

# The Journal of Computing Sciences in Colleges

Papers of the 22nd Annual CCSC  
Northwestern Conference

October 2-3, 2020  
North Idaho College  
Coeur d'Alene, ID

Baochuan Lu, Editor  
Southwest Baptist University

Sharon Tuttle, Regional Editor  
Humboldt State University

**Volume 36, Number 1**

**October 2020**

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

Copyright ©2020 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.

# Table of Contents

<b>The Consortium for Computing Sciences in Colleges Board of Directors</b>	<b>5</b>
<b>CCSC National Partners &amp; Foreword</b>	<b>7</b>
<b>Welcome to the 2020 CCSC Northwestern Conference</b>	<b>8</b>
<b>Regional Committees — 2020 CCSC Northwestern Region</b>	<b>9</b>
<b>Reviewers — 2020 CCSC Northwestern Conference</b>	<b>10</b>
<b>Inclusion in Computer Science Education — Keynote</b> <i>Rebecca Long, Future Ada</i>	<b>11</b>
<b>Hands-On Teaching of RSA Public-Key Cryptosystems With Snap!</b> <i>Alain Kägi, Jens Mache, Lewis &amp; Clark College</i>	<b>12</b>
<b>Using Sentiment Analysis to Highlight the Discrepancy Between High and Low Resource Language Translations</b> <i>Caitlin Garcia, Kaylee-Anna Jayaweera, Quinn Vinlove, Kris Gado, Jens Mache, Lewis &amp; Clark College, Richard Weiss, The Evergreen State College</i>	<b>21</b>
<b>Predicting Student Success in Cybersecurity Exercises With a Support Vector Classifier</b> <i>Quinn Vinlove, Jens Mache, Lewis &amp; Clark College, Richard Weiss, The Evergreen State College</i>	<b>26</b>
<b>Refactoring a Full Stack Web Application to Remove Barriers for Student Developers and to Add Customization for Instructors</b> <i>Jack Cook, Richard Weiss, The Evergreen State College, Jens Mache, Lewis &amp; Clark College</i>	<b>35</b>
<b>Introduction to Parallel Programming Using MPI and OpenMP on the Raspberry PI — Conference Tutorial</b> <i>Xuguang Chen, Saint Martin's University</i>	<b>45</b>

**Supporting and Teaching Students at Liberal Arts Colleges in  
Online Courses — Panel Discussion** 47

*Haiyan Cheng, Willamette University, Shereen Kjoa, Pacific University,  
Anna Ritz, Reed College, Tammy VanDeGrift, University of Portland*

## The Consortium for Computing Sciences in Colleges Board of Directors

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

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

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

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

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

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

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

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

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

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

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

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

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

**Kevin Treu**, Southeastern Representative (2021), (864)294-3220, kevin.treu@furman.edu, Furman University, Dept of Computer Science, Greenville, SC 29613.

**Bryan Dixon**, Southwestern Representative (2020), (530)898-4864, bcdixon@csuchico.edu, Computer Science Department, California State University, Chico, Chico, CA 95929-0410.

**Serving the CCSC:** These members are serving in positions as indicated:

**Brian Snider**, Membership Secretary, (503)554-2778, bsnider@georgefox.edu, George Fox University, 414 N. Meridian St, Newberg, OR 97132.

**Will Mitchell**, Associate Treasurer, (317)392-3038, willmitchell@acm.org, 1455 S. Greenview Ct, Shelbyville, IN 46176-9248.

**John Meinke**, Associate Editor,

meinkej@acm.org, UMUC Europe Ret, German Post: Werderstr 8, D-68723 Oftersheim, Germany, ph 011-49-6202-5777916.

**Shereen Khoja**, Comptroller, (503)352-2008, shereen@pacificu.edu, MSC 2615, Pacific University, Forest Grove, OR 97116.

**Elizabeth Adams**, National Partners Chair, adamses@jmu.edu, James Madison University, 11520 Lockhart Place, Silver Spring, MD 20902.

**Megan Thomas**, Membership System Administrator, (209)667-3584, mthomas@cs.csustan.edu, Dept. of Computer Science, CSU Stanislaus, One University Circle, Turlock, CA 95382.

**Deborah Hwang**, Webmaster, (812)488-2193, hwang@evansville.edu, Electrical Engr. & Computer Science, University of Evansville, 1800 Lincoln Ave., Evansville, IN 47722.

## CCSC National Partners

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

### **Platinum Partner**

*Turingscraft*

*Google for Education*

*GitHub*

*NSF – National Science Foundation*

### **Silver Partners**

*zyBooks*

### **Bronze Partners**

*National Center for Women and Information Technology*

*Teradata*

*Mercury Learning and Information*

*Mercy College*

## Welcome to the 2020 CCSC Northwestern Conference

The 2020 Northwest Steering Committee is very pleased to welcome everyone to the Twenty Second Annual CCSC Northwestern Conference hosted virtually by North Idaho College.

Many individuals and groups have helped to coordinate and support this year's conference and we want to thank them for all of their time and effort, especially given these times of extreme health and safety initiatives. We especially thank the authors who submitted papers, workshops, and tutorials. This year we have accepted four papers, two tutorials, one panel discussion, student lightning talks, and student posters. All papers, panels, and tutorials went through a double-blind review process. We had colleagues across the region serve as professional reviewers and we recognize their generous efforts in providing time and guidance in the selection of our conference program. We are extremely grateful to have Rebecca Long as our keynote speaker for this year! Rebecca is the President and Founder of the Spokane-based non-profit, Future Ada. Rebecca will begin our conference with her keynote address on Inclusion in Computer Science Education.

A final thank you goes out to you the attendees whose participation is essential not only to the continuance of conferences such as this, but also for the continued communication and collegiality you provide between all of us involved in the advancement and promotion of our discipline. We hope you enjoy the first virtual CCSC-NW conference.

Nadra Guizani  
Washington State University  
Conference Chair

Razvan Alexandru Mezei  
Saint Martin's University  
Papers Chair

## **2020 CCSC Northwestern Conference Steering Committee**

Nadra Guizani, Conference Chair ..... Washington State University  
Gabriel de la Cruz, Site Chair ..... North Idaho College  
Bob Lewis, Program Chair ..... Washington State University, Tri-Cities  
Razvan Mezei, Papers Chair ..... Saint Martin's University  
Adamou Fode Made, Panels/Tutorials Chair ..... Humboldt State University  
Gina Sprint, Speakers Chair ..... Gonzaga University  
Ben Tribelhorn, Partners Chair ..... University of Portland  
Xuguang Chen, Student Posters Chair ..... Saint Martin's University

## **Regional Board — 2020 CCSC Northwestern Region**

Shereen Khoja, Regional Representative ..... Pacific University  
Dan Ford, Treasurer ..... Linfield College  
Sharon Tuttle, Editor ..... Humboldt State University  
Shereen Khoja, Past Conf. Chair ..... Pacific University  
Nadra Guizani, Next Conf. Chair ..... Washington State University  
Clint Jeffery, Registrar ..... University of Idaho  
David Hansen, Webmaster ..... George Fox University

## Reviewers — 2020 CCSC Northwestern Conference

Chen, Xuguang ..... Saint Martin's University, Lacey, WA  
Davis, Janet ..... Whitman College, Walla Walla, WA  
Guizani, Nadra ..... Washington State University, Pullman, WA  
Lewis, Robert R. .... Washington State University, Richland, WA  
Mezei, Razvan A. .... Saint Martin's University, Lacey, WA  
Sprint, Gina ..... Gonzaga University, Spokane, WA  
Tribelhorn, Ben ..... University of Portland, Portland, OR  
Williams, Chadd ..... Pacific University, Forest Grove, OR

# Inclusion in Computer Science Education\*

## Keynote

*Rebecca Long*  
*President and Founder*  
*Future Ada*  
*(Spokane-based non-profit)*

Inclusion is an important component of any culture during normal times and in any environment. During a crisis, it's even more critical to the success of communities and organizations. Higher education in STEAM fields, such as Computer Science, is no exception. As we transition forward into this uncharted pandemic world of remote work and schooling, being extra mindful to create inclusive environments in classrooms, advisory meetings and office hours are needed for maximizing the success of STEAM students. I'll show how inclusive techniques can be used to not just support students but also fellow faculty through these trying times.

Rebecca Long is the President and Founder of the Spokane-based non-profit, Future Ada, which supports and advocates for diversity and inclusion in STEAM (science, technology, engineering, art, and mathematics) and by day she is the Quality Assurance Manager at Engie Impact. She has 15 years' experience in software engineering and is a double alum in computer science from Eastern Washington University.

---

\*Copyright is held by the author/owner.

# Hands-On Teaching of RSA Public-Key Cryptosystems With Snap!\*

*Alain Kügi and Jens Mache*  
*Mathematical Sciences Department*  
*Lewis & Clark College*  
*Portland, OR 97219*  
*{alaink, jmache}@lclark.edu*

## Abstract

We describe and evaluate a hands-on activity to teach important aspects of the RSA public-key cryptographic system. We first outline the activity aiming to give students a real and practical understanding of RSA, including a look at the implementation of specific steps in the protocol. Then we discuss our experience using this activity in two separate courses (spring and fall 2019) including a summary of student feedback based on a survey given at the end of one of the classes.

## 1 Introduction

This paper describes our experience designing and using an active-learning approach to teaching RSA [8]. RSA is a public-key cryptographic system that plays a key role today in many internet transactions, including those effected through the secure hypertext transfer protocol or HTTPS [6]. We used the visual, drag-and-drop programming environment called Snap! [9]. Snap! is a programming system well suited for beginners and experts alike. Snap! is easy to use but also supports advanced concepts such as first-class lists, functions, and continuations.

We developed this hands-on activity in the context of our department's introductory computer science course for non-CS majors whose aim is to give students a taste of "algorithmic thinking, the nature of electronic computers,

---

\*Copyright ©2020 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.

and the place of information technology in society.” But we think that this approach can also be successfully applied to more advanced courses.

We used this activity in two different instances of the course (spring and fall 2019). We report our experience teaching this material and present the result of a student survey conducted during the second course (fall 2019).

## 2 Background

In this section we present some details about the RSA public-key cryptographic system and Snap! We do so both to give the reader enough information to understand the activity but also to give a feel for the amount of material a teacher would have to cover in class should they decide to adopt this activity in a future syllabus.

### 2.1 RSA

RSA, named after its inventors Ron Rivest, Adi Shamir, and Len Adelman, is one of the first public-key encryption methods and is widely used today to exchange information on the internet securely [8]. It relies on the current lack of an efficient algorithm to factor the product of two large prime numbers [10].

Let us say that Alice wants to send Bob a message confidentially. Bob, as a new user, will first need to generate a key pair following these steps:<sup>1</sup>

1. Generate two very large, random prime numbers  $p$  and  $q$
2. Compute  $n = p \times q$
3. Compute  $\lambda(n) = \text{lcm}(p - 1, q - 1)$
4. Choose integer  $e < \lambda(n)$  such that  $\text{gcd}(\lambda(n), e) = 1$  (i.e.,  $\lambda(n)$  and  $e$  are coprime)
5. Compute  $d$  where  $d \times e = 1 \pmod{\lambda(n)}$

At the completion of these steps, Bob forgets  $p$ ,  $q$ , and  $\lambda(n)$ ; the tuple  $(e, n)$  becomes his **public** key which he makes available to everyone (e.g., by publishing it to a key server); and he keeps his **private** key,  $d$ , secret!

At this stage, Alice can send message  $m$  to Bob by first encrypting it with his public key  $(e, n)$ :

$$\text{Ciphertext: } c = m^e \pmod{n}$$

Bob can decrypt the message using his private key  $d$ :

---

<sup>1</sup>This description gives enough information to understand the high-level principles of the RSA public-key cryptographic system. However, a truly robust implementation of this scheme depends critically on subtle details omitted here.

Plaintext:  $m = c^d \pmod{n}$

Here is a concrete example using (unrealistically small) prime numbers  $p = 11$  and  $q = 17$ . From this pair of numbers Bob derives his public key ( $n = p \times q = 187, e = 7$ ) and his private key  $d = 23$ . If Alice wants to send 65, the ASCII code of uppercase letter ‘A’, to Bob confidentially, she computes  $c = 65^7 \pmod{187} = 142$ . To decode the message, Bob computes  $m = 142^{23} \pmod{187} = 65$ .



Figure 1: Snap! hello, world! Program.

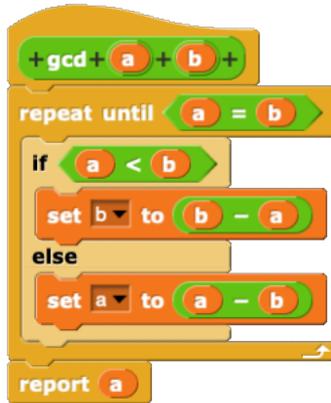


Figure 2: Snap! Greatest Common Denominator Function.

## 2.2 Snap!

Snap! is a visual, drag-and-drop programming language where the user builds a program by gluing together blocks of various kinds [9]. Snap! allows for perhaps a gentler introduction to computer programming. Color schemes help students identify related components. Dedicated geometric contours help students determine legal ways to compose components.

For illustration’s purposes, Figure 1 and Figure 2 show Snap!’s version of Kernighan and Richie’s “hello, world” program [5] and the implementation

of Euclid's algorithm to find the greatest common denominator (GCD) of two numbers. The latter can be used to compute the least common multiple (LCM) of two numbers, a function required in Step 3 of the RSA key generation described earlier.

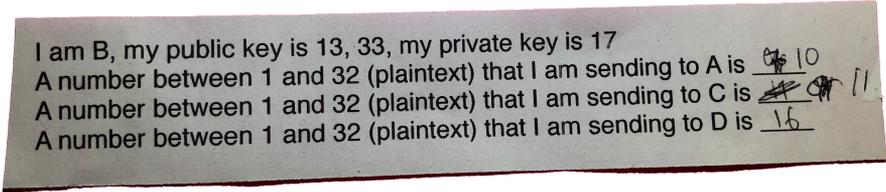


Figure 3: A Sample Public and Private Key Pair.

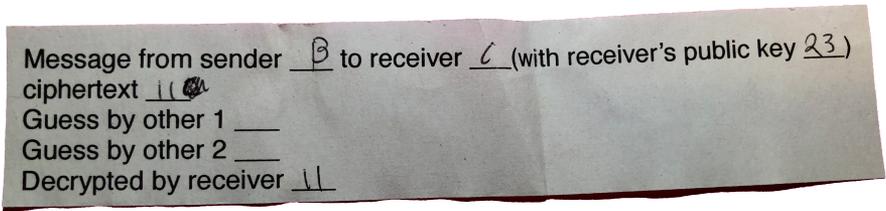


Figure 4: An Encrypted Message.

### 3 The Activity

#### 3.1 First Iteration

In the first iteration of this activity (spring 2019), we gave each student a distinct “toy” public/private key pair (see Figure 3). The students published their assigned public key (simulating a public key server). Then we asked each student to work in groups of four and to use those students’ public keys to encrypt three different messages consisting of a single number between 0 and 32, possibly a birthday (the day of the month). In this iteration, students used the Python interpreter (in a terminal window) to perform the encryptions, for instance evaluating the expression  $31**13 \% 33$  to find that the cipher of message “31” is “25.” Once completed, the students exchanged their encrypted messages (see Figure 4), simulating the open internet. We challenged them to attempt to crack any message. They also decrypted the messages “sent” to

them, again using the Python interpreter, for instance evaluating the expression `25**17 % 33` to decrypt message “25” back to plaintext “31.” Finally, we asked them to verify the decrypted messages with their original senders.

### 3.2 Second Iteration

In the second iteration of this activity (fall 2019), we followed a very similar protocol with the following changes. Instead of providing the students with precomputed public/private key pairs, we distributed a Snap! program that generated those keys on demand. With the key generator came two companion programs to encode and decode messages. A single Snap! project hosted all three programs. The expectation was that students would run the key generation step first, so that the decoder could use the generated keys to perform its job (the student still had to input the encrypted message). The encoder required the user to enter the public key of the desired recipient, in addition to the plaintext. Finally, at the end of lecture, we asked the students for feedback in the form of a survey whose results are presented and discussed in the following Section.

## 4 Evaluation

### 4.1 First Iteration

Many students seemed to enjoy participating in the action, writing down secrets (numbers), handling messages (paper), and computing with keys (using the Python interpreter). Groups of four and everybody sending to everybody else within their group meant 12 total messages per group. To reduce chaos, it may help to be explicit about everybody sending three (potentially different) secrets, everybody decrypting three messages as intended recipient, and everybody optionally trying to break the confidentiality of the remaining six messages that were passed around.

### 4.2 Second Iteration

By the time this exercise was assigned, the students had already studied or completed several Snap! programs (e.g., to compute the greatest common denominator of two numbers, which comes handy when generating the keys; to draw recursive geometric figures; to approximate  $\pi$  using a Monte Carlo simulation; and to study the Monty Hall problem, again, using randomized samples). The availability and relative simplicity of the public/private key generation code gave us an additional opportunity to study a program, perhaps even a step closer to what one might encounter in the “real world.”

Numbers in Snap! appears to be stored in the double precision IEEE floating-point format (at least on 64-bit platforms) [3]. This choice limits the range of whole numbers that can be represented exactly (approximately  $[-2^{53}, 2^{53}]$ ). Unfortunately quantities and computation in cryptography can rapidly overwhelm any fixed-size representation. Thus our activities involved unrealistically small keys and messages.

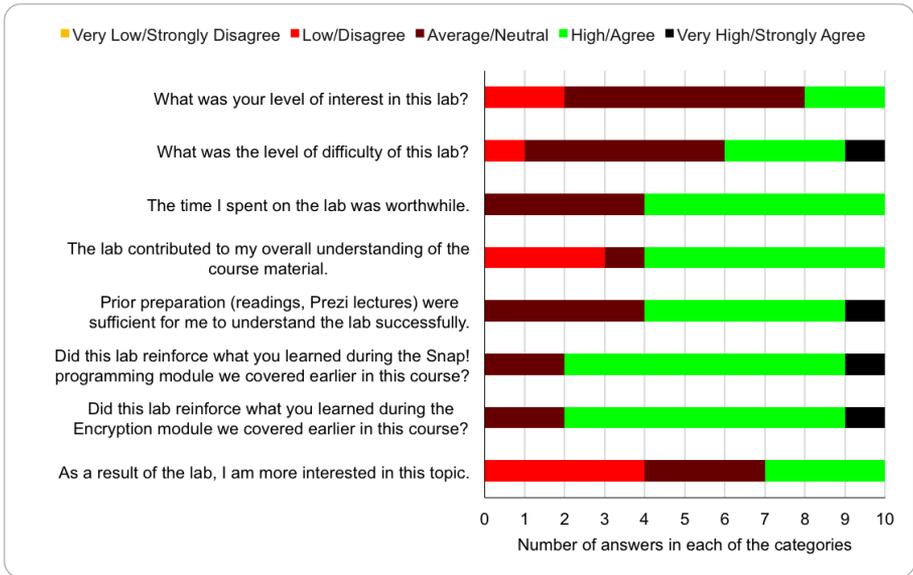


Figure 5: A Subset of the Survey Results. The questions in their entirety appear in Appendix 7. Note that no students gave the answer Very Low or Strongly Disagree to any of these questions.

### 4.2.1 Survey Results

23 students enrolled in the fall 2019 course and 10 students attended this particular class (this class was held at the beginning of Thanksgiving week with many students returning home early). Before the end of class, we asked them to fill the survey described in Appendix 7.

Figure 5 summarizes the students’ answers to Questions 1, 2, and 4 through 9. With regard to Question 3, all students spent at most 60 minutes doing this exercise, with three of them choosing to spend 40 minutes or less. And with regard to Question 10, the majority of students felt that at least a little more guidance through the exercise would have been welcome, with the exception of

a single student who would have wanted more independence.

The majority of the students in this course were upperclassmen majoring in a non-STEM field. Therefore it may not be completely surprising that these students expressed only a mild interest in this subject matter. Perhaps sadly, the activity hardly changed their level of interest in the subject matter. They did find the exercise challenging, with a student who “*felt lost when going through Snap!*” and another confused about “*what exactly n, e & d did in all the operations.*”

Most students reported that the activity was worthwhile and reinforced their understanding of public-key cryptographic operations. In the open-ended section of the survey, some students expressed that the activity was perhaps a little rushed and could have benefited from a little more introductory explanations. A student said that “[*m*]aybe a little more explaining about how to set up the decryption” would have enhanced their learning.

## 5 Related Work

Jeffrey Humpries and Martin Carlisle developed a cryptographic system tutorial [4] embedded in an Adobe Flash plugin [1]. Their application has more robust error checking. Unfortunately, Flash is a deprecated technology, no longer supported by every web browser [2]. Also, the details of their cryptographic programming are hidden in the Flash plugin and therefore these details cannot be inspected, explained, or modified. Our Snap!-based solution allows the instructor to reveal the details of its implementation and gives the students an opportunity to tinker with its programming.

## 6 Concluding Remarks

We have described a hands-on activity to teach RSA public-key cryptography. We have reported on our experience using this activity in two introductory courses (offered in spring and fall of 2019) and shown the results of a survey given to the students of the second course. Students appear to have appreciated and learned some tangible lessons with regard to the importance and details of cryptography.

Specifically, this activity helps students solidify their RSA knowledge with a practical exercise and answer questions such as: When do I use the **private** key? When do I use the **public** key? And whose **public** key?

While we have designed this lesson around the Snap! programming environment, there might be other systems equally well suited to introduce students to cryptography (e.g., Scratch [7]).

The Snap! programming environment supports only numbers with limited precision. A possible extension of this work would be to write a small Snap! library to support basic operations on numbers with arbitrary precision. Such numbers could be implemented with Snap! lists, for instance.

## 7 Acknowledgments

We would like to thank Kaylee-Anna Jayaweera, Richard Weiss, and the anonymous referees for providing useful feedbacks on earlier versions of this manuscript. This work was partially supported by National Science Foundation grant 1723714.

## References

- [1] SWF file format specification. <http://www.adobe.com/devnet/swf/>.
- [2] Flash & the future of interactive content, July 2017. <https://theblog.adobe.com/adobe-flash-update/>.
- [3] IEEE standard for floating-point arithmetic. Technical Report IEEE 754-2019, IEEE Standards Association, July 2019.
- [4] Jeffrey W. Humphries and Martin C. Carlisle. Introduction to cryptography, 2002. <http://williamstallings.com/Crypt-Tut/Crypto%20Tutorial%20-%20JERIC.html>.
- [5] Brian W. Kernighan and Dennis M. Richie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey, USA, second edition, 1988.
- [6] Eric Rescorla. HTTP over TLS. Technical Report IETF RFC 2818, The Internet Engineering Task Force, May 2000. <https://tools.ietf.org/html/rfc2818>.
- [7] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, November 2009.
- [8] Ron L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

- [9] Bernat Romagosa i Carrasquer. The Snap! programming system. In Arthur Tatnall, editor, *Encyclopedia of Education and Information Technologies*. Springer International Publishing, Cham, Switzerland, September 2019.
- [10] Samuel S. Wagstaff, Jr. *The Joy of Factoring*. American Mathematical Society, Providence, Rhode Island, USA, 2013.

## Appendix: Survey Questions

The questions in our student survey appear below. Questions 1 through 10 were multiple choice questions with five alternatives each (Questions 1 and 2: Very low, Low, Average, High, and Very high; Question 3:  $\leq 20$  mins,  $\leq 40$  mins,  $\leq 60$  mins,  $\leq 100$  mins, and  $> 100$  mins; Questions 4 through 9: Strongly disagree, Disagree, Neutral, Agree, and Strongly agree; Question 10: Much more guidance, A little more guidance, Neutral, A lot more independence, Much more independence). Questions 11 through 15 were open-ended.

1. What was your level of interest in this lab?
2. What was the level of difficulty of this lab?
3. Approximately, how many minutes did you spend on this lab?
4. The time I spent on the lab was worthwhile.
5. The lab contributed to my overall understanding of the course material.
6. Prior preparation (readings, Prezi lectures) were sufficient for me to understand the lab successfully.
7. Did this lab reinforce what you learned during the Snap! programming module we covered earlier in this course?
8. Did this lab reinforce what you learned during the Encryption module we covered earlier in this course?
9. As a result of the lab, I am more interested in this topic.
10. Next, would you recommend more guidance (demo, telling) or more independence?
11. What was the most important thing you learned from this lab?
12. Which aspects of the lab were most valuable to your learning?
13. What was the most time consuming or tedious part of the lab?
14. What problems did you encounter in completing the lab?
15. What changes would you make to the lab to enhance your learning?

# Using Sentiment Analysis to Highlight the Discrepancy Between High and Low Resource Language Translations\*

*Caitlin Garcia<sup>1</sup>, Kaylee-Anna Jayaweera<sup>1</sup>, Quinn Vinlove<sup>1</sup>  
Kris Gado<sup>1</sup>, Jens Mache<sup>1</sup>, Richard Weiss<sup>2</sup>*

*<sup>1</sup>Lewis & Clark College*

*{garcia, kjayaweera, quinnvinlove, krisgado, jmache}@lclark.edu*

*<sup>2</sup>The Evergreen State College*

*weissr@evergreen.edu*

## Abstract

Given the enormous cost of cybercrimes each year, many cybersecurity researchers are working to create an automated threat detection system, especially one that can accurately crawl non-English forums and markets. While text classification techniques involving sentiment analysis are fairly successful in English settings, these efforts are ultimately depreciated in non-English platforms. Translation efforts fail to acknowledge the semantic qualities of different languages, especially languages with relatively few bodies of text corpora. To fix these shortcomings and improve current threat detection techniques, it is important to first understand by what degree current methods are failing. In this preliminary study, we highlight the discrepancies of translation quality across three different languages with varying degrees of resourcefulness.

## 1 Introduction

Non-English Dark Net Markets (DNMs) have been on the rise since 2013 [3]. These markets that span across different geopolitical regions offer products ranging from drugs and digital goods to ransomware and keyloggers [4]. Given

---

\*Copyright ©2020 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.

the economic blow from cybercrimes each year resulting in net losses of billions of dollars [2], many cybersecurity specialists have dedicated their time to searching these markets and hacker forums to gain insight into these products and even impending attacks. It is clear that these attacks could be mitigated if we could only detect these threats beforehand. Nonetheless, the unrelenting addition of new products as well as the tens of millions of posts in these forums make it virtually impossible to manually cover all bases.

In the past researchers have employed sentiment analysis to crawl these sites and generate warnings if keywords suggest an impending attack[2][5]. For non-English sites, they are typically first translated into English by services such as Google Translate and then undergo sentiment analysis. While this method works fairly well for English DNMs and forums, its ability to detect threats falls dramatically when it is not English. In particular, the process fails most when confronted with a low resource language [3]. These languages have few text corpora available compared to languages such as English, in which there is plenty of test data for machine translators to apply supervised learning algorithms and thus improve accuracy. Despite positive changes to Google Translate such as the introduction of neural network machine translation in 2016 [6], the service still struggles. When confronted with a language such as Hindi, Russian or even French, it is common that machine translation will overlook the structure and various nuances of the language. Words can have multiple meanings, innocuous idioms such as “killing time” may raise false positives, and what about major misspellings and grammatical errors that are all too common in informal settings?

In order to improve threat detection we must be aware of our blind spots. Cybercrimes are non-discriminatory in their origin and thus it is simply irresponsible to overlook mediocre translations of low resource languages. This paper serves to highlight the discrepancy between high and low resource languages using AWS Comprehend and Google Translate.

## 2 Sentiment Analysis with AWS Comprehend

AWS Comprehend is a Natural Language Processing (NLP) service that uses machine learning to perform sentiment analysis on text data [1]. After text is inputted, the tool outputs a confidence level between 0 and 1 across its perceived positive, negative, and neutral sentiments in the text. A confidence level of 0.99 positive, for example, would suggest that Comprehend is 99% confident that the text is positive. If there are both positive and negative elements in the text, the service provides a “Mixed” score. An overall neutral sentiment suggests a heavy use of objective words like dates, names, or places. This tool can detect a variety of languages and thus felt appropriate for this experiment.

### 3 Approach and Results

Given the relative scarcity of labelled data for low resource languages, our data consisted of randomly selected editorials from the Opinions section from three native news sources of each language. Because many news articles are primarily objective in their sentiment we decided that the Opinions section from newspapers with varied political leanings would be appropriate. These articles were analyzed by AWS Comprehend and the corresponding sentiment levels were recorded to be used as test data. We then translated the same articles into English using Google Translate. In a perfect translation, the sentiment levels would virtually match that of the original, or be very close. Noting the varying confidence levels between the original version and the translated version, as well as if sentiments completely flipped (i.e. went from majority positive to majority negative) indicate Google Translate’s aptitude (or ineptitude) with that particular language.

We chose to evaluate Spanish, French and Hindi for this experiment. While these are all fairly high resource languages when compared to the over 7,000 spoken languages, they all vary in their resourcefulness relative to one another. Spanish is used as the high resource language, French the medium resource language and Hindi as the low resource language in this scenario. There are ten data points per language. Figure 1 visualizes the confidence levels of the overriding sentiment between the translated and original text. Red markers indicate “flipped” sentiment once translated, meaning that the dominant connotations were highly skewed (i.e. a confidently positive text is suddenly confidently negative once translated, or vice versa). Green markers indicate that the sentiment has not changed after the translation. The solid line across the figure represents the ideal correlation, in which the difference in confidence between the two versions is negligible. This ideal scenario would suggest a virtually perfect translation as the sentiments would be the exact same. Clearly, the data does not follow this trend.

One can see that there is a significant discrepancy between our specific high and low resource languages in regards to accuracy rates. While overall sentiment flipped in only about 10% of Spanish text data, that rate jumped to 50% of entries in Hindi data, with French being in the middle at 30%. Confidence levels amongst all three languages faltered slightly across the lower resource languages, but overall these changes were not statistically significant. While this is still a preliminary study, the trends showing a wide gap of translation accuracy across different languages is alarming.

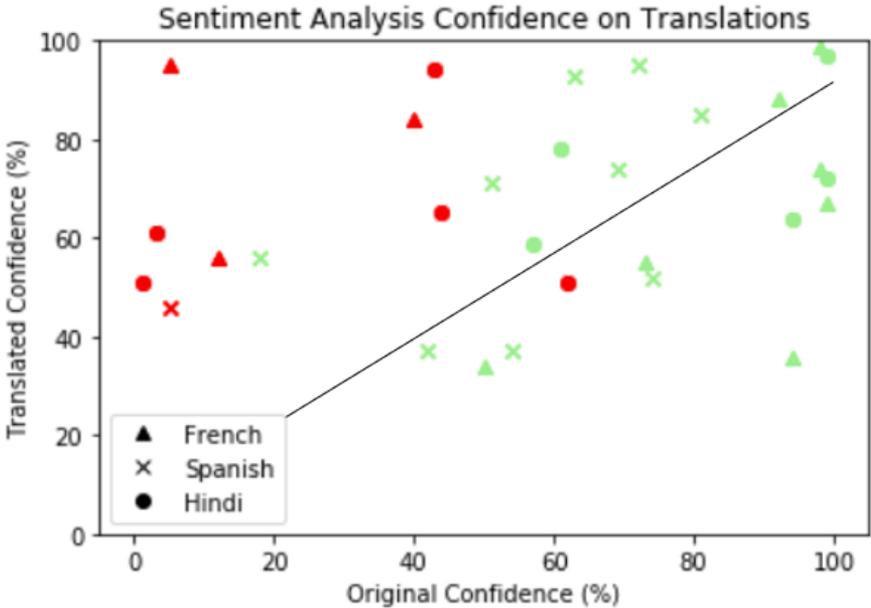


Figure 1

## 4 Discussion and Conclusion

Despite the fact that our data is not reflective of the typical type of data used for threat detection, we are confident that testing sentiment on news editorials is a reliable metric that could be easily extrapolated to these settings. The data used in this experiment, compared to datasets scraped from hacker forums and DNMs is relatively “safe” for Google Translate and AWS Comprehend. In other words, the Opinions section from a reputable news source is generally edited and familiar in its semantics. Trouble ensues once hacker specific language, misspellings and slang enter the arena. This begs the question, if Google Translate struggles with formal, edited text from a low resource language, how can we expect it to perform well once confronted with subsets of regional dialect and no spell check in sight? Even the addition of a threat dictionary and the addition of popular terms would only make this system marginally better given the countless permutations of ways to get a point across and the covert nature of impending attacks.

It is important that we continue to study our blind spots when it comes to machine translation and sentiment analysis. Further studies can use web scrapers to test sentiment in more informal settings such as social media sites

or hacker forums. Additionally, the inclusion of more languages would provide information as to possible ulterior aggravating factors that make one language more difficult to translate, such as its structure. Only once we know where machine translators falter, and by what degree, can we address these issues, and ultimately improve automated threat detection.

## 5 Acknowledgements

We would like to thank Alain Kägi for his guidance and expertise. This work was partially supported by National Science Foundation grant 1723714.

## References

- [1] What is AWS Comprehend? <https://docs.aws.amazon.com/comprehend/latest/dg/what-is.html>.
- [2] Nolan Arnold, Mohammadreza Ebrahimi, Ning Zhang, Ben Lazarine, Mark Patton, Hsinchun Chen, and Sagar Samtani. Dark-net ecosystem cyber-threat intelligence (CTI) tool. In *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 92–97. IEEE, 2019.
- [3] Mohammadreza Ebrahimi, Mihai Surdeanu, Sagar Samtani, and Hsinchun Chen. Detecting cyber threats in non-english dark net markets: A cross-lingual transfer learning approach. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 85–90. IEEE, 2018.
- [4] Sagar Samtani, Ryan Chinn, Hsinchun Chen, and Jay F. Nunamaker Jr. Exploring emerging hacker assets and key hackers for proactive cyber threat intelligence. *Journal of Management Information Systems*, 34(4):1023–1053, 2017.
- [5] Anna Sapienza, Alessandro Bessi, Saranya Damodaran, Paulo Shakarian, Kristina Lerman, and Emilio Ferrara. Early warnings of cyber threats in online discussions. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 667–674. IEEE, 2017.
- [6] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

# Predicting Student Success in Cybersecurity Exercises With a Support Vector Classifier\*

*Quinn Vinlove<sup>1</sup>, Jens Mache<sup>1</sup>, Richard Weiss<sup>2</sup>*

*<sup>1</sup>Lewis & Clark College*

*{quinnvinlove, jmache}@lclark.edu*

*<sup>2</sup>The Evergreen State College*

*weissr@evergreen.edu*

## Abstract

In this paper we explore if we can detect whether students are struggling to complete simple hands-on cybersecurity exercises based on their command line history. These exercises are becoming more popular, especially with the increase in remote instruction. However, students may struggle for many reasons, including lack of some skills, confusion by what is being asked, or confusion about how the testbed works.

Using a small collection of annotated log files from a sample exercise on DeterLab, we were able to generate three features and construct a support vector classifier to predict with 80% accuracy if students would complete the remaining parts of the exercise. Our work could be applied to early detection of students who likely will have difficulty completing the exercise, and offer them hints to boost engagement and learning.

## 1 Introduction

Capture the flag (CTF) exercises have been used in cybersecurity for fun and for training and education for many years. In our own experience, students often become frustrated when they do not make progress. This has also been reported by others [2]. Even with instructor office hours and TA support,

---

\*Copyright ©2020 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.

many students have trouble and struggle with completing hands-on exercises. It can be especially problematic with cybersecurity because it draws on many aspects of computer science, and some of the concepts are difficult. Hands-on exercises can be both more engaging than written homework and can also be more frustrating. Yet, hands-on exercises are very useful for teaching critical thinking and how to apply theory to practice. Our goal is to provide tools to help students be more engaged by automatically detecting when they are struggling, and to potentially offer a hint when that happens.

We used the ‘Introduction to DETER and Unix’, or ‘Intro’ exercise [6] on DeterLab [4], completed by 25 students in a small cybersecurity class, and developed features based on the resulting annotated bash history files produced by ACSLE [3], an automated reporting and log file collection program for testbeds. While the first class we tested our methods on was small, the data collection and annotation system would scale to larger classes and potentially give even more precise results with more data.

One of the important features of the ACSLE framework is the detection of milestones. For each of the exercises we tested, we relied on a well-defined set of milestones that measure student completion of different tasks in the exercise. They are defined in terms of the student’s input, the command output, and the context in which it was used, e.g. the host or directory. The state of the student’s accomplishments at any given time is measured as the set of milestones that have been completed.

We evaluated the classifier using primarily the number of milestones completed, but also the log files themselves with both student input and command line output. We did not have ground truth information on each student as to whether they struggled or not. By constructing three features from this data: the number of related commands between each new milestone achieved, number of repeated commands, and distance from a few ‘known good’ commands, and using these features as inputs into a multi-dimensional support vector classifier, we were able to predict exercise completion with 80% accuracy. Our work will provide a good basis for a new model to suggest hints automatically.

## 2 Related Work

Our work provides a useful addition to the field of cybersecurity education research. Švábenský et al. [9] argued in their meta-analysis of all security education papers that while there are many papers on education methods, there are few new methods beyond surveying students directly. We hope that this paper motivates the use of machine learning models to make evaluation succinct and useful for both students and instructors.

For teaching introductory programming, Piech et al. [7] described a learner’s

path through an exercise as a Markov chain, and also sought to build effective generalizations of the types of ways that new programmers completed a simple assignment in Java. Their work featured more data, more robust statistical analysis, and validation from midterm and final grades, which was difficult for us because our research was mainly conducted after the school year. Rafferty et al. [8] used a more complex hidden Markov model (HMM) for their cognitive science paper, which has a broad focus on how to apply HMMs to model student learning in a variety of contexts. In both cases, a HMM may prove to be a compelling model, and while we looked into it, we were unable to generalize our exercises as a model with a finite number of hidden states.

### 3 Methodology

To construct inputs for our classifier, we first created three features, computed them for every student, and checked our work by evaluating each log file by hand. These features are described below.

The first metric was the average number of commands between new milestones reached. This could possibly describe how much effort was spent for each learning objective met. Initial analysis showed that students who completed more milestones either entered relatively few commands for each milestone that they reached, or took their time and entered more, possibly indicating that they didn't know the answer at first, but worked hard to complete the exercise.

The second metric was the longest string of repeats of any single command. To count commands that were close, but not identical, we would compute the edit distance (or Levenshtein distance) from each new command to the last. With long periods of time where the edit distance was low ( $\leq 3$  characters), we could determine that a similar command with few variations was tried a lot, with little variation. We anticipated that users who achieved more milestones, and subsequently understood the exercise better, would have repeated commands less, but the opposite was true: users who did better generally had a wider variation in the length of their longest repeat sequence than users who didn't do as well, showing that frustration may not entirely be indicated by students trying the same thing over and over again.

The last metric we used was the smallest edit distance between a list of 'known good' commands and bash history. This operated on the initial assumption that while the REGEX search method of seeing if a command matches a milestone may be good, it is binary, and sometimes a continuous metric can produce better results. In some cases, like *find*, a command can be missing an option and not work, but still be close enough to indicate that a student knows what they're doing. We computed this minimum edit distance for each com-

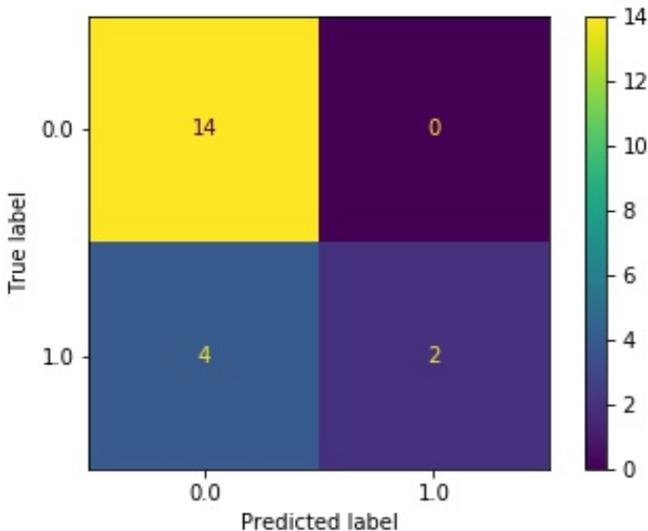


Figure 1: Confusion matrix for the evaluation dataset after training our SVC. Note here that 0.0 represents a ‘will not complete’ prediction and 1.0 represents a ‘will complete’ prediction. Each point in the matrix represents a measured state and its outcome.

mand, and summed them. We found that most high-achieving students had a sum that was small, indicating that what they did was pretty close to a known good solution, while students who got few milestones had bash commands with a large edit distance between these predefined commands.

The exercise log data from two classes at USC and one at LC were loaded into a Jupyter notebook with Pandas, processed with the help of numpy, then normalized with a min-max scalar, shuffled, and split 80-20 between the testing set and validation set. For desired tags, we assigned a true to each student if they got 6 or 7 milestones (all of them) and a false if they didn’t. Then, we placed this data into a scikit-learn support vector classifier [5].

## 4 Results

After training a support vector classifier with the training set, we evaluated our work with the validation set and found that the classifier was able to accurately predict true or false values for 80% of the data. The confusion

matrix is shown in Figure 1. Note that the classifier is pessimistic, which is good, since we want to minimize false negatives. There were no false negatives with the sample data. There were some false positives, i.e. it predicted that 4 out of 20 students would not finish but they did.

## 5 Discussion

Drawing conclusions based on this data set was limited by not having a direct measure of which students actually struggled; nevertheless, we were able to infer this in many cases by reading the bash history logs and counting the milestones achieved. For a machine learning project, we had relatively little data: the ‘biggest’ exercise was the ‘intro’ exercise, with only around 75 samples between all schools involved in the study. Inferring any pattern from this data wasn’t exactly straightforward.

One of the simplest trends we observed was that more persistent students would score higher on the exercises. Encouraging persistence may increase exercise completion rate. In the future, we plan to interview students afterwards to see what they struggled with and correlate that with their command line history. It’s important to note that because our model does not account for overall clock time to completion, persistence simply means trying many new things repeatedly. If a student started the exercise and walked away from the keyboard, our model wouldn’t consider that to be persistent.

Some preliminary work explored how well this model applies when students are just beginning the exercise, not just at the end of it. A variation of this model trained only on the snippets of bash history between new milestones achieved still maintains 80% accuracy (Fig. 3), and points toward increasing completion rates as students persist longer (Fig. 2). To construct this dataset and generate more data, we sliced the data for each student several times progressively, so that a single student who, for example, achieved three new milestones, had three data points, each representing what their bash history data would look like if they stopped working at that point.

ACSLE can be used with EDURange [1, 10, 11], and we plan to integrate an improved version of ASCLE that uses string-edit distance, so that instructors can receive feedback about students in real time. We hypothesize that the string-edit distance can distinguish trial and error guessing from knowledge-based exploration.

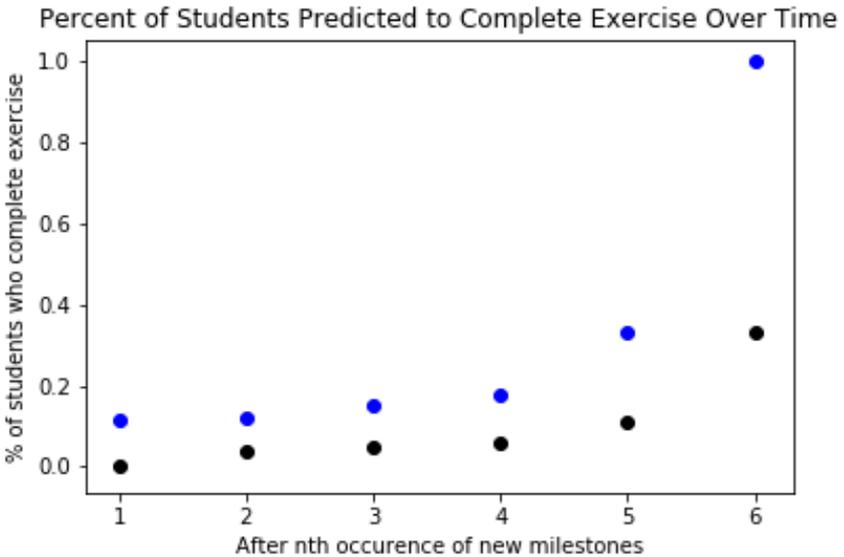


Figure 2: This figure shows the percentage of students who achieved one or more new milestones, then went on to complete the exercise. As students progress and continue to reach new milestones, more stop working. As fewer continue, the ones who do end up with a higher likelihood of completing. The blue dots represent the actual value, and the black dots represent the predicted value.

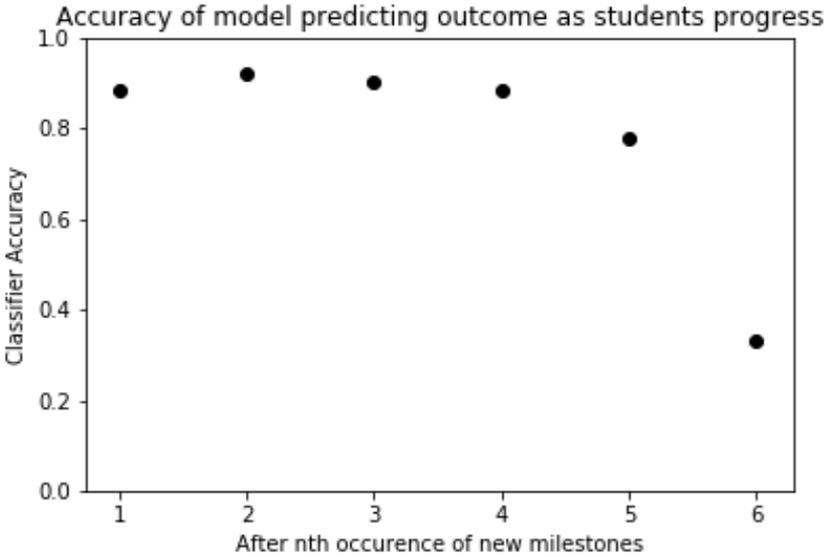


Figure 3: Classifier accuracy for the scale on the x axis described by Fig. 2. Our classifier is able to accurately predict exercise completion with 80 percent accuracy or better for the first five occurrences of new milestones. Accuracy goes down near the end because we had few students who made it that far, but the model is not needed at this point because students who did ended up completing the exercise anyway.

## 6 Conclusion and Future Work

Given a small set of log files with limited information, the system was able to construct a model that estimates whether a student will complete the exercise. This was run using information taken at different times during the exercise, and could potentially indicate when a student was struggling and should be offered a hint. Our work will be useful in extending EDURange, and might be useful in application to other areas where generalized models of student success could be built with very little data.

Since the ACSLE system shows the failed attempts at a milestone, it makes it easier for exercise authors to identify common misconceptions and then produce hints for each of them. One way to use this would be to alert the instructor when a student is struggling, and the instructor could choose one of the pre-recorded hints based on a digest of the student's failed attempts.

In the future, instructors will be able to write a single file that describes commands that students could use in a solution. The string-edit distance between what they typed and those commands would be used to assess progress. Potentially, 'snippets' of them could be used as a hint to assist stuck students after our system detects that they may not finish. After implementing this system, we also plan on evaluating its effect on exercise scores.

### Acknowledgments

We would like to thank Jelena Mirkovic from USC. This work was partially supported by National Science Foundation grants 1723714 and 1723705.

### Research Artifacts

Datasets, along with the accompanying Jupyter Notebook, can be found on GitHub at <https://github.com/edurange/predicting-student-success>.

## References

- [1] Stefan Boesen, Richard Weiss, James Sullivan, Michael E Locasto, Jens Mache, and Erik Nilsen. EDURange: meeting the pedagogical challenges of student participation in cybertraining environments. In *7th Workshop on Cyber Security Experimentation and Test (CSET)*, 2014.
- [2] Kevin Chung and Julian Cohen. Learning obstacles in the capture the flag model. In *2014 {USENIX} Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, 2014.

- [3] Jelena Mirkovic, Aashray Aggarwal, David Weinman, Paul Lepe, Jens Mache, and Richard Weiss. Using terminal histories to monitor student progress on hands-on exercises. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 866–872, 2020.
- [4] Jelena Mirkovic and Terry Benzel. Teaching cybersecurity with DeterLab. *IEEE Security & Privacy*, 10(1):73–76, 2012.
- [5] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.
- [6] Peter A. H. Peterson and Peter Reiher. *Introduction to DETER and Unix*, (accessed August 27, 2020). <https://www.isi.deterlab.net/file.php?file=/share/shared/LinuxandDeterLabintro>.
- [7] Chris Piech, Mehran Sahami, Daphne Koller, Steve Cooper, and Paulo Blikstein. Modeling how students learn to program. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 153–160, 2012.
- [8] Anna N Rafferty, Michelle M LaMar, and Thomas L Griffiths. Inferring learners’ knowledge from their actions. *Cognitive Science*, 39(3):584–618, 2015.
- [9] Valdemar Švábenský, Jan Vykopal, and Pavel Čeleda. What are cybersecurity education papers about? a systematic literature review of sigcse and iticse conferences. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 2–8, 2020.
- [10] Richard Weiss, Michael E. Locasto, and Jens Mache. A reflective approach to assessing student performance in cybersecurity exercises. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE '16*, page 597–602, New York, NY, USA, 2016. Association for Computing Machinery.
- [11] Richard Weiss, Franklyn Turbak, Jens Mache, and Michael E. Locasto. Cybersecurity education and assessment in EDURange. *IEEE Security & Privacy*, 15(03):90–95, May 2017.

# Refactoring a Full Stack Web Application to Remove Barriers for Student Developers and to Add Customization for Instructors\*

*Jack Cook<sup>1</sup>, Richard Weiss<sup>1</sup>, Jens Mache<sup>2</sup>*

*<sup>1</sup>The Evergreen State College*

*{coojac09, weissr}@evergreen.edu*

*<sup>2</sup>Lewis & Clark College*

*j mache@lclark.edu*

## Abstract

This paper describes our experience refactoring EDURange, a full-stack Web application, in order to make it easier for students to do undergraduate research and contribute. As a result, more students were able to contribute to this open source project. In addition, as instructors we wanted to have a simple interface to customize existing exercises and parameterize them so that students could repeat an exercise without it being identical. The main differences were: changing from Ruby on Rails to Python Flask, changing from Virtual Machines to Docker containers, and eliminating dependence on AWS through Terraform. These changes reduced the number of lines of code from 28K to 12K.

## 1 Introduction

Refactoring [2] a large program is hard and time-consuming. So, why would we do it? In our case, we wanted to make some significant changes to EDURange:

- Make it easy for undergraduates to contribute code,
- Make it easy for instructors to create and add their own exercises,
- Make it more efficient, and
- Make it more portable by removing dependence on AWS.

---

\*Copyright ©2020 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.

We made it easier for undergraduates to contribute code by switching to a language they were more familiar with, switching to a simpler Web framework, and reducing the size of the codebase. The original framework was Ruby on Rails and we switched to Python Flask. Ruby is not taught in our curriculum, but Python is. We made it easier for instructors to create exercises by adding a level of automation, switching from Chef to Terraform, and creating a version of the application that runs on a local server as opposed to a Cloud. Using Terraform also increased the portability. Terraform is a more flexible tool and supports several environments. We made it more efficient by switching from using virtual machines (VMs) to Docker containers.

### 1.1 Background

EDURange [1, 5, 6, 4] is an NSF-funded project that is both a collection of interactive, collaborative cybersecurity exercises and a framework for creating these exercises. It is designed to provide students with an active learning environment focusing on analysis skills rather than the latest tools.<sup>1</sup> EDU-Range also allows the instructor to observe student interactions, which can help instructors identify student problems. This environment has typically been provided in the form of one or more Amazon Web Services (AWS) virtual machines, which are launched on-demand with the proper configurations for different exercises. For example, there is an exercise Total Recon where students find hosts with specific ports open or services running. In our original version, each host was a separate VM.

The original version of EDURange was developed using Ruby on Rails for the front-end web application in connection with a Postgres database for managing accounts, groups, and exercises. Additionally, the back-end of EDU-Range is supported by a Redis server for issuing and scheduling commands, such as starting and stopping VMs, or sending emails to users who request a password reset.

One of the design goals for the original version was that it should be easy to use in a demo. We were able to use either pre-made accounts or we could have users sign up at the beginning of a workshop. The one problem that we had was that if users arrived late, then we couldn't add them to the scenario. We have given over 20 demos at conferences and while in the beginning there were some delays of up to 20 minutes, we never had to cancel a demo. It was very successful, but sometimes it became expensive to use in the classroom.

EDURange was initially designed to allow instructors to create and customize their own scenarios. The main feature that enabled this was that scenario creation was scripted. There were template scripts with parameters that

---

<sup>1</sup><https://edurange.org/about.html>

would control the complexity of the exercise. That turned out to be difficult to create and maintain using the combination of Chef, YAML and Ruby on Rails. Terraform is allowing us to reintroduce these customization features because of its simplicity and flexibility.

Terraform works with multiple platforms, e.g. AWS, Docker, OpenStack, VSphere, and it is more modular than other virtual infrastructure provisioning frameworks. To get started with Terraform, we create a basic template for a configuration file that includes basic container image and network structure details. From this template, each individual exercise can then add its own required packages, scripts, and the accounts needed for students to play it. This allows complete configuration files to be easily written in a modular fashion with scripts, as opposed to manually writing the configuration for every individual container required for an exercise. There are two layers of automation: the configuration file automates creating the containers, and the template automates creating the configuration file. The templates we need to use are also very concise, Terraform can launch a single Docker container with a configuration file as short as 20 lines.

## 2 EDURange Structure

There are three main parts to the structure for EDURange: 1) the web framework, 2) the control framework for provisioning the virtual environment and collecting data while the exercises are running, and 3) the database for storing information about scenarios and users.

### 2.1 Web Framework

In this refactor, we chose to use Python as our language for web development instead of Ruby on Rails. The primary motivation for this choice was that although Ruby on Rails may be a more popular enterprise platform, Python is offered as part of our curriculum, so Python is much more likely to be accessible by future developers. In fact, we have had summer research students who worked on designing scenarios but never managed to learn Ruby on Rails. Additionally, we wanted a more lightweight framework than Rails, allowing developers to more easily stay organized and make small, modular changes to the application.

To kickstart the development of the new Flask application, the `cookiecutter-flask`<sup>2</sup> tool was used to setup the directory structure of the project, as well as provide basic initial configuration for several Flask extensions. WebPack is

---

<sup>2</sup><https://github.com/cookiecutter-flask/cookiecutter-flask>

integrated to compress Javascript and CSS files, which significantly improves the performance of the website.

At a high level, the refactor web application is a REST API that processes user requests in three stages. First the application determines what page is being requested and by what method, also known as the route. Then, browser session variables are checked against the database to validate whether or not a user is allowed to access that particular route. Lastly, if the check passes, the data from any form submitted by the request is saved, and the page is served. This design choice was motivated by the desire to create more explicit separation between different utilities of the web application, and requiring the ability to make modular and incremental changes. For example, if a developer wanted to add a new page to the website, it would be a three step process:

1. Define the route to your page, `edurange.org/new.html` for example.
2. Create that HTML file, which only requires two lines to inherit the site-wide navigation and style options.
3. Create any forms or tables as needed for the functions of the new page.

This workflow and separation of different utilities greatly simplifies adding new features and pages to the website. Comparatively, adding a new page on the Ruby on Rails EDURange platform would require the definition of the new route from the Rails command line, followed by the creation and modification of several Ruby and HAML files, as well as individual modals for all forms, tables, and popup dialog boxes.

In summary, Flask is in a preferable framework because it is more lightweight and more easily extensible. One of the most important extensions to our application that Flask allows us to preserve is the Redis service, which is integral to exercise management.

## 2.2 Control Framework: Scenario Management

One of the advantages of containers is that it would bring down cost if running on AWS, and it would make it feasible to run scenarios on a local server. Amazon also imposes limits on the number of virtual clouds and machines can be in use in a given region, which can limit the number of exercises that can be running simultaneously. Some scenarios require 10 virtual environments. Lastly, we also wanted instructors to be able to run EDURange on a local network not connected to the Internet. [3]

Using containers means:

1. Only one machine would be required; the same machine that hosts the EDURange application would host the containers that encapsulate different scenario environments.
2. While this host machine would need significantly improved performance over the current host machine, the EDURange application could be more easily distributed and hosted anywhere, without dependence on AWS.

Because Terraform is capable of configuring virtual subnets to connect containers, there is no need to create virtual clouds and new IP addresses through AWS, instead scenarios are hosted on an open port of the host server. This does mean that everything is running on one machine, which would require significant boosts to computational power and network bandwidth. However, containers are lightweight enough that the capacity for running scenarios could be nearly limitless, while the costs would remain static for only the price of one machine. Alternatively, instructors could put EDURange on one of their institution’s servers, and access it locally without concerning themselves with AWS fees.

### 2.3 Database Schema Improvements

Another major improvement was refactoring the Postgresql Database, to remove obsolete tables and simplify the relationships between them. Here is a comparison of the tables contained in the legacy system, versus the refactor:

Legacy Database			
Answers	Bash-histories	Clouds	Groups
Instance-groups	Instance-roles	Instances	Players
Questions	Recipes	Role-recipes	Roles
Scenarios	Student-group-users	Student-groups	Subnets
Users			

Refactor Database			
Answers	Bash-histories	Group-users	Groups
Questions	Scenario-users	Scenarios	Users

Some tables that stored internal metadata in the legacy system have been omitted for brevity, but overall we’ve managed to reduce the number of tables required for the minimal operation of the platform from 24 to 8. Primarily, all of the tables related to Roles and Recipes were only required for Chef, and the monitoring of individual instances and subnets has been transferred to Terraform rather than relying on the database. We’ve also simplified the

way groups and scenarios are managed, since having tables for "Users, Players, Student-Groups, Student-Group-Users and Instance-Groups" is needlessly confusing for new developers. It is extremely difficult to determine the importance or roles of those tables in the legacy database, even when looking deeper than just the table names.

### 2.3.1 Scheduling: Redis with Celery

As previously mentioned, Redis has been the primary service used for enabling EDURange to run console commands. This includes the two core Terraform commands "terraform apply" and "terraform destroy", which respectively allocate and free resources. For the purposes of security and organization, these commands are handled by a scheduling worker that runs within the Redis broker. In the Rails version of EDURange this scheduler is Sidekiq, and for the Python refactor the scheduler is a service called Celery. These services perform the same utility, and that is to make sure that requested tasks (such as starting a scenario, or collecting logs) are monitored, executed, and don't cause conflicts. For example, it should prohibit the modification of a running scenario or the collection of logs from a scenario that's been deleted.

What makes Celery perfect for this piece of the project is the ease with which developers can specify new tasks. Essentially, each task is just a single python function that is defined in central "tasks.py" file. These tasks can range from being extremely complex scripts that write out Terraform configuration files for scenarios (a work in progress), or very simple like downloading logs from a running container.

One simple task that we are able to re-use for many different utilities in EDURange is our *send\_async\_email* task, which is only 7 lines long. Celery simply packages *email\_data* from a web form into a Flask Mail *Message* object, and sends it off. Because of how generally this function is designed, we can use it for things like password resetting, scenario duration warnings, or any other utility that requires an email to be sent. These Celery tasks can be written to support any arbitrary utilities, and the number of concurrent running tasks can be scaled easily by running parallel workers, though that shouldn't ever be necessary.

## 2.4 Deployment

At this point it may seem that all of these services would take a great deal of effort to install and configure to work properly, but a major priority of this project has been to make the EDURange platform more easily accessible, and even possible for instructors to easily run it on their own machines. In service of this goal, we've implemented two options for running the EDURange platform.

The first is a wrapper using Node Package Manager which, after a one-time installation of Node, can gather and update all the requirements of the application, and run all the required services with a working out-of-the-box configuration. This is how the EDURange server will run once all scenarios are available in the refactor.

The second option is to run the platform through a pre-defined Docker-compose file. The advantage of this option is that it requires no additional installation of tools on the host server, once Docker is installed. Instead of running the required services directly on the system, it will download a pre-configured container for each micro-service. We have yet to do performance testing on this approach, but it may be more costly due to running extra possibly nested containers. That wouldn't matter for instructors that may only be running one or two scenarios at a time, but it would be much more costly in terms of performance if the central server were running this way.

## 3 Results

### 3.1 Website

The front-end web application has been fully implemented, excluding some specific functionality related to scenarios such as the code responsible for monitoring and logging student activity. Because some of these features are missing, it's not completely fair to compare the amount of code from the two web applications. However, the massive difference between them is still indicative of the scale of the reorganization, yet this was accomplished in 6 months as a two-quarter project course for three students.

The legacy application actually has more Python than the refactor, despite being written primarily in Ruby. This is due to frequent redundancy in code, whereby some very large Python files for logging student activity need to be stored redundantly across all scenarios. The reduction in code was one part of the reorganization. Simply cleaning up and removing redundant code would not have simplified the structure and could have taken a comparable amount of time.

Legacy Web App		Refactor Web App	
Language	Number of Lines	Language	Number of Lines
Bash	1204	Assembly	394
C	359	Bash	745
CoffeeScript	49	C	1977
CSS	418	CSS	152
HCL	3973	Dockerfile	46
JavaScript	1917	HTML	2062
Markdown	2133	Javascript	391
Pan	1238	JSON	1668
Perl	3680	Python	3323
Python	3770	Shell	47
Ruby	7635	YAML	932
Ruby HTML	840		
YAML	1254		
Total	28,290	Total	11,737

### 3.2 Scenarios

At this point, six of the eight original EDURange scenarios have been converted to work in the refactor, and the system is fully functional. Scenarios are being dynamically created from Terraform templates, and the management system ensures that network addresses do not overlap, and that all containers are accessible.

The first release demonstrated some performance and efficiency improvements. Most notably, the minimum amount of time required to boot up a scenario has gone from two minutes, to less than ten seconds. This is because Terraform can simply start up containers from pre-downloaded images, rather than having to communicate with AWS and waiting for virtual machines to start. The refactor also improves the performance of the scenarios themselves, because containers can evenly divide up the hardware resources of the host machine rather than being restricted by preset AWS instance sizes.

## 4 Conclusions and Future Work

At this point, the refactor has been a success in terms of making the platform easier to approach for new developers, as well as more easily expandable. This can be demonstrated through the vastly simplified database schema, the reduction of the amount of code used to initialize the web application, and the better organized workflow for adding new pages and features to the platform. The best evidence for the success of the refactor, however, comes from the active participation and rapid progress made by the new development team.

Previously, after attempting to train 10 new developers to work on the Rails platform over the Summer of 2019, only 1 developer was able to become confident with Ruby on Rails development. By comparison, the 3 students trained during this refactor were all able to feel equipped to begin contributing within a few weeks of learning how to use Flask.

The next task in this refactor will be expanding the scenario management functionality. The platform should facilitate instructors creating their own fully customized scenarios with relative ease, by automating the generation of Terraform configurations from templates through the instructor interface. Since Terraform configurations are modular, we can easily implement a form on the web application where instructors can input information about the changes they'd like to make to a scenario. Student researchers can contribute early on in their learning process by writing bash scripts that accomplish small tasks, such as installing packages or changing ssh port numbers. As our library of small bash scripts grows, instructors will more easily be able to customize and create new scenarios.

The scenario scoring system must be re-implemented. In order to assess student understanding we don't just rely on their completing the tasks, instead we have questions that they answer in the student interface. In general, the correct answers are supplied by the database. This was never fully implemented in the original system, where ad hoc methods were used. However, using the database allows the creation of queries to filter and project tables to give instructors customized views of the scoring results.

Another feature that will be integral to research work related to the platform is student activity logging. This activity includes everything a student types and sees, and everywhere a student goes within a scenario. On the Rails EDURange platform this is implemented through custom TTYLog scripts and a data pipeline using an AWS S3 bucket from the scenarios to the central server [4] Log management will hopefully be simplified since the scenario containers will be more easily accessible than VM's, but the logging software is fragile since it must handle anything the student can type.

## Research Artifacts

If you would like to find out more about EDURange, and maybe even use it yourself, see our GitHub repository for more details and setup instructions: <https://github.com/edurange/edurange-flask>

## Acknowledgements

This work was partially supported by National Science Foundation grants 1723705 and 1723714.

## References

- [1] Stefan Boesen, Richard Weiss, James Sullivan, Michael E Locasto, Jens Mache, and Erik Nilsen. EDURange: meeting the pedagogical challenges of student participation in cybertraining environments. In *7th Workshop on Cyber Security Experimentation and Test (CSET)*, 2014.
- [2] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.
- [3] Cynthia E. Irvine, Michael F. Thompson, Michael McCarrin, and Jean Khosalim. Live lesson: Labtainers: A docker-based framework for cybersecurity labs. In *2017 USENIX Workshop on Advances in Security Education (ASE 17)*, Vancouver, BC, aug 2017. USENIX Association.
- [4] Jelena Mirkovic, Aashray Aggarwal, David Weinman, Paul Lepe, Jens Mache, and Richard Weiss. Using terminal histories to monitor student progress on hands-on exercises. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 866–872, 2020.
- [5] Richard Weiss, Michael E. Locasto, and Jens Mache. A reflective approach to assessing student performance in cybersecurity exercises. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE '16*, page 597–602, New York, NY, USA, 2016. Association for Computing Machinery.
- [6] Richard Weiss, Franklyn Turbak, Jens Mache, and Michael E. Locasto. Cybersecurity education and assessment in EDURange. *IEEE Security & Privacy*, 15(03):90–95, May 2017.

# Introduction to Parallel Programming Using MPI and OpenMP on the Raspberry PI\*

## Conference Tutorial

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

Parallel computing can be explained as using multiple processing units to simultaneously solving a single problem. The potential application areas are many, such as climate modeling, computational astrophysics, financial risk management, medical imaging, agriculture estimates, and computational fluid dynamics.

Parallel programming models are various, and two widely known models are message-passing models and shared-memory models. Each task in a message-passing programming model has its private memories, and different tasks can communicate each other through message exchange. On the other hand, all tasks in the shared-memory programming model will share a common address space where they can asynchronously read and write.

Message Passing Interface (MPI) is a specification that primarily addresses the message-passing models. It is designed by a group of researchers from academia and industry. MPI specifies the syntax and semantics of a core of library routines for message-passing programs written in C, C++, Java, or Fortran. Open Multi-Processing (OpenMP) is jointly defined by a group of computer hardware and software vendors. It is an application programming interface supporting multi-platform and shared memory multiprocessing programming in C, C++, Java, or Fortran.

The Raspberry PI is a series of small single-board computers that were developed in the United Kingdom by the Raspberry PI Foundation, the purpose of which is to promote teaching of basic computer science. It can be applied

---

\*Copyright is held by the author/owner.

to many areas including parallel programming. Raspberry PI is inexpensive, but it can provide each student with his or her own parallel processors and full advantage of the multicore capabilities.

This tutorial will cover the basic techniques and skills needed by parallel programming using MPI and OpenMP on the Raspberry PI. It has three parts. In part I how to acquire and install the software needed for MPI and OpenMP programming on the Raspberry PI will be introduced, especially such software as MPICHI, MPJ Express, omp4j, and Pyjama. Part II will focus on how to compile and run MPI and OpenMP programs. It begins with the examples of MPI programs written in C, followed by Java MPI examples. After that, how to run and compile OpenMP programs in C and Java is described. Finally, some basic MPI routines and OpenMP directives needed for parallel programming is explained in part III.

The intended audience of this tutorial is anyone who is a beginner to parallel programming and/or is interested in MPI and OpenMP programming. The expected learning outcomes include the followings. Firstly, after attending the tutorial, the audience should know what MPI and OpenMP are and how to implement them in C and Java. Then, the audience should know the software used for MPI and OpenMP, how to get a copy and install. Other than that, how to run MPI and OpenMP programs in C or Java on the Raspberry PI will be learned. In the end, the audience should know some basic OpenMP directives and MPI routines needed for parallel programming. The recommended devices for an audience include a laptop and if available a raspberry PI. During the tutorial, a limited number of the raspberry PIs will also be available for the audience to try. At the end of the tutorial, the e-version of the lecture notes, code in C and Java as examples, and other materials for self-study can be provided, if needed.

# Supporting and Teaching Students at Liberal Arts Colleges in Online Courses\*

## Panel Discussion

*Haiyan Cheng<sup>1</sup>, Shereen Kjoa<sup>2</sup>, Anna Ritz<sup>3</sup>, Tammy VanDeGrift<sup>4</sup>*

*<sup>1</sup>Computer Science Department*

*Willamette University, Salem, OR 97301*

*hcheng@willamette.edu*

*<sup>2</sup>Mathematics and Computer Science Department*

*Pacific University, Forest Grove, OR 97116*

*shereen@pacificu.edu*

*<sup>3</sup>Biology Department*

*Reed College, Portland, OR 97202*

*aritz@reed.edu*

*<sup>4</sup>Computer Science, Shiley School of Engineering*

*University of Portland, Portland, OR 97203*

*vandegri@up.edu*

## 1 Summary

Faculty members from four different institutions will share practices to support and teach students in online courses based on experiences from spring 2020. Each panelist will provide an overview of pedagogy, resources, tools, and technology they used during in-class and remote instruction. Each panelist will share experiences about how their pedagogy transitioned to an online setting and how they continued to support students' welfare and learning. Panelists will share what went well, what did not go as well, and what lessons and strategies they plan to bring to the face-to-face classroom (see Figure 1). After the panelists present, attendees will be invited to share tips and resources for the benefit of the CCSC community.

---

\*Copyright is held by the author/owner.

Panelist	Willamette	Pacific University	Reed College	Univ of Portland
Courses	1. Fundamentals of Data Science 2. Computer Science Senior Seminar	1. CS II (co-taught) 2. Computer Security 3. Software Engr II 4. Independent Study & Intern Supervision	1. CS0 (non-majors) 2. Journal Club Style Seminar	1. Data Structures 2. Analysis of Algorithms
Tools	Zoom live and video Datacamp Linkedin Textbook	Moodle Zoom Box iPad (Goodnote) Videos created by instructor (iPad & zoom)	Repl.it: online coding environment Moodle Zoom Videos created by instructor & youtube videos from textbook	Microsoft Teams Moodle Videos created with ipad and comprehension questions
In-Class Time	Working through examples Team projects using breakout rooms	Discussion of videos Working through problems Small discussions in breakout rooms	Watching videos, Q/A, completing comprehension questions. collaborative online labs	Q/A Problem-solving practice over Programming labs
Out-of-Class	Team/individual consulting Watching assigned Datacamp lessons and working on exercises Completing assignments	Complete reading questions about the videos and book reading	Watching videos, completing programming assignments	Watching videos Reading textbook Homework Programming projects
Supporting Welfare	Written homework of self-reflection on individual learning styles	Checking in with students during class and office hours	Moodle polls to see what's working, one-on-one checking-in during lab.	In-class check-ins (polls and discussion) Welfare Surveys Emails
Lessons Moving Forward	Constructive Feedback	Sharing materials with colleagues	Synchronous lab-time Structured assignments	Virtual Office Hours Flipped Classroom
Challenges	Grading individual and team project, counting on students to report contributions	Student motivation and student expectations	Shifting rules & regulations Easier for students to "check out"	Proctoring exams Stressed students Delivering the small-college experience

Figure 1: Courses, Tools, Coursework, Supports, and Challenges of online courses