The Journal of Computing Sciences in Colleges

Papers of the 14th Annual CCSC Southwestern Conference

March 26th-27th, 2021
UC San Diego (virtual)
San Diego, CA

Baochuan Lu, Editor
Southwest Baptist University

Mariam Salloum, Regional Editor
UC Riverside

Volume 36, Number 10 **April 2021**

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

Table of Contents

The Consortium for Computing Sciences in Colleges Board of Directors

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

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

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

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

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

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

Judy Mullins, Central Plains Representative (2023), Associate Treasurer, (816)390-4386, mullinsj@umkc.edu, UMKC, Retired.

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

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

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

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

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

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

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

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

Bryan Dixon, Southwestern Representative (2023), (530)898-4864,

bcdixon@csuchico.edu, Computer Science Department, California State University, Chico, Chico, CA 95929-0410.

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

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

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

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

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

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

CCSC National Partners

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

Platinum Partner

Turingscraft
Google for Education
GitHub
NSF – National Science Foundation

Silver Partners

zyBooks

Bronze Partners

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

2021 CCSC Southwestern Conference Committee

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

Regional Board — 2021 CCSC Southwestern Region

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

An Analysis of Using Coral Many Small Programs in CS1*

Joe Michael Allen and Frank Vahid
Department of Computer Science and Engineering
University of California, Riverside
jalle010@ucr.edu, vahid@cs.ucr.edu

Abstract

Coral is an ultra-simple programming language designed to look like pseudocode while resembling industry programming languages like C++, Java, and Python. Coral was created specifically for learners and thus, in 2019, our CS1 began teaching programming fundamentals with Coral during the first 3 weeks before switching to C++ for the remainder of the term. Our university already adapted a many small programs (MSP) teaching approach which involves assigning students multiple smaller assignments instead of only giving them one large assignment each week. In this work, we share our experience using a hybrid Coral/C++ MSP approach versus a pure C++ MSP approach. We summarize similarities and differences between student performance and other metrics such as time spent, start date, and more. We found that instructors can use a hybrid Coral/C++ approach to have an easier class startup while maintaining high student grade performance.

1 Introduction

Introductory programming courses known as CS1, are known for their plethora of problems leading to high student stress and high DFW rates. These high numbers are a result of many different factors [9, 6, 7, 10]. These issues are especially problematic for CS1 courses as they are essential to keep students in the major, attract new students to programming, and to introduce non-major students to the basics of programming. Like many other universities

around the nation, our university has also struggled to find ways to alleviate these issues. To try and remedy this problem, we began actively pursuing intervention strategies to make our CS1 more accommodating to our students.

1.1 Many small programs (MSPs)

In 2018, we adopted a many small programs (MSPs) teaching approach. An MSP teaching approach involves assigning students multiple smaller programming assignments, typically 5-7, each week instead of only assigning one large programming assignment (OLP) each week. Using an MSP approach allows instructors to give students more assignments to practice programming concepts without overloading them with too much additional work. Previous research [5, 4] has shown that using an MSP teaching approach can improve student grade performance and decrease student stress. Other research has shown other benefits such as earlier start dates, good time spent on programming assignments, better exam preparation, and more. We have found success with this approach and have received positive feedback from our students and our CS1 instructors.

1.2 Coral

In 2019, we tried another intervention technique: we taught our CS1 via a hybrid approach of Coral and C++ together. Coral is an introductory web-based, pseudocode-like language designed to help learners [1]. Coral is free to use and resembles popular commercial programming languages like C++, Java, and Python, allowing for a smooth transition between languages. The language comes with a limited set of 7 instructions to help students focus on the fundamentals of programming. Not only is Coral fully executable, it also comes with a flow chart language to help visualize the execution of the code in real-time.

The authors of Coral published an initial work showing Coral's ease of use and we decided to apply the language in our CS1 [8]. We had considered using other introductory programming languages like Snap [3] or Scratch [2], but we found they are not designed for a CS1 class. We began using Coral at the start of the class and then switched midway to C++. This paper is written to share our experiences and findings from our second time using this approach.

2 Methodology

2.1 Course

We analyze a Spring 2020 CS1 course taught at our public research university. Our CS1 typically serves around 300-500 students during a 10-week quarter (fall, winter, spring) split into 3-5 sections of 80 students. All sections use the same zyBooks interactive textbook and require students to complete the same weekly participation activities (class readings), challenge activities (small coding homeworks), and lab activities (programming assignments). Our CS1 regularly serves half computing major students and half non-major students. The course is taught fully in C++ and covers basic input/output, variables, expressions, branches, loops, functions, and vectors.

2.2 2.2 Experiment details

For one CS1 class section (hybrid Coral/C++ group) we taught Coral for the first 3 weeks and then switched to C++ instead of the typical way of teaching C++ for all 10 weeks (pure C++ group). Other differences between each group include the instructors; however, they both have a very similar teaching style and consistently earn similar marks on the end-of-quarter student reviews and the midterm as hybrid group had a few additional Coral related questions. All other class components were the same, including the lesson plan, interactive online textbook, assignment deadlines, etc.

2.3 Data Collection

We asked zyBooks to provide us with a detailed log of all student activity for our CS1 class. Student activity consists of develop runs, when a student tests their code using zyBooks' automated system, and submit runs, when a student turns in their code for grading. Each log entry includes the activity name, an anonymized user ID, a score, a max score, and a timestamp.

3 Student grade performance

We gathered gradebooks for each section and to calculate average scores on weekly MSP assignments we gathered all student activity. Students that did not submit any code for grading in a given week were excluded from calculations.

Results: Figure 1 shows our results. The pure C++ group data is shown on the left bars and hybrid Coral/C++ group data is shown on the right bars. The grade percentage is on the y-axis and the week number is on the x-axis. A

Table 1: Student grade performance on all categories of our CS1 class

Class category	Pure C++	Hybrid Coral/C++
Total class grade	88%	95%
Final exam	83%	88%
Midterm exam	83%	95%
Participation activities	94%	95%
Challenge activities	94%	95%
Lab activities	96%	93%

total grade average column is added to the end of the chart. Table 1 summarizes the average grades for all class categories.

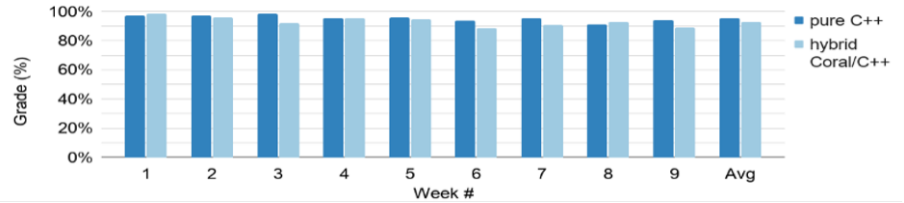

Figure 1: Grade performance results: Both the pure C++ group (avg. 96%) and the hybrid Coral/C++ group (avg. 93%) scored equally well on MSP assignments.

Figure 1 shows that both the pure C++ group (96%) and the hybrid Coral/C++ group (93%) do equally well on weekly MSP assignments. Table 1 also shows that both groups perform well in all categories of the class, with the hybrid C++/Coral group slightly outperforming the pure C++ group.

4 Weekly MSP assignment metrics

We report results on various metrics related to weekly MSP assignments. For each metric, calculations exclude students that did not attempt any lab activities for the given week. For all charts, the pure C++ group data is shown on the left bars and the hybrid Coral/C++ group data is shown on the right bars.

4.1 Time spent

We expect students to spend around 3 hours working on lab activities each week. To measure student time spent, we summed the differences between

each activity timestamp; excluding differences greater than 10 minutes as we considered the student to have taken a break or moved on something else. As such, this data is likely an under-representation as students could have spent that time studying or testing their code outside of the zyBooks IDE.

Results: Figure 2 displays our results. The total time spent is on the y-axis and the week number is on the x-axis. A total time spent average column is added at the end of the chart.

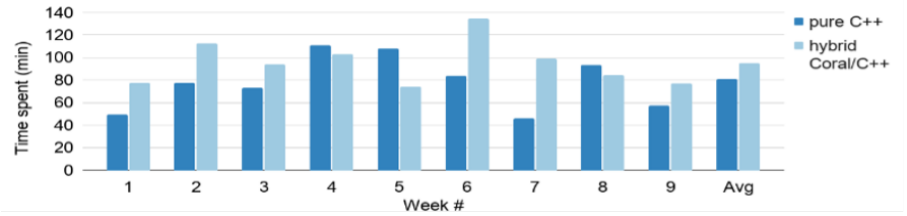

Figure 2: Time spent results: The hybrid group (avg 95min) spends slightly more time working on MSPs each week than the pure C++ group (avg. 81 min).

Figure 2 shows that the pure C++ group (81 minutes) spends less time working on MSPs each week than the hybrid Coral/C++ group (95 minutes).

4.2 Activity runs (develops / submits)

We sought to understand how students develop their code and how frequently students test (develop run) and check (submit run) their code while working. We gathered all student activity and calculated the average number of develop runs and submit runs on weekly MSP assignments.

Results: Figure 3 displays our results. Develop runs are indicated by the dark bars at the bottom and the submit runs by the light bars at the top. The total number of develop/submit runs are on the y-axis and the week number is on the x-axis. A total average column is added at the end of the chart.

Figure 3 shows that the pure C++ group develops less than the hybrid Coral/C++ group, but submits more frequently. To fully understand the data, a more in-depth analysis is required; however, since there are more develops than submits on average, it seems like students show a healthy programming practice of testing their code (developing) and then submitting.

4.3 Start date

Each assignment is due one week from the time it is assigned. We consider starting at least 2 days prior to the assignment's due date as healthy behavior.

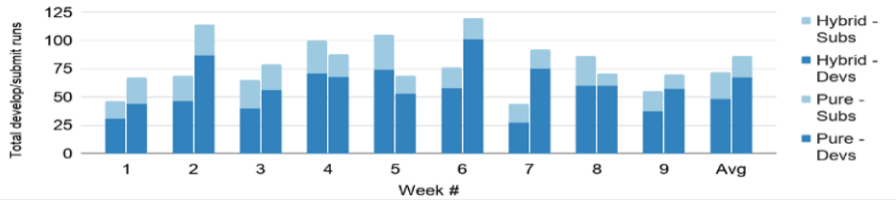

Figure 3: Activity run results: The pure C++ group (avg. 48dev / 24sub) develops less and submits more than the hybrid Coral/C++ group (avg. 67dev / avg. 16 sub).

To calculate students' average start date each week, we found each students' earliest activity timestamp, calculated the difference between that and the due date, and averaged the differences.

Results: Figure 4 displays our results. The number of days are on the y-axis and the week number is on the x-axis. A total start date average column and an adjusted total average column is added at the end of the chart to account for a 'grace period' (late submissions allowed) during weeks 1 and 2.

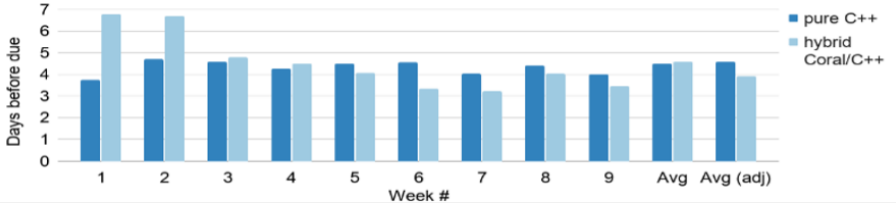

Figure 4: Start date results: The pure C++ group (avg. 4.5days / 4.8days adj.) begins working earlier than the hybrid Coral/C++ group (avg. 4.6days / 3.9days adj.)

Figure 4 shows that both groups begin working about 4.5 days before the due date. Removing weeks 1 and 2 to account for the 'grace period', Figure 4 shows that the pure C++ students begin 4.6 days early whereas the hybrid Coral/C++ students begin 3.9 days early (see 'Avg (adj)' column).

4.4 Pivoting

A pivot is when a student switches from one lab activity to another without completing (scored 100%) the current one they are working on. Pivoting enables students to score additional points when stuck or even use another lab activity to help them solve the current problem they are facing.

Results: Figure 5 displays our results. The total number of pivots are on

the y-axis and the week number is on the x-axis. A total pivot average column and total pivot average adjusted column is added at the end of the chart to account for the midterm in week 6.

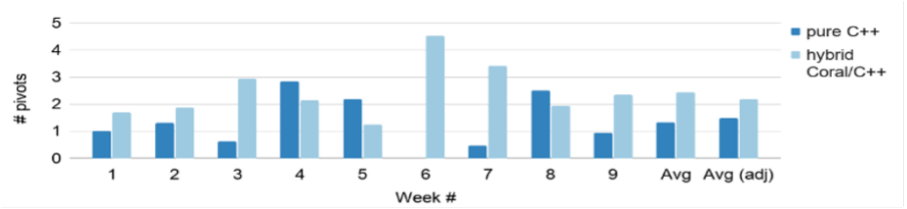

Figure 5: Pivot results: The hybrid Coral/C++ group (avg. 2.4 / 2.2adj.) pivots more than the pure C++ group (avg. 1.3 / 1.5adj.) each week.

Figure 5 shows that the hybrid Coral/C++ group (2.4) pivots more frequently each week than the pure C++ group (1.3). Even after removing week 6 from the calculations to account for the midterm, the hybrid Coral/C++ group (2.2) still pivots more than the pure C++ group (1.5).

5 Conclusion

In this paper, we shared our experience using a hybrid Coral/C++ MSP teaching approach in our CS1 class. We found that using a hybrid Coral/C++ approach did not harm student grade performance. We found that both groups spent a healthy amount of time working on lab activities. We saw that students in the hybrid group developed their code more and submitted their code less frequently than the pure C++ group. Both groups start working about 4 days before the deadline and both groups make good use of pivoting. This work is not meant to conclude that one teaching approach is better, but rather to show that both approaches work. Using a Coral/C++ approach to begin a CS1 class does not harm students but can offer benefits such as having an easier time teaching programming fundamentals when the class begins. As such, we will likely continue using this approach in our CS1, and we encourage others to try this approach as well.

References

[1] Coral. https://corallanguage.org/ Accessed: August, 2020.

[2] Scratch. https://scratch.mit.edu/ Accessed: August, 2020.

[3] Snap. https://snap.berkeley.edu/ Accessed: August, 2020.

[4] Joe Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. An analysis of using many small programs in cs1. pages 585–591, 02 2019.

[5] Joe Michael Allen, Frank Vahid, Kelly Downey, , and Alex Daniel Edgcomb. Weekly programs in a cs1 class: Experiences with auto-graded many-small programs (msp). In *2018 ASEE Annual Conference & Exposition*, number 10.18260/1-2–31231, Salt Lake City, Utah, June 2018. ASEE Conferences. https://peer.asee.org/31231.

[6] Jens Bennedsen and Michael E. Caspersen. Failure rates in introductory programming: 12 years later. *ACM Inroads*, 10(2):30–36, April 2019.

[7] Susan Bergin and R. Reilly. The influence of motivation and comfort-level on learning to program. In *PPIG*, 2005.

[8] A. Edgcomb, F. Vahid, and R. Lysecky. Coral: An ultra-simple language for learning to program. *ASEE Annual Conference and Exposition, Conference Proceedings*, 2019.

[9] Päivi Kinnunen and Lauri Malmi. Why students drop out cs1 course? In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, page 97–108, New York, NY, USA, 2006. Association for Computing Machinery.

[10] Christopher Watson and Frederick W.B. Li. Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, ITiCSE '14, page 39–44, New York, NY, USA, 2014. Association for Computing Machinery.

Progression Highlighting for Programming Courses*

Nabeel Alzahrani and Frank Vahid
Department of Computer Science and Engineering
University of California, Riverside
nalza001@ucr.edu, vahid@cs.ucr.edu

Abstract

New program auto-graders can provide a log file having an entry for each student code run, either during development or when submitting for points. Using that log file as input, we introduce "code progression highlighting" for instructors to gain visibility into a student's programming process. For any student, an instructor can view the student's program for every run, highlighted to show changes from the previous run (the "progression"), and with statistics per entry like time spent, characters changed, and current score. The progression highlighter opens several new opportunities, like aiding instructors in helping students during office hours, allowing awarding points for good process (starting early, developing incrementally, etc.), detecting some cheating not detectable by similarity checkers, and helping discover where students are struggling.

1 Introduction

Cloud-based commercial auto-graders continue to be adopted in more college programming classes. Previously, most instructors graded programs manually. Some schools used custom-built auto-graders, or used the freely-available Web-CAT tool [9]. In recent years, cloud-based commercial auto-graders have appeared, such as zyBooks [6], Gradescope [16], Mimir [2], Vocareum [5], CodeLab [4], and MyProgrammingLab [3]. For example, zyBooks' auto-grader was released in 2016 and reports that in 2020 the tool was used in 2,000 classes

Student ID	Timestamp	Code Link	Run	Score
102	1/1/20 9:20:03	URL1	Dev	
102	1/1/20 9:21:15	URL2	Sub	4
102	1/1/20 9:30:10	URL3	Sub	5
101	1/1/20 9:32:49	URL4	Dev	
101	1/1/20 9:33:40	URL5	Dev	

Table 1: Example log file from an auto-grader (abbreviated).

by 100,000 students, with surveys indicating most classes previously did manual grading [12]. Such tools give students instant feedback to improve success, while eliminating most manual grading and hence conserving teaching resources.

Cloud-based auto-graders can record every program submission run, including the program source code, the date/time, and score. In fact, some auto-graders provide a built-in IDE (integrated development environment) where students develop and test their programs with their own input (vs. the instructor's test cases as in a submission run), and thus every development run can also be recorded. Such recording provides new opportunities of giving instructors a view into how their students develop their programs. We use the zyBooks program auto-grader (aka 'zyLabs'), which recently began providing instructors with a log file, as in Table 1 (abbreviated). Each entry includes the student's ID number, a timestamp, a URL link to the run's source code, the type of run (develop or submit), the score received (for submit runs), and more (not shown). A real class' log file may have thousands of entries.

With the new availability of such log files, this paper describes a tool to provide instructors with insight into a student's programming behavior.

2 Progression Highlighting

Seeing only a student's final program submission yields no information on the student's programming process, such as when the student started (early or late), whether the student develop incrementally (vs. writing the entire program and then running it), how the student debugged (methodically or haphazardly), and more. Instructors have long wanted more insight into the student programming process, so they can better help students, provide points for good process, see where students are struggling, and more.

To achieve such insight, some instructors have recently begun having students maintain their code on GitHub, so instructors can see the code history. But this approach has the drawback of requiring use of GitHub, which can be a bit much for introductory classes.

Instead, auto-grader log files provide a unique and new opportunity to provide more insight to instructors. In our experience, one of the most desired pieces of insight is a quick view of how the student's program changed over time, what we call the 'progression' of the program. Some tools, like that from zyBooks, indeed make the source code for every run available to an instructor. However, the difference between runs can be hard to see. Below, an instructor may not notice the difference between the two program runs of a given student (the code is abbreviated for space).

`if ((x > y) && (y > z))`
`  cout << x;`
`else if ((y > x) && (y > z))`
`  cout << y;`
`else`
`  cout << z;`
	1/1/20 9:20:03

`if ((x > y) && (x > z))`
`  cout << x;`
`else if ((y > x) && (y > z))`
`  cout << y;`
`else`
`  cout << z;`
	1/1/20 9:21:15

Table 2: Instructors cannot easily see differences in successive runs.

Thus, we created a tool to highlight changes between successive runs. Fortunately, a library for highlighting text differences is available in Python language, so we made use of that library [1]. We add key statistics next to each entry. A simplified version of our progression highlighter appears below.

The table is for student 102 from Table 1. The first column shows each run's program code, with insertions/changes from the previous run highlighted in yellow. The next column shows whether the run was a develop or submit, followed by the score (for submits). The next indicates time in minutes since the previous run, followed by total elapsed time (gaps of >10 minutes are assumed breaks and ignored), and then the timestamp. The last column summarizes differences, showing number of insertions, changes, and deletions. Not shown is a column showing the previous run's code with deletions highlighted in red.

Our tool takes a log file in a format similar to Table 1, which is the format from zyBooks (and hence immediately usable by thousands of classes that already use zyBooks), but other commercial or custom auto-grader can have their log files auto-converted to that format for importing to our tool as well.

Code	Run type	Score	Min. since prev	Total min	Time-stamp	Ins(+) Change(^) Del(-)

`if ((x > y) && (y > z))`
`  cout << x;`
`else if ((y > x) && (y > z))`
`  cout << y;`
`else`
`  cout << z;`
	Dev		0	0	1/1/20 9:20:03	

`if ((x > y) && (x > z))`
`  cout << x;`
`else if ((y > x) && (y > z))`
`  cout << y;`
`else`
`  cout << z;`
	Sub	4	0.5	0.5	1/1/20 9:21:15	^1

`if ((x >= y) && (x > z))`
`  cout << x;`
`else if ((y > x) && (y > z))`
`  cout << y;`
`else`
`  cout << z;`
	Sub	5	6.9	7.4	1/1/20 9:30:10	+1

Table 3: The progression highlighter's output (simplified).

Highlighting, and some statistics, are available from program version control systems like Git [7], Subversion [11], CVS [13], and Mercurial [14]. Our approach doesn't require use of such version control tools; students simply develop and submit using their class auto-grader. Also, our statistics are specifically intended for instructors, so include items not found in most version control systems. And, we estimate time spent, which is usually lacking in version control systems.

3 Applications and Experiences

The progression highlighter can be applied by instructors in many ways, some of which we have begun doing in our own classes. Note: The progression highlighter works best if students do all development in the auto-grader's IDE. But, for courses that want students using an external IDE, instructors can require frequent submissions, such as every 20 minutes or every 20 lines of code (for example).

One application is to bring up a student's highlighted progression for a

student who has come to office hours in need of help. The progression gives the instructor a powerful view of the student's programming process. Did the student start early or late? Are they developing incrementally or writing large pieces of code all at once? Are they testing their created functions first or just using them untested in their larger code? A related application is to enable instructors to give points for good programming process, such as for starting early, developing incrementally, or testing thoroughly – all readily visible via observing a student's progression (and potentially automatable as well).

Another application is cheating detection or prevention. An instructor can sort students by time spent, and investigate progressions for students who spent little time, to see if they simply copy-pasted a solution. By showing students the progression highlighter tool in class, and requiring frequent develop/submits using the auto-grader, students may be less tempted to copy-paste a solution. Also, with a MOSS-like [15] similarity checker that flags similar code, instructors can use the progression highlighter to investigate further. We have found some cases where for a flagged student pair, the first student has a normal progression leading to full points, and then a second student submitted that same code soon after – increasing confidence of cheating. But we've seen cases where the flagged students had independent progressions that led to similar code, decreasing confidence that cheating occurred (especially for smaller programs where similar solutions may be common).

Yet another application is to help us to determine what coding mistakes caused students to struggle. Previous research described the most common mistakes [8, 10], but just because a mistake is common does not make the mistake bad; much learning comes from making and then fixing mistakes. In contrast, some mistakes cause students to struggle, which means the student cannot find their error, resulting in numerous runs, excessive time spent, and frustration. Frustrated students are more likely to resort to cheating, or to give up on the assignment, which may eventually lead to failing grades or dropping the class as well.

We experimented with detecting such struggle using the progression highlighter. For each of 5 selected programming assignments that we assigned in Spring 2017, we uploaded the auto-grader's log file into our tool. We sorted by time, which is one measure of struggle for the students who spent more than 30 minutes. We discovered the errors in Table 4 were common logical errors that caused struggle.

Using the tool, we only needed tens of minutes per program to find common errors. In contrast, in the previous summer we did a similar analysis but that took us several weeks, since we had to manually examine code without the benefit of the progression highlighter, which not only took longer but was also tedious.

Programming assignment	Errors
Brute force equation solver	• Typos in formula
Convert to binary - functions	• Data type conversion • Return value • String indexing • Loop counter
Palindrome	• Lack of plan • String indexing
Count characters	• Lack of plan • Using cin() instead of getline() • String indexing
Leap year - functions	• If vs. If-else

Table 4: Common logical errors among strugglers, found in just tens of minutes.

4 Conclusions

This paper introduced a tool for instructors of programming classes that takes as input a log file from a program auto-grader, and then for any student, provides a table showing the student's progression through the programming process. Each table entry shows each run's code, with highlights showing insertions/changes, plus statistics for that code like time spent, number of additions/deletions, and score. The tool enables new capabilities for instructors, like providing better help, giving points for good process, detecting (and even better, preventing) some cheating, and determining common causes of struggle. We look forward to seeing how instructors use this tool to improve their classes and to conduct future research.

References

[1] difflib. `https://docs.python.org/3/library/difflib.html`. Accessed: 12/01/2020.

[2] Mimir. `mimirhq.com`. Accessed: 12/01/2020.

[3] Pearson: Myprogramminglab. `pearsonmylabandmastering.com`. Accessed: 12/01/2020.

[4] Turing's craft: Codelab. `turingscraft.com`. Accessed: 12/01/2020.

[5] Vocareum. `vocareum.com/home/programming-lab`. Accessed: 12/01/2020.

[6] zybooks. `zybooks.com`. Accessed: 12/01/2020.

[7] *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Inc., 2012.

[8] R. C. Bryce, A. Cooley, A. Hansen, and N. Hayrapetyan. A one year empirical study of student programming bugs. In *2010 IEEE Frontiers in Education Conference (FIE)*, pages F1G–1–F1G–7, 2010.

[9] Stephen H. Edwards and Manuel A. Perez-Quinones. Web-cat: Automatically grading programming assignments. *SIGCSE Bull.*, 40(3):328, June 2008.

[10] Andrew Ettles, Andrew Luxton-Reilly, and Paul Denny. Common logic errors made by novice programmers. In *Proceedings of the 20th Australasian Computing Education Conference*, ACE '18, page 83–89, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Louis Glassy. Using version control to observe student software development processes. *J. Comput. Sci. Coll.*, 21(3):99–106, February 2006.

[12] Chelsea Gordon, Roman Lysecky, and Frank Vahid. The rise of the zylab program auto-grader in introductory cs courses. *Whitepaper*.

[13] Keir Mierle, Kevin Laven, Sam Roweis, and Greg Wilson. Mining student cvs repositories for performance indicators. In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, page 1–5, New York, NY, USA, 2005. Association for Computing Machinery.

[14] Daniel Rocco and Will Lloyd. Distributed version control in the classroom. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, page 637–642, New York, NY, USA, 2011. Association for Computing Machinery.

[15] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, page 76–85, New York, NY, USA, 2003. Association for Computing Machinery.

[16] Arjun Singh, Sergey Karayev, Kevin Gutowski, and Pieter Abbeel. Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, L@S '17, page 81–88, New York, NY, USA, 2017. Association for Computing Machinery.

vWaterLabs: Developing Hands-On Laboratories for Water-focused Industrial Control Systems Cybersecurity Education*

Stu Steiner[1], Matthew J. Kirkland[2], Daniel Conte de Leon[2]
[1]Computer Science Department
Eastern Washington University Spokane, WA 99201
[2]Center for Secure and Dependable Systems
University of Idaho Moscow, ID 83844
ssteiner@ewu.edu, kirk8182@alumni.uidaho.edu, dcontedeleon@ieee.org

Abstract

The increase in the number of cybersecurity attacks on Industrial Control Systems (ICS) creates an increased demand for qualified cybersecurity professionals. Training qualified professionals for ICS cybersecurity is costly and currently not scalable. In this article, we present vWaterLabs, an easily replicated, and pedagogically sound set of labs for ICS cybersecurity education.

1 Introduction

The need for educational institutions to train cybersecurity professionals, including specialized training for cybersecurity of Industrial Control Systems (ICS), is at an all time high. A recent cybersecurity report shows an increase in cyber-attacks against Ethernet enabled ICS of approximately 30% from 2018 to 2020 [8]. This increasing growth of Ethernet enabled ICS devices creates additional attack opportunities [2], and since educational institutions are struggling to keep up with the demand for cybersecurity professionals a fresh solution is needed.

Currently, most educational institutions have well-defined Computer and Network Security labs [5]; however, educational institutions typically have only minimal instruction of ICS cybersecurity. Furthermore, a set of well-defined ICS labs, similar to the well-defined Computer and Network Security labs, do not exist. This proposed solution seeks to change this paradigm.

1.1 Proposed Solution

We developed vWaterLabs, a fully virtualized set of labs for ICS cybersecurity education. vWaterLabs was implemented as effective, economical, shareable, and widely available educational resources for hands-on ICS cybersecurity education and training with the following objectives: (1) Create replicable, easily deployable, and cost-effective hands-on learning; and (2) Satisfy the Centers for Academic Excellence in Cyber Defense Education Industrial Control Systems Knowledge Units through pedagogically current labs. The labs are found at `https://github.com/ICSSecurityLabs`

1.2 Contribution

The contributions of this article are: (1) Pedagogical analysis of the knowledge and skills needed for water-focused ICS cybersecurity practitioners; (2) Design considerations and decisions for easily reproducible labs for water focused ICS cybersecurity education; and (3) Introduction of Programmable Logic Controllers (PLC) programming and Modbus injection and mitigation lab that uses the vWaterLabs virtual ICS testbed.

A separate article, titled *vWaterLabs: Design and Characteristics of a Virtual Testbed for Water-focused ICS Cybersecurity Education*, describes the following contributions: (1) Analysis of the characteristics of educational and water focused ICS testbeds; (2) Design details of a virtual and cost effective testbed for ICS cybersecurity education focused on water systems; and (3) Introduction of an ICS vulnerability assessment lab implemented using the vWaterLabs virtual testbed.

Our ultimate goal is that vWaterLabs will provide instructors and students hands-on understanding of the approaches, techniques, and tools used to protect today's ICS systems related to water processing.

1.3 Overview of this Article

The rest of this article is organized as follows: Section 2 describes the pedagogical foundation required for ICS cybersecurity education. Section 3 describes the design of the lab framework. Section 4 introduces one ICS cybersecurity lab. Section 5 presents the conclusion and future work related to vWaterLabs.

2 Pedagogical Analysis of the Knowledge and Skills for Water-Focused ICS Cybersecurity

To define the pedagogy required for vWaterLabs, a search was conducted using Google Scholar, IEEE Explore, and the ACM Digital Library, for papers between 2010 to 2020. The search phrases included: **["ics education" "industrial control systems education" "ics labs" "industrial control systems labs"]**. The search resulted in a total of 50 papers, and after review five papers were selected.

Of the five papers selected, three primary student learning objectives were identified. Antonioli et al., Čeleda et al., Green et al., and Sitnikova et al. [1, 9, 4, 7], discussed teaching fundamental cybersecurity concepts. Gao et al. and Čeleda et al. [3, 9] discussed teaching ICS fundamentals. The identified learning outcomes, with corresponding concepts, include:

- **Cybersecurity Fundamentals**
  - Confidentiality, Integrity, Availability (CIA)
  - Authentication, Authorization, Accounting (AAA)
  - Network traffic, monitoring tools, firewalls, malware, intrusion detection, cryptography, certificates, digital signatures, threats, etc.
- **ICS Hardware and Programming**
  - Supervisory Control and Data Acquisition (SCADA), PLCs
  - Ladder Diagram (LD), Function Block Diagram (FBD)
- **ICS Protocols**
  - Modbus, Distributed Network Protocol (DNP3), Process Field Bus (PROFIBUS), Process Field Net (PROFINET), BACnet

3 vWaterLabs Characteristics for Virtual Labs

vWaterLabs was designed based on a real-world ICS wastewater treatment facility using the ANSI/ISA-99 reference model. The lab exercises require:

- **Zones and Network** - Antonioli et al. and Čeleda et al. [1, 9], discussed integrated labs exercises containing Ethernet connected zones, including:
  - **Level 0**: Production I/O Network;
  - **Level 1**: Control Network of PLCs;
  - **Level 2**: Supervisory Control Network with Human Machine Interface (HMI);
  - **Level 3**: Corporate Network.
- **ICS Protocols** - The labs will allow students hands-on experience with multiple ICS protocols.
- **Virtualization** - The labs will be virtual, allowing for exercise replication and synchronous participation of multiple individuals.

- **Bloom's Taxonomy** - The labs will be based on Bloom's Taxonomy. Plumley and Yardley et al. [6, 10] state `Create`, `Evaluate`, `Analyze`, and `Apply` require hands-on activities, where `Understand` and `Remember` occur via lecture-oriented teaching.

4 vWaterLabs Hands-On Lab: PLC Programming and Modbus Injection

Multiple ICS cybersecurity labs were built to create a foundational set of labs similar to the Syracuse SEED Labs [5]. Each lab requires approximately two hours to complete and is a combination of lecture with hands-on application. These labs are best completed as part of a virtual testbed.

This lab's learning objectives include: (1) Understand PLC and HMI components; (2) Introduce PLC programming; (3) Use Wireshark to understand ICS traffic; (4) Understand the Modbus protocol; and (5) Perform a Modbus injection attack.

This lab provides an introduction to ICS and PLCs. First, students learn PLC logic and programming through a series of progressive exercises. This exercise starts with basic LD programming and progresses to controlling multiple inputs and outputs. Figure 1 illustrates a LD challenge. The challenge states *Given all I/O is in the '0' state, except for input, what will the value of output be after this LD is done running?*

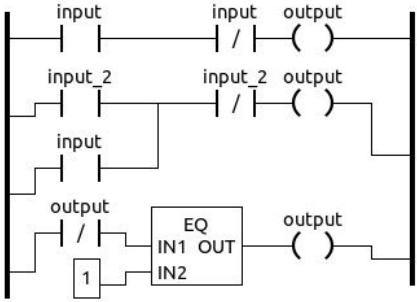

Figure 1: Ladder Diagram Programming Example

Next, the lab focuses on understanding both ICS network traffic and the Modbus protocol. Figure 2 illustrates the network configuration for this lab, which contains three virtual machines configured on the same subnet, and disconnected from the internet. The students use Wireshark to examine packets for ICS network traffic. Based on this traffic, there are a series of progressive questions that the students must answer to demonstrate their understanding of the ICS network traffic and the Modbus protocol.

Once the students have demonstrated an understanding of the ICS network traffic and the Modbus protocol, the students move on to using pyModbus and

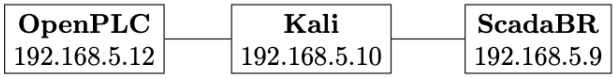

Figure 2: The Virtual Machines Network Configuration

IPython to read setpoints from a provided system. Similar to previous portions of the lab, students must demonstrate an understanding through a series of lab questions, and a series of values that must be displayed to the HMI.

Based on the student's understanding of pyModbus and IPython, the students are asked to program an injection attack that turns on a tank overflow light regardless of the tank level. Finally, the students complete the lab by writing a summary lab report.

5 Conclusion and Future Work

This article introduced vWaterLabs: Developing Hands-On Laboratories for Water-focused Industrial Control Systems Cybersecurity Education. We presented the design and pedagogy for a set of ICS cybersecurity educational labs. We also introduced a single hands-on lab, to illustrate the content contained in our foundational set of ICS cybersecurity educational labs.

Currently, the students download and configure multiple virtual machines before the lab can be completed. Future work will include moving these virtual machines to headless Docker containers, which should remove the configuration problems currently experienced by some students.

Acknowledgments

This work and the used computing infrastructure were partially funded by an Idaho IGEM grant (IGEM17-001), the U.S. National Science Foundation (NSF) CyberCorps® award 1565572, and the M.J. Murdock Foundation. The opinions expressed in this paper are not those of the NSF, the M.J. Murdock Foundation, or the State of Idaho.

References

[1] Daniele Antonioli and Nils Ole Tippenhauer. Minicps: A toolkit for security research on cps networks. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*, CPS-SPC '15, pages 91–100, New York, NY, USA, 2015. Association for Computing Machinery.

[2] Mario Ayal, Rob Cantu, Richard Holder, Jeff Huegel, Niten Malik, Michalina M, Adrienne Raglin, Ashley Reichert, Ash M. Richter, and Kimberley Sanders. The industrial internet of things (iiot): Opportunities, risks, and mitigation, Sept. 2020.

[3] Haihui Gao, Yong Peng, Kebin Jia, Zhonghua Dai, and Ting Wang. The design of ICS testbed based on emulation, physical, and simulation (EPS-ICS Testbed). *Proceedings - 2013 9th International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIH-MSP 2013*, pages 420–423, 2013.

[4] Benjamin Green, Anhtuan Le, Rob Antrobus, Utz Roedig, David Hutchison, and Awais Rashid. Pains, Gains and PLCs: Ten Lessons from Building an Industrial Control Systems Testbed for Security Research. *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET '17)*, pages 1–8, 2017.

[5] M. J. Kwon, G. Kwak, S. Jun, H. Kim, and H. Y. Lee. Enriching security education hands-on labs with practical exercises. In *2017 International Conference on Software Security and Assurance (ICSSA)*, pages 100–103, 2017.

[6] Evan G. Plumley. A framework for categorization of industrial control system cyber training environments. Master's thesis, U.S. Air Force Institute of Technology, March 2017.

[7] Elena Sitnikova, Ernest Foo, and Rayford B. Vaughn. The power of hands-on exercises in scada cyber security education. *IFIP Advances in Information and Communication Technology*, 406:83–94, 2013.

[8] Symantec Corporation. 2020 internet security threat report. Online, March 2020.

[9] Pavel Čeleda, Jan Vykopal, Valdemar Švábenský, and Karel Slavíček. Kypo4industry: A testbed for teaching cybersecurity of industrial control systems. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, pages 1026–1032, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Tim Yardley, Suleyman Uludag, Klara Nahrstedt, and Pete Sauer. Developing a smart grid cybersecurity education platform and a preliminary assessment of its first application. In *Proceedings of the Frontiers in Education Conference 2014*, pages 1–9. IEEE, February 2014.