 8 of the schools closed after the study period, in 2019 or 2020.

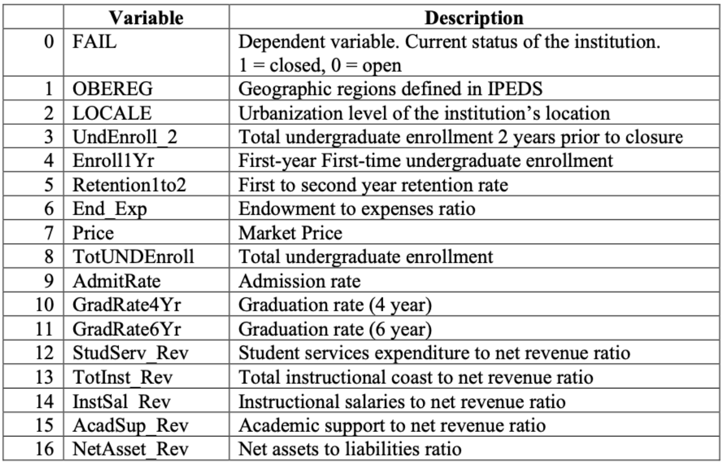
The data is right aligned so the right-most column represents the final year open for schools that closed prior to 2019, and each column to the left represents years prior to closure. For schools that remain open the far-right column is data for 2018. We then realign the data for the 8 schools that closed after 2018 to the left so that the dataset provides a consistent representation for all schools that closed. After properly aligning all data, a total of 45 schools that closed and 672 schools remaining open are obtained. With 6.3% of the schools closed the dataset is highly imbalanced.

2.1 Features

Modeling the health of private not-for-profit universities often includes data in several broad categories; enrollment, retention rates which are highly correlated with graduation rates, financial stability including endowments and long-term investments, and tuition rates. It is also recognized that trends in the features over time are important. That is, a specific enrollment number provides different information than increasing or decreasing enrollment over several years.

After an initial investigation of the available data the features that are chosen for the analysis are listed in Table 1. Several features are aggregates of IPEDS variables. We use ratios of expenses to revenue which provide a normalized measure across schools. Note that all financial data is adjusted using the GDP deflator. The features End_Exp and Price are inspired by Zemsky et. al. [6]. Price is calculated as the weighted average of the difference between the advertised price and the average amount of institutional aid, multiplied by the percentage of students receiving the aid, and advertised price multiplied by the percentage not receiving institutional aid. This is the price that students pay, on average, after the discount rate is applied.

Table 1: Model Features Used



	Variable	**Description**
0	FAIL	Dependent variable. Current status of the institution. 1 = closed, 0 = open
1	OBEREG	Geographic regions defined in IPEDS
2	LOCALE	Urbanization level of the institution's location
3	UndEnroll_2	Total undergraduate enrollment 2 years prior to closure
4	Enroll1Yr	First-year First-time undergraduate enrollment
5	Retention1to2	First to second year retention rate
6	End_Exp	Endowment to expenses ratio
7	Price	Market Price
8	TotUNDEnroll	Total undergraduate enrollment
9	AdmitRate	Admission rate
10	GradRate4Yr	Graduation rate (4 year)
11	GradRate6Yr	Graduation rate (6 year)
12	StudServ_Rev	Student services expenditure to net revenue ratio
13	TotInst_Rev	Total instructional coast to net revenue ratio
14	InstSal_Rev	Instructional salaries to net revenue ratio
15	AcadSup_Rev	Academic support to net revenue ratio
16	NetAsset_Rev	Net assets to liabilities ratio

2.2 Data Engineering

Data from five to two years prior to closure is used. Our interest is in predicting a closure while there is still time to intervene. Thus, the prediction models using data from two or more years prior to closure is prudent. The rate of change in the variable from these three years is calculated and used as our features in the models. For each of the variables 4 through 16 the slope of the line is calculated from 5 years prior to 2 years prior, which is used as the feature in the models.

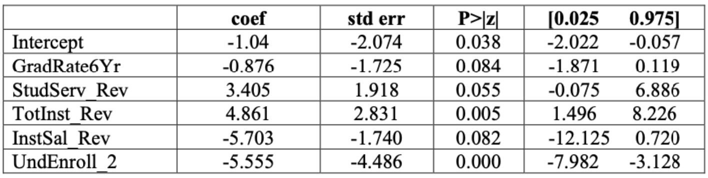
Current enrollment is particularly valuable in determining the health of an institution. Very small schools have trouble remaining viable when small drops in enrollment numbers occur. Because current enrollment is very important, Total undergraduate enrollment (UndEnroll_2) from 2 years prior to closure (2017 for open schools) is used as another feature. This feature is normalized by dividing by the largest total undergraduate enrollment value.

3 Method

The data is highly imbalanced and sparse in the feature space. This creates many challenges in obtaining quality classification models. Thus, several different types of learning algorithms are applied to the data. The six learning algorithms Logistic regression, Logistic Regression with Elastic Net, Random Forest, XGBoost, Naïve Bayes, and K-Nearest Neighbors (KNN) are used to classify the schools as either 1 = closed, or 0= open. The output of KNN is 0 or 1. The output of the other algorithms is a probability of closure.

Recursive feature elimination is used with logistic regression to help understand feature significance. Table 2 shows the most significant features based on p-values from the logistic regression.

Table 2: Logistic Regression Features Significance



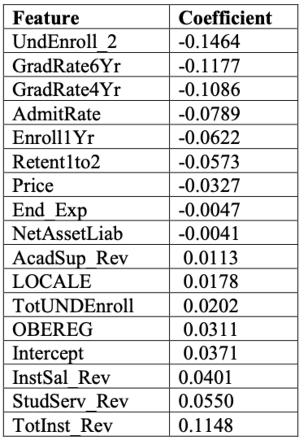
	coef	std err	P>|z|	[0.025	0.975]
Intercept	-1.04	-2.074	0.038	-2.022	-0.057
GradRate6Yr	-0.876	-1.725	0.084	-1.871	0.119
StudServ_Rev	3.405	1.918	0.055	-0.075	6.886
TotInst_Rev	4.861	2.831	0.005	1.496	8.226
InstSal_Rev	-5.703	-1.740	0.082	-12.125	0.720
UndEnroll_2	-5.555	-4.486	0.000	-7.982	-3.128

With logistic regression we normally calculate the odds ratio for an accurate interpretation of the coefficients. However, we can state that because of the negative coefficient, as UndEnroll_2 increases the probability of failure decreases. Apart from UndEnroll_2 the features are rates of change obtained from calculating the slope of the line from five to two years prior to the last year of the study. Much care in the interpretation of these features needs to be taken. For example, as InstSal_Rev increases the probability of closure decrease. That is, as the rate of change in the ratio of Institutional salaries to net revenue from 5 years prior to 2 years prior increases, the probability of closure decreases. This rate of change in the ratio can be caused by either an increase in instructional salaries or a decrease in revenues across those years.

The next algorithm is an elastic net with logistic regression which combines L1 and L2 regularization for feature elimination. These regularization techniques move the coefficients of unimportant features toward zero. Features in the middle of Table 3 have coefficients closest to zero. Eliminating features with coefficient values smaller than the absolute value of the intercept's coefficient (0.037) yields a set of features that includes the features from the previous logistic regression, indicating some agreement among these two approaches.

Decision tree algorithms perform automatic features selection. Here we used random forest and XGBoost algorithms. XGBoost outperforms random forest in precision. The feature importance for the Random Forest algorithm

Table 3: Results for Logistic Regression with Elastic Net



Feature	Coefficient
UndEnroll_2	-0.1464
GradRate6Yr	-0.1177
GradRate4Yr	-0.1086
AdmitRate	-0.0789
Enroll1Yr	-0.0622
Retent1to2	-0.0573
Price	-0.0327
End_Exp	-0.0047
NetAssetLiab	-0.0041
AcadSup_Rev	0.0113
LOCALE	0.0178
TotUNDEnroll	0.0202
OBEREG	0.0311
Intercept	0.0371
InstSal_Rev	0.0401
StudServ_Rev	0.0550
TotInst_Rev	0.1148

is given in Figure 1 and XGBoost in Figure 2. Here we see similar feature importance for the decision tree-based algorithms. Note that UndEnroll_2 is most significant indicator in these models as well.

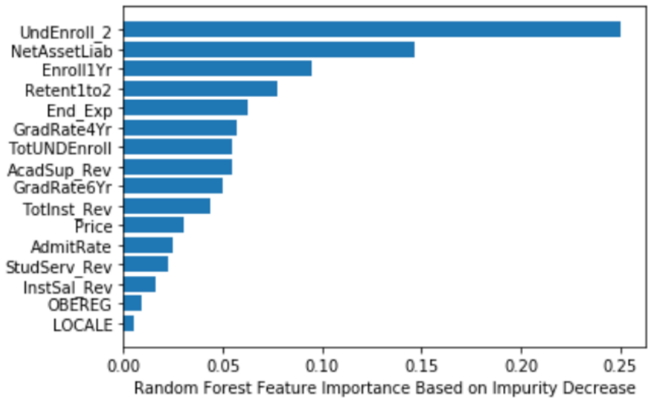


Figure 1: Random Forest Feature Importance

The Naïve Bayes algorithm implemented with all features listed in Table 1 provides relatively good results. After experimenting with various feature combinations, K-Nearest Neighbors (KNN) is implemented with the features UndEnroll_2, TotInst_Rev, GradRate6Yr, and InstSal_Rev.

Because the data is imbalanced and we are concerned with the minority

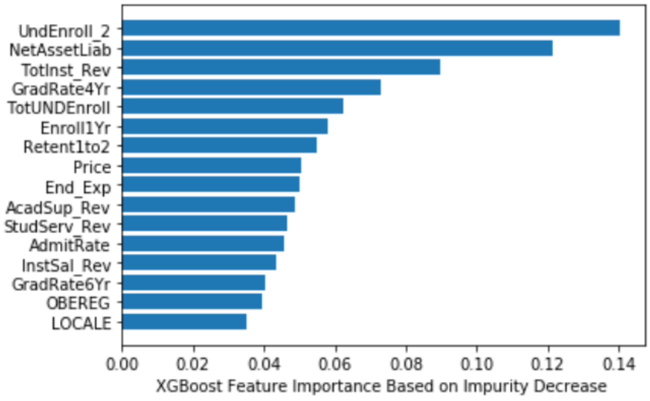


Figure 2: XGBoost Feature Importance

class (closed=1), recall, precision, and F1-scores are reported for each method. Table 4 lists the results.

Table 4: Results of the Algorithms

	Recall	**Precision**	**F1-Score**
Logistic regression	0.2000	0.5000	0.2857
ElasticNet LR	0.7000	0.1489	0.2456
Random Forest	0.5000	0.2778	0.3571
XGBoost	0.5000	0.6250	0.5556
Naïve Bayes	0.7000	0.3043	0.4242
K-Nearest Neighbors	0.5000	0.3125	0.3846
Ensemble Method	0.9000	0.4500	0.6000

Finally, an ensemble of the Random Forest, XGBoost, Naïve Bayes, and KNN algorithms is built. The output of each of these algorithms is weighted at 0.34 and summed to produce the ensemble value. All records with a value >0.499 are classified as 1=closed. This is approximately a majority vote method. The last row in Table 4 provides results for the ensemble. A recall of 0.90 indicates that 90% of the closed schools are correctly predicted. A precision of 0.45 indicates that 55% of the schools predicted to close, have not closed. The ensemble method has the highest recall and F1-Score.

4 Conclusions

Higher education institutions in the United States face many challenges. In this article we investigate data from IPEDS to provide some insight into continued viability of small Private Not-For-Profit 4-year schools. The data is limited in number, imbalanced, and sparse in the feature space making prediction difficult. However, we are able to determine features that indicate if a school may be facing possible closure. Finally, an ensemble of models generated from random forest, XGBoost, Naïve Bayes, and KNN algorithms were used to obtain good results. The result from the ensemble shows a marked improvement from the individual algorithms. In addition to the broad categories mentioned in section 2.1, the models indicate that the ratios of total instruction cost and instruction salaries to net revenue play a role in the viability of these schools.

A future study investigates the schools predicted to be closed by these models that are currently open. One study in progress includes other data sources to complement the IPEDS data and to gain deeper insight into the health and viability of higher education institutions in the U.S.

References

[1] M. S. Amour. Inside higher ed: Few positives in final fall enrollment numbers. `https://www.insidehighered.com/news/2020/12/17/final-fall-enrollment-numbers-show-pandemics-full-impact##:~:text=The%20final%20word%20is%20that,about%20400%2C000%20students%20this%20fall`.

[2] Clayton M Christensen and Henry J Eyring. *The innovative university: Changing the DNA of higher education from the inside out.* John Wiley & Sons, 2011.

[3] National Center for Educational Statistics. Digest of education statistics. `https://nces.ed.gov/programs/digest/d19/tables/dt19_105.30.asp`.

[4] D. Lederman. Clay christensen, doubling down. `https://www.insidehighered.com/digital-learning/article/2017/04/28/clay-christensen-sticks-predictions-massive-college-closures`.

[5] R. Vedder. The decline of the american university. `https://www.forbes.com/sites/richardvedder/2020/06/22/the-decline-of-the-american-university/?sh=76e58f2921f0`.

[6] Robert Zemsky, Susan Shaman, and Susan Campbell Baldridge. *The college stress test: Tracking institutional futures across a crowded market.* JHU Press, 2020.

Python Data Structures for Java Programmers[*]

Conference Tutorial

Bill Nicholson , Isaiah Dicrostoforo
University of Cincinnati
nicholdw@ucmail.uc.edu

This tutorial introduces Python data structures such as Lists, Sets, Tuples, Strings, and dictionaries. It draws connections to data structures in Java. Attendees will learn how Python data structures differ from and overlap with similar data structures in Java.

Attendees can participate using open-source tools such as JupyterLab notebooks, the python REPL shell, and online Python fiddle web sites. All materials presented will be available during the tutorial in a public GitHub repository: `https://github.com/nicomp42/CCSCMidwest2021Tutorial`

[*]Copyright is held by the author/owner.

A Tutorial on Flutter

Conference Tutorial

Michael P. Rogers
Department of Computer Science
University of Wisconsin Oshkosh
Oshkosh, WI 54901
mprogers@mac.com

Flutter is an open-source, cross-platform development kit from Google that allows developers to create apps for iOS, Android, and other platforms (web and desktop), from a single codebase.

Flutter takes a modern approach to app development. The SDK is based on a declarative UI paradigm that is widely used in the workforce. Apps are written in Dart, a strongly typed, easy-to-learn language that reinforces good coding habits. Development can be done in Visual Studio Code, which students appreciate for its flexibility and simple interface.

Flutter allows instructors to sidestep the issue of what platform to target: students with any smart phone and operating system can now participate, sparing the university from having to provide a development environment. It provides a vehicle and means for introducing declarative UI design into the curriculum, so that when students do see this in industry, they will be prepared for it.

This technology could be used in a mobile computing class, a software engineering/capstone class, or any other situation where students need to develop for more than one platform.

In this 90-minute tutorial, participants will be introduced to the language, tools, and paradigms that drive Flutter, and provided with tips, based on the presenter's experience, on how best to incorporate this SDK into a mobile class.

Using GitHub Classroom for Assignment Management and Automated Feedback

Conference Tutorial

Zachary Kurmas
Department of Computer Science
Information Systems
Grand Valley State University
Allendale, MI 49401
`kurmasz@gvsu.edu`

GitHub Classroom is an instructor-facing tool that automates the creation and management of GitHub repositories for student assignments. In addition, instructors can optionally utilize GitHub Actions to provide automated feedback to students.

This tutorial/workshop will

- Describe several use cases for GitHub Classroom (i.e., the types of assignments that can potentially benefit from the tool)
- Demonstrate how to set up GitHub classroom (including creating a GitHub organization)
- Demonstrate basic GitHub Classroom usage including
  - Configuring starter code as a template repository
  - Setting up an assignment in GitHub Classroom
  - Monitoring student progress and helping students
  - Reviewing student submissions and providing feedback
- Demonstrate how to use GitHub Actions to provide automated feedback to students
- Demonstrate how to write scripts that use the GitHub CLI to automate the process of managing student assignments (e.g., cloning all repositories for a given assignment)

GitHub Classroom is a web interface provided by GitHub that automates the process of creating separate GitHub repositories for each student (or group of students) completing an assignment. Instructors create a GitHub repository containing instructions and/or "starter code" then set it as a "template" repository. GitHub Classroom then generates a unique URL that can be shared with students. Upon clicking on that link, GitHub will clone the template repository and give the new repository unique name that is a combination of the student's github account name and the assignment name.

The repositories created by GitHub Classroom are owned by the instructor (or, more typically by a GitHub Organization administered by the instructor). As a result, (1) instructors can ensure that the repositories remain private; and (2) instructors can see and manage the complete list of repositories. (In contrast, if students were independantly forking the starter code, then the only way the instructor would know about the repos is if the students sent an invitation to the instructor.) Because all the repositories automatically appear in an account controlled by the instructor, he or she can automate the process of checking out the code (e.g., to evaluate the assignment).

The student GitHub repositories have access to GitHub Actions which can be used to provide automated feedback. For example, the instructor can configure an Action that will run unit tests whenever code is committed. The GitHub Classroom web page displays the list of all repositories for a given assignment and shows whether the most recently executed GitHub Action was successful.

The relatively new GitHub CLI (Command Line Interface) provides a number of tools that make it easy to manage a group of repositories. Those features not provided directly by the CLI can easily be added by using the CLI to make GitHub API calls.

IndianaComputes! Views of a K-12 Professional Development Program[*]

Panel Discussion

Karen M. Morris
IndianaComputes! Project Manager
University of Notre Dame
Notre Dame, IN 46556
morris.3@nd.edu

IndianaComputes! is a collaboration of Computer Science and STEM faculty from 12 Indiana public and private colleges and universities to provide computer science professional development to K-12 teachers. Funded with a contract from the Indiana Department of Education, this professional development program started in June 2020 and has been developed to enable schools to meet the state law that requires public schools to include Computer Science in their science curriculum (K-8 schools) and offer a Computer Science course each year to students (high schools) by July 1, 2021. This professional development program is a combination of asynchronous online content delivery and synchronous sessions (Zoom or in-person) coordinated and implemented by Computer Science faculty. This professional development design differs significantly from traditional teacher professional development programs in that it is year-long, includes both content knowledge as well as pedagogy, identifies foundational information all teachers should know as well as grade-banded specific fluencies, and uses only university faculty for instruction.

The Panel Discussion will provide the audience a vision for a long-term professional development program from both the K-12 participant and the Computer Science faculty perspectives. A moderator will facilitate the discussion between four Computer Science Faculty and K-12 teacher pairs about the IndianaComputes! professional development program. Computer Science Faculty will be our Synchronous Session providers representing the broad spectrum of colleges and universities involved in this project. K-12 teachers will represent the grade bands: K-2, 3-5, 6-8, and 9-12.

[*]Copyright is held by the author/owner.

Questions for Discussion will include:

1. What encouraged you to participate in the IndianaComputes! professional development program?
2. What is your perspective of an online Computer science professional development program? What would you say the best and worst parts are?
3. What was the most challenging aspect of learning/teaching Computer Science concepts?
4. How did your interaction with your K-12 Teacher/CS Faculty member support you? What else might you have wanted?
5. How has this impacted your work in the classroom?
6. Audience Questions

Computer Science Faculty will most likely attend this session in-person, while K-12 teachers will most likely attend online. Panelists will include (but may be changed due to availability):

Higher Education	**K-12 Teacher**
Karen M. Morris - University of Notre Dame (moderator)	K-2 Teacher: Gaisha Williams- McCullough Academy
Michele Roberts – Indiana University – Bloomington	3-5 Teacher: Sherry Evert - Indiana Horizon Academy
David Largent – Ball State University	6 -8 Teacher: Sister Mary Jacqueline Oranye- Stonybrook Intermediate & Middle School
Jeff and Devon Kinne – Indiana State University	9 – 12 Teacher: Summer Ehresmann – Center Grove High School
Hossein Hakimzadeh – Indiana University – South Bend	

Teaching Heterogeneous Parallel Programming With CUDA[*]

Conference Workshop

David P. Bunde
Computer Science
Knox College, Galesburg, IL 61401
dbunde@knox.edu

CS faculty have spent the last several years adding parallel computing to their curricula since essentially all processors sold today have multiple cores. A typical target system is a multicore processor with identical cores. This is the configuration for most current desktop and laptop systems, but the technology continues to evolve and systems are incorporating heterogeneity, with cores or varying size and specialized processing elements to optimize performance and power. In this hands-on workshop, I will present modules for teaching about computational and memory heterogeneity with CUDA, a common approach to graphics processing unit (GPU) programming. Then attendees will have time to work on the modules themselves as well as to begin planning how the modules could be used in their courses. Importantly, these modules can be done using Google Colab, a cloud environment that provides free access to GPUs without requiring the purchase or installation of hardware.

The first module introduces CUDA to students with no prior exposure to CUDA and limited experience with parallel computing. It highlights the main features of GPU programming, which requires the transfer of data to/from the GPU and features a SIMD model of computing, where the same operations are performed on all the data. This module demonstrates both the potential of heterogeneous computing and the greater effort required to realize that potential. The application is image processing, giving students a motivating example with plenty of potential parallelism to unlock.

The second module builds on the first and highlights the heterogeneous memory types on a GPU. GPUs have several kinds of memory, each with different performance characteristics. In this module, block-level shared memory is used as a programmer-managed cache, allowing a significant performance

[*]Copyright is held by the author/owner.

improvement. Thus, the module both builds on a previous introduction to CUDA and reinforces general lessons about the memory hierarchy.

Each module fits within a few days of class time in a standard course. They are posted online with slides, laboratory activities, and potential homework assignments (`https://github.com/TeachingUndergradsCHC/modules/`). This repository also contains other modules teaching heterogeneous parallel programming developed by our project. These will also be briefly introduced during the workshop.

An Introduction to Tableau as a Data Visualization Tool[*]

Conference Workshop

Mary Jo Geise
Department of Computer Science
University of Findlay
Findlay, OH 45840
`geise@findlay.edu`

The connection between data science and computer science is high! Many of our computer science graduates will find jobs that require data analytic skills as there is currently a huge shortage of data scientists in today's workforce. Of the many skills needed to be a data scientist, being able to visualize data and tell a data story are considered critical skills.

Tableau is one of the leading products for business intelligence, analytics, and data visualizations. This workshop will introduce participants to various features of Tableau utilizing its drag-and-drop functionality. Workshop attendees will become familiar with basic Tableau terminology and concepts. Through a series of hands-on activities, participants will learn how to make a variety of meaningful visualizations, how to create dashboards from these visualizations, and how to create data stories from dashboards and worksheets. Lastly, those attending the workshop will learn how the Computer Science Department at the University of Findlay has incorporated Tableau into their curriculum.

Participants of this workshop will need a laptop with Internet access. Tableau will need to be installed and directions will be provided to registered participants prior to the conference on how to download a free trial version of Tableau. No previous knowledge of Tableau is required.

[*]Copyright is held by the author/owner.

Building Regional Community for Computing Education Graduate Students[*]

Conference Workshop

Morgan M. Fong, Max Fowler, Seth Poulsen, Vidushi Ojha, and Geoffrey L. Herman
Computer Science
University of Illinois, Urbana-Champaign
Urbana, IL 61801
{mmfong2,mfowler5,sethp3,vojha3,glherman}@illinois.edu

1 Extended Abstract

This workshop aims to provide a safe space for computing education graduate students to build community with regional peers, talk about graduate student life, and discuss and receive feedback on current research projects.

Although the number of graduate students in computing education research continues to grow, students' connections are often limited to a handful of graduate students and a couple of faculty at their own institutions. Building a community with local peers strengthens networks and enables cross-pollination of ideas and feedback that is beneficial for everyone. These stronger networks can lead to productive research collaboration. Additionally, having a dedicated space for rough, initial research ideas is important for graduate students of all stages in order to build confidence and get early feedback in a low-stakes environment. Finally, establishing strong regional connections will support the growth of computing education research in the Midwest.

Current graduate students at any stage are welcome to attend. Due to the nature of discussions, faculty will not be invited. We plan on having 20 participants (not including the facilitators), using our network to recruit from Ball State University, Indiana University Bloomington, University of Southern Indiana, Purdue University, University of Kansas, University of Nebraska-Lincoln, Michigan State University, University of Michigan, and Ohio State University. We are working with SageFox Consulting Group, who have an active research project on our target audience, to help recruit graduate students who may be interested.

[*]Copyright is held by the author/owner.

2 Anticipated Agenda

The workshop is organized around the three main themes above: building community with regional peers, talking about graduate student life, and discussing and receiving feedback on current research projects. As a result, there are two main activities for participants outlined below: introductions and discussion of graduate student life in small groups, and a poster session. All activities will take place in small groups to allow for newer students to present their work in a lower-stakes environment and to enable more equal participation.

0:00	Welcome
0:05	Small Group Introductions I (Groups of 4-5): Participants will introduce their name, pronouns, institution, year, advisor (if applicable), and what they hope to gain from the workshop
0:20	Small Group Discussion of Graduate Student Life I: Sample conversation starters include getting started with research, finding an advisor, maintaining work-life balance
0:40	Shuffle Small Groups
0:45	Small Group Introductions II: Same as Small Group Introductions I
1:00	Small Group Discussion of Graduate Student Life II: Same as Small Group Discussion of Graduate Student Life I
1:20	10 minute break
1:30	Poster Session in Small Groups: Participants will submit a poster to the organizers prior to the workshop. The poster should be formatted as one Google slide that includes at least the participant's name, institution, and contact information. During the workshop, each participant will have 5 minutes to present their poster, followed by 10 minutes of small group Q&A. After the workshop, the organizers will share the posters with all participants.
2:50	Debrief
3:00	End of Workshop

Aspects of US-China Competition May Motivate Students*

Work In Progress

Pradip Peter Dey and Bhaskar Raj Sinha
Department of Engineering and Computing
National University
San Diego, CA 92123
{pdey,bsinha}@nu.edu

Certain aspects of the US-China competition in technology may motivate students for majoring in computer science and related areas. On March 5, 2021, CNBC reported that "In its 14th five-year plan, China laid out seven technology areas it will focus research on including artificial intelligence, quantum computing, semiconductors . . ." [1]. In response, the US Senate adopted a bipartisan bill, the US Innovation and Competition Act of 2021 on July 6, 2021 [2, 3, 4].

Should academia pay attention to these developments? What are the best ways to deal with aspects of the US-China competition in learning environments? How do we prepare our students for these areas? US universities, colleges, NASA, and other organizations have played important roles in developing innovative technologies and their applications. Millions of Americans were inspired by President John F. Kennedy's speech at Rice University on September 12, 1962: "We choose to go to the Moon in this decade and do the other things, not because they are easy, but because they are hard, because that goal will serve to organize and measure the best of our energies and skills, ...". The goal, Kennedy mentioned, was achieved in the context of another competition between the USA and the Soviet Union.

This research analyzes different aspects of the US-China competition, and suggests ways to motivate students that may require changes in academia in order to prepare students for the 21st century workforce. We are optimistic about the positive outcomes from ethical participation of students, researchers, scientists, and professionals in the peaceful US-China competition because these participants would be able to take opportunities for making aspiring contributions towards innovations in science and engineering with encouraging effects

*Copyright is held by the author/owner.

on global developments. One of the consequences of the peaceful competition would be a new wave of worldwide prosperity and sustainable development. We hope that conflicts, wars, and other destructive activities may be avoided if mutually beneficial culture, understanding and rules of completion are practiced.

References

[1] A. Kharpal. In battle with U.S., China to focus on 7 'frontier' technologies from chips to brain-computer fusion. https://www.cnbc.com/2021/03/05/china-to-focus-on-frontier-tech-from-chips-to-quantum-computing.html.

[2] Tom Lee and Juan Londono. The United States Innovation and Competition Act (USICA): A primer. https://www.americanactionforum.org/insight/the-united-states-innovation-and-competition-act-usica-a-primer/.

[3] Wikipedia. United States Innovation and Competition Act. https://en.wikipedia.org/wiki/United_States_Innovation_and_Competition_Act.

[4] Patricia Zengerle and David Brunnstrom. Details of sweeping effort to counter China emerge in U.S. senate. https://www.usnews.com/news/us/articles/2021-04-08/us-senate-panel-to-to-consider-major-china-competition-bill-on-april-14-source.

Developing a Cross-Platform Mobile Course Using a Multi-Paradigm Library[*]

Work In Progress

Alisa Neeman
Muskingum University
New Concord, OH 43762
`aneeman@muskingum.edu`

Cross-platform mobile development enables companies to develop one single application that runs on multiple platforms, such as Android and iPhone. Using web-based languages is one way to accomplish this goal. This talk covers developing a Mobile course using the React JavaScript Library for cross-platform apps. React JavaScript apps have imperative, declarative, functional, object oriented, markup, and scripting language features. Although React is a JavaScript library, it has its own compiler that enables syntax extensions. The talk includes discussions of approaches for teaching about React's unique features such as: (1) Classes and functions that are render-able components, with a mixture of HTML, CSS, and React's JavaScript extensions, (2) a new syntax for Document Object Model event handling with functional programming behavior, (3) differences between modifiable state ("state") and non-modifiable state ("properties") for a component, and (4) asynchronous data storage and access. Course content could also potentially be used to illustrate concepts for Programming Languages and Operating Systems courses.

[*]Copyright is held by the author/owner.

IndianaComputes! a K-12 Professional Development Program*

Work In Progress

Karen M. Morris
IndianaComputes! Project Manager
University of Notre Dame
Notre Dame, IN 46556
morris.3@nd.edu

IndianaComputes! is a collaboration of Computer Science and STEM faculty from 12 Indiana public and private colleges and universities to provide Computer Science professional development to K-12 teachers. Funded with a contract from the Indiana Dept. of Education, this professional development program started in June 2020 in order to enable schools to meet the state law that requires public schools to include Computer Science in the science curriculum (K-8 schools) and offer a Computer Science course each year to students (high schools) by July 1, 2021. This professional development program is a combination of asynchronous online content delivery and synchronous sessions (Zoom or in-person) coordinated and implemented by Computer Science faculty. This professional development design differs significantly from traditional teacher professional development programs in that it is year-long, includes both content knowledge as well as pedagogy, identifies foundational information all teachers should know as well as grade-banded specific fluencies, and uses only university faculty for instruction. For this session, we will share results from the first program year, and early findings for the second program year, which includes the incorporation of a faculty-mentored teacher coaching model.

*Copyright is held by the author/owner.

Can OneUp Gamified Challenges Boost Undergrad Student Motivation Plus Engagement and Supplement Learning in An Online Introductory Cybersecurity Course?[*]

Work In Progress

Ankur Chattopadhyay, Meghyn Winslow, Momoka Kinder
Computer Science Department
Northern Kentucky University
Highland Heights, KY 41099
{chattopada1,goodridgem,kinderm1}@nku.edu

Motivating undergraduate students for active participation and consistent performance in online courses on introductory cybersecurity topics can be challenging. Prior literature shows research studies on using gamification tools for increasing student motivation and engagement in computer science (CS) courses. OneUp is one such gamification tool that utilizes proven game design principles, in the form of digital badge-based incentive driven online gamified challenges, which motivate students to work on additional course topic challenges, thus resulting in increased engagement and learning. However, even though there are research studies on boosting student motivation and engagement using OneUp in CS courses, there is limited research on the benefits of using OneUp in introductory cybersecurity classes, specifically at the undergraduate level. This work-in-progress research performs a preliminary study to determine OneUp's impact as a supplementary aid in an online introductory undergraduate course on cybersecurity fundamentals. We build an additional OneUp layer of gamified challenges for this particular online class to support the learning process, to supplement the learning environment, and to motivate students for working on beginners' cybersecurity topic-based challenges. We develop these supplemental OneUp-based gamified learning components to

[*]Copyright is held by the author/owner.

provide students an incentivized opportunity for engaging beyond the regular course assignments through optional practice and self- testing via online OneUp exercises. We study the impact of utilizing OneUp for this course by comparing the class performances of students, who practiced using the OneUp challenges created by us, with the course performances of the students, who did not participate in OneUp. We analyze the initial learner data, collected by surveying the OneUp participants, to determine OneUp's potential for enhancing student motivation, engagement and performance in this class. This initial research study is a timely intervention for improving overall student experiences in online introductory cybersecurity classes amidst the current challenges posed by classes going online due to the COVID-19 pandemic. Looking ahead, we plan to continue our OneUp experimental studies in future online introductory cybersecurity courses, so that we can collect more research data for doing further analysis to investigate the impact and efficacy of OneUp as a supplementary aid in uplifting the motivation, engagement and performance of remote students in a distance learning environment.

Practical Program Verification with DAFNY[*]

Work In Progress

Ramachandra B. Abhyankar, Robert W. Sternfeld
Mathematics and Computer Science
Indiana State University
Terre Haute, Indiana 47809
`{R.B.Abhyankar,Robert.Sternfeld}@indstate.edu`

It is well-accepted that the single most important requirement of a program is “correctness.” Program verification is the process of establishing the correctness of a program. Despite its importance, the “Correctness Problem in Computer Science” has remained unsolved, and errors in programs continue to occur, with often serious consequences, in an increasingly computerized world.

The lack of tool support has required a “pencil and paper” approach to verification, making it impractical. This has resulted in the use of “testing” to increase the confidence in the quality of the software. It is widely accepted that testing cannot establish correctness of programs. Testing can only expose the presence of errors.

We deal only with the verification of imperative programs; this requires the use of a formalism like Hoare Logic [3] as well as the ability to perform logical reasoning in propositional and predicate logic, and in theories of equality, integers, etc. Theorem provers like OTTER, Isabelle/HOL, Model Builders like MACE 2, and SMT and SAT solvers, have been employed in the task of program verification.

Automation of Hoare logic has been attempted in systems like Jape [4] and HAHA [2]. The development of “Program Verifiers” such as DAFNY [1] represents a major advance in Verification Technology, which provides a seamless integration of many of these tools. Using DAFNY, programs can be verified without users having to leave the DAFNY environment. Tools like DAFNY are making verification practical. Without a “specification”, one cannot talk about “correctness.” A specification is often stated as a contract, consisting of a precondition and a post-condition. “Total Correctness” is defined as follows:

[*]Copyright is held by the author/owner.

If the program begins execution in a state that satisfies the precondition, then it is guaranteed to terminate in a finite amount of time, in a state that satisfies the post-condition. “Partial Correctness” drops the termination requirement, and only demands that the post-condition be satisfied if and when the program terminates.

DAFNY verifies the static total correctness of a program (that is total correctness without having to run the program). DAFNY enables users to enter the precondition using the “requires” clause, and the post- condition using the “ensures” clause. DAFNY allows users to enter assertions, loop invariants, and loop variants, using other clauses. These help DAFNY to carry out the verification. The failure of DAFNY to verify a program does not necessarily mean that the program is wrong. DAFNY may just need more help in the form of variants and invariants to establish correctness. On the other hand, a program verified by DAFNY represents something that testing can never achieve: a correct program.

Program verification using DAFNY can be taught in several Computer Science Courses, such as Programming Languages, Software Engineering, Theory of Computation, etc.

This talk will provide examples of programs verified in class using DAFNY.

References

[1] DAFNY - a language and program verifier for functional correctness. `https://rise4fun.com/dafny`.

[2] Nahid A. Ali. A survey of verification tools based on hoare logic. *International Journal of Software Engineering & Applications*, 8:87–100, 2017.

[3] José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto, and Simao Melo De Sousa. *Rigorous software development: an introduction to program verification.* Springer Science & Business Media, 2011.

[4] Richard Bornat. *Proof and disproof in formal logic: An introduction for programmers.* Oxford University Press, 2005.

Spear phishing attack using Kali Linux*

Work In Progress

Imad Al Saeed
Computer Science Department
Saint Xavier University
Chicago, IL 60655
alsaeed@sxu.edu

This assignment is for a course on Cybersecurity for students with no prior knowledge of social Engineering in terms of phishing and spear phishing email attacks. The course instructor explains the social engineering, phishing, and spear phishing attack concepts and the differences between them. The course instructor explains to the students how spear phishing is highly targeted and targets a single individual compared to how the phishing attack targeted and targets hundreds and sometimes thousands of recipients. A spear phishing attack scenario might involve an attacker who is impersonating an organizational IT consultant that sent an email to one or more employees. Their emails will be worded and signed the same way as the IT consultants normally do with their emails. In their way, the company employees will be deceived into thinking that these are authentic emails. This assignment requires using a special software called Kali Linux. The instructor provides the students with the following link to download the Kali Linux software: https://www.kali.org/. Students should download the Kali Linux software image and install it through their virtual box on their machines. In this way, students' computers setting will not be affected. In this assignment, each student chooses a partner, communicates with him/her, and sends simulated phishing emails that contain a suspicious link to a decoy website, such as Google account site, Twitter, Facebook, etc. to his/her partner. Upon receiving the signed email, the student's partner read the email, click on the link, and try to access a decoy website using their username and password (fake username and password). This process should be done in five steps:

Step 1: Creating a web page.
The student should create an HTML page using Notepad software and include a link to a decoy Google account.

*Copyright is held by the author/owner.

Step 2: Initiate the attack.
The student runs the Kali Linux through his/her virtual box, selects the applications menu, and chooses the "Social Engineering" attack option. A new menu with the following selections will show up:

1. Social – Engineering Attack.
2. Penetration Testing (Fast – Track).
3. Third Party Modules.
4. Update the Social – Engineering Toolkit.
5. Update SET configuration.
6. Help, Credits, and About.

The student should choose the first option "Social – Engineering Attack." Inside the social Engineering attack, there are also several options and one of them is "Website attack Vectors. The student will click on that option, and then select the "Credential Harvester Attack Method."

The students can either use "Web Template" provided by Kali Linux or create their own website and import it. But for this assignment, the student will go with the "Web Templets" option to save time.

The software will ask the student to enter the Kali Linux IP address. The student can find their kali Linux IP address by using the (IP address) command. In this way, the software will be entering the listener mode waiting for the user's input.

After that, the student will choose the right website template they want to use. For this assignment, the student will choose the "Google" template as the clone website.

Step 3: Capturing the user's login information.
The student will send an email along with the suspicious link to his/her partner. The partner will open and read the email, click on the link, and enter his/her fake username and password, and click on "Sign in" to access their Google account. At the same time, the Kali Linux listener will record his partner's username and password.

Step 4: Change order.
Now, the students will exchange their roles and repeat the process again.

Step 5: Student observations and attack prevention.
Students should write one page of original content to discuss their experience of the excrement. Also, they should research several ways that might be used to eliminate or prevent this attack from happening. Student research findings will be used for the next class discussion as well.

Faculty-Advisor Relationship Impact on Student Pathways to IT Careers/Education*

Work In Progress

Matthew Cloud
School of Information Technology and Criminal Justice
Ivy Tech Community College of Indiana
Gary, IN 46409
`mcloud3@ivytech.edu`

How do faculty and academic advisor relationships affect students in their decision-making process for careers and education choices in IT? We will explore findings from interviews of faculty and advisors for 8 Computer Science/Information Technology programs at 18 campuses across Indiana in the Ivy Tech Community College on why students follow the paths they do, as well as the challenges and successes of advising within a community college.

*Copyright is held by the author/owner.