UNIVERSITY OF CALIFORNIA
RIVERSIDE

A Many Small Programs (MSP) Approach in a CS1 Course

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Joe Michael Allen

June 2021

Dissertation Committee:

- Dr. Frank Vahid, Chairperson
- Dr. Stefano Lonardi
- Dr. Paea LePendu
- Dr. Tony Givargis

The Dissertation of Joe Michael Allen is approved:

_____

_____

_____

_____

Committee Chairperson

University of California, Riverside

ACKNOWLEDGEMENTS

First and foremost, I give praise, glory, and thanks to the Lord for blessing me with all the opportunities I have had and for giving me the strength and the patience to complete this work.

I give thanks to my mom, Lisa Allen, who is one of my greatest supporters throughout this work and over my entire life. Being a single mom is not easy, but your strength, sacrifice, and devotion to me and the family is something I will always admire. Your love and support is unconditional and I am grateful to God to be your son. I strive to be more like you every day. Love you mom!

To my beautiful wife Mrs. Ashley Allen. You are my rock, my joy, my love, and my life. You show me so much love and so much support every day. I would not have been able to complete this journey without you. You make each day worth living, thank you for always being there for me and making me smile during the rough days. I love you so much.

To my brother Christopher Allen, our walking miracle. It is my honor to be your brother. You may not know it, but I look up to you so much and admire you in all you do. You are one of a kind and I love you so much. Thank you for always being there for me. Thank you to all my other siblings as well: Alex Estevez, Aaron Allen, Presley Allen, and Grace Allen. I am blessed to be your brother and I am so proud of the lives you are all leading. Your love and support have always been with me.

To my Gaga Nora Allen, Papa Joe V. Allen, and Margie Briseno. Though you may be my grandparents, I will always look to you as my second set of parents. You have raised me, loved me, and nurtured me into the man I am today. If I did not have you in my life, I would surely be lost. Thank you for all the support you've given me all my life. I love you all forever until the sun don't shine.

A mi suegra Ana Martínez, mi tío político Edgar Martínez, mi cuñado Andy Polanco y mi primo el pequeño Edgar. Estoy muy bendecido de ser parte de la familia (¡oficialmente ahora!) Y muy agradecido por todo el amor y el apoyo que me han brindado. Me han mostrado tanta amabilidad, ánimo y calidez desde el primer día que los conocí. Los amo mucho a todos y estoy muy agradecido con Dios por el crecimiento de mi familia.

I give thanks for all the support, guidance, and opportunity given to me by my Ph.D. advisor Dr. Frank Vahid. You have always been a role model to me. Thank you for teaching me critical thinking skills and how to write an email :) I will always admire your passion and devotion for others - especially the students that you teach, including myself.

Thank you to my committee members: Dr. Stefano Lonardi, Dr. Paea LePendu, and Dr. Tony Givargis for supporting me in this work and providing helpful feedback that shaped this work.

I give thanks to my best friend Rishi Naik. I am so blessed by your friendship and all the wisdom you have shared over years. I have always looked up to you and admire you a ton. Thank you for putting up with me, like almost every day, but I appreciate all our time together these past 10 years and looking forward to a lifetime more.

I would like to thank the Association of Computing Machinery (ACM), the American Society for Engineering Education (ASEE), the Consortium for Computing Sciences in Colleges, and the Institute of Electrical and Electronics Engineers (IEEE) organizations for providing constructive criticisms and opportunities for publishing my works.

Finally, I thank the University of California, Riverside (UCR) for providing me with an education and opportunities such as this Ph.D., all my colleagues over the years that have knowingly and unknowingly supported me, and anyone else I forgot to mention or do not have room to mention. God bless.

DEDICATION

I dedicate this work to my mom, Lisa Allen. You are my rock and everything I hope to be one day. Words on this page could never express the gratitude and love I have for you. You have been by my side every day of my life and continue to support me in every step I take, giving everything you have to me, my brother, and the family. Without you, none of this would be possible, I mean this with all of my heart. You deserve every blessing God can give and I hope this small token of gratitude can express how much you mean to me. Love you mom.

I dedicate this work to my grandparents Nora Allen, Joe V. Allen, and Margarita Briseno. I am not sure what I did to deserve you, but I will be sure to thank God when I see Him. I hope you know how much I love you and how thankful I am for all the love you have given me and for raising me as your own son. Life has not always been easy for us and the family, but with the support and devotion from you, we have all remained strong and united. There is nothing I can do to repay you back for all the blood, sweat, tears, and sacrifices you have made for me and the family, but may this small token of gratitude serve as a reminder of my love for you.

I dedicate this work to my wife Ashley Allen. It's crazy thinking back and realizing that in the timespan of my Ph.D., we have dated, become boyfriend and girlfriend, gotten engaged, and then gotten married. Wow! You are my everything Love and I hope I prove that to you each and every day. You are my strength, my support, my joy, my happiness, my everything. We have been through so much together and we are

just starting our life together. Your support and your love mean everything to me and I look forward to growing old with you. I love you so much my Ashley. May this small token of gratitude show you how much you mean to me and how thankful to God I am to be your husband.

I dedicate this work to my uncle Michael Postich. You were taken away far before your time uncle. I am sad to say we didn't know each other as well as I would have liked, but I do have fond memories of us on camping trips and white-water rafting. In your final days, you showed us all so much strength, love, and kindness to all those around you. Even through your suffering, you were always the devoted and loving husband to my aunt and the kind, gentle, and goofy brother to my mom. You are an inspiration and although you are no longer with us to share with me in this moment of success, may this small token of gratitude reach you in heaven where you are at peace and enjoying the company of our Lord. Rest in peace Uncle Michael, we love you and we will see you again soon.

ABSTRACT OF THE DISSERTATION

A Many Small Programs (MSP) Approach in a CS1 Course

by

Joe Michael Allen

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2021
Dr. Frank Vahid, Chairperson

A well-run introductory CS1 course is essential for all students within CS education. CS1 is necessary to keep students in the major and important to attract non-majors to the CS field. Unfortunately, there are many well-known issues that most CS1 courses have in common: high drop rates, low retention, high student stress, student struggle, academic dishonesty, and low grades. In this work, we aim to address these issues and seek to improve CS1 courses by focusing on weekly programming assignments. Our work introduces a different teaching approach from the traditional One Large Program (OLP) teaching approach, to a Many Small Programs (MSP) teaching approach. Instead of assigning students one large programming assignment to complete each week, the MSP approach involves assigning students multiple smaller programming assignments, for example 5-7 programs each week, instead. Such an approach has become more feasible with the advent of program auto-graders with immediate feedback to students, partial credit, and resubmit capabilities. In this dissertation, we discuss the

conception of the MSP approach, provide insight into the process of transitioning from an OLP approach to an MSP approach, discuss various benefits that an MSP approach offers, discuss some pros and cons for using such an approach, present results from surveys and multiple analyses on various metrics related to an MSP approach, and discuss future use and improvements to the current MSP approach and tools used to analyze student interaction. This work shows that an MSP approach can lead to reduced student stress, can improve student grade performance, finds students making good use of the benefits an MSP approach offers, and shows that students are still well prepared for a CS2. Finally, we introduce a tool for instructors to upload their own MSP data sets to gain deep insight into their own students' behavior when using an MSP approach in their own classes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1. Introduction of Dissertation

1.1 Improving CS1

Student success in introductory programming courses (known as CS1) is critical to keeping students in the computer science (CS) major, training students in other majors who need some programming experience, and attracting students to CS. Unfortunately, CS1 courses have many well-known issues: high drop rates, low retention, high stress, academic dishonesty, and low grades [34][47][8][9]. In 2005, Beaubouef and Mason [7] reported that drop rates between 30%-40% is the norm for most CS programs. Similarly, in 2014 Watson and Li [67] reported that over a 30-year period between 1979 and 2013, CS1 courses have an average 30% non-passing rate. Figure 1.1 summarizes their findings. These issues have drawn the attention of education researchers to find ways to improve CS1.

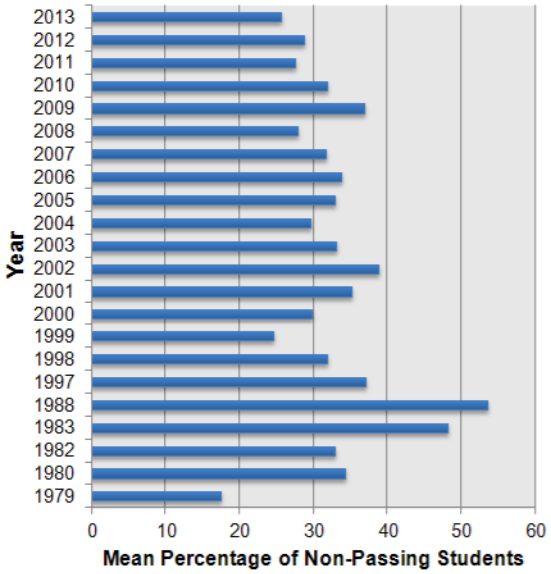

Figure 1.1: 2014 study by Watson and Li found that CS1 classes have a 30% non-passing rate over a 30-year period from 1979 - 2013 [67].

CS educational research is vast and as such, this dissertation focuses on one small aspect of CS1: weekly programming assignments. Weekly programming assignments form a large portion of the CS1 curriculum and thus have a high impact on students' experience in CS1. This dissertation introduces a Many Small Programs (MSP) teaching approach meant to improve the student experience while maintaining good student grade performance.

The following chapters consist of:

- A description of a Many Small Programs (MSP) teaching approach including details of creating, using, and analyzing MSP programming assignments.
- An analysis of student grade performance and student stress after being taught CS1 via an MSP approach.
- An examination of factors such as time spent, start time, pivot usage, student workflow, CS2 performance, and more as a result of using an MSP approach in CS1.
- An introduction of student workflow charts and how to understand student behavior on weekly programming assignments.
- An introduction of tools for other instructors to gain insight on their students' interaction and performance with weekly MSP assignments.

1.2 Background on CS1 Educational Research

College-level CS1 education researchers are focused on finding ways to improve CS1 courses for students. Section 1.2 introduces and discusses seven CS1 research areas currently being analyzed.

1.2.1 Pair Programming

Researchers have examined how student collaboration and instruction affects the student experience. Pair programming is when at least two students collaborate on a single program; working on the same design, algorithm, etc. Pair programming offers benefits such as community building, stress reduction, active learning, and increased student support. Nagappan et al. [44] researched pair programming in their introductory programming courses and found an increase in the retention of students in CS, reduced burden on students since pairs help each other, and no loss in student performance. Rodriguez et al. [55] examined how pair programming and student collaboration affected learning outcomes, finding that if pair programming is done properly, collaboration increases learning and understanding. Blaheta [11] studied cooperative learning and found that students had a positive reaction. Simon et al. [51] found that peer instruction had a positive impact on student perception of learning. Porter and Simon [48] taught CS1 using three practices, media computation, pair programming, and peer instruction. They found an improvement in student retention in CS and a decrease in drop/fail rates. Porter et al. [52] reports that pair programming, contrary to the claim that some students may fail to learn since their partners do all the work, actually improves student grade performance. Williams et al. [70] found that pair programming leads to higher grades on

programming assignments, exams, and in the class overall. An added benefit is that pair programming also reduces the stress on instructors and teaching assistants since students get help from their peers. Work [37][53][59] continues to study the effects of peer collaboration, finding many benefits.

1.2.2 Peer Instruction

Peer instruction is a teaching approach where students engage in small group discussions and answer featured questions. This method adds interactivity to a traditional lecture and has shown increased student engagement in class. Porter et al. [49] evaluated 10 years of instruction, looking at 16 courses that used peer instruction and concluded that peer instruction reduced the failure rate by 61%. Porter et al. [50] performed a similar study in upper division classes and found peer instruction did help students, showing that peer instruction is beneficial even in upper level classes. Simon et al. [60] performed a study of peer instruction in CS1 and CS1.5. Students were given clicker questions, then they answered individually, discussed with their peers, and answered again. The results found a 40% increase in the correct answer and found students felt peer instruction was valuable. Zingaro [71] notes that peer instruction increases self-efficacy in students while being enjoyable to both students and instructors.

1.2.3 Game Design and Gamification

Gamification is defined as "the use of game design elements in non-game contexts" [19]. This could include having students play games in class to learn, creating programming assignments such that students are coding up a game, adding class leaderboards to increase competition, and more. Cliburn and Miller [13] gave students

three assignments with three project options each. The project options consisted of a game, "choose your own adventure," or a traditional project. 71% of projects submitted were the game option, although this choice did not have any statistical significance on overall grade performance. Leutenegger and Edgington [38] describe using a "game first" approach when teaching CS1. The course began teaching basic CS concepts like variables, loops, conditional statements in flash, then teaching C++ programming with pointers, concluding with graphics using OpenGL. Each assignment/project had a game-like element or animation like moving a ball, creating a simulation, or designing a custom game. Surveys showed that students learned effectively, enrollment in CS courses increased, and retention of students in the game development CS class sequence increased as well. Soh [62] applied gamification to a multiagent system class designed for college seniors and graduate students. Students had to participate in several "game days" where the students would compete against one another in games related to various class concepts. Surveys showed students enjoyed the game days and thought that they were useful to their learning.

1.2.4 Flipped Classroom

The idea of a flipped classroom has been used in classrooms for years already. Bishop and Verleger [10] define the flipped classroom as "an educational technique that consists of two parts: interactive group learning activities inside the classroom, and direct computer-based individual instruction outside the classroom." Instead of most of the learning taking place in the traditional lecture, the flipped classroom inverts this idea and suggests that most of the learning be done outside the classroom. Students are responsible

for doing the reading and primary learning at home, and the classroom will be used for problem solving, group activities, and group discussions. Findlay-Thompson and Mombourquette [24] add that the "intent is to create a more collaborative learning environment where students are focused on working through problems with both the guidance of their teachers and the support of their peers." Giannakos et al. [28] surveyed and summarized 32 peer-reviewed articles on flipped classrooms and reported benefits such as increased student performance, attitudes, and engagement in the class. Students reported to be more engaged with "authentic" learning due to the higher level of problem solving in the classroom; thus student perception of the quality of learning is increased. Additionally, the concept of a flipped classroom can be applied in many ways. Some instructors assign students quizzes to take outside the classroom [22], some make videos for students to watch outside the classroom [22], others give students additional online studying resources for additional learning [27]. Flipped classrooms have shown various benefits including improved performance, fewer drops, and happier students [25][29][42][56].

1.2.5 Programming Language and Applications

Another area of educational research is to change the programming language or the programming applications that are being used to teach CS1. Norman and Adams [45] switched from C++ to Python, and also replaced weekly homework assignments with labs and online problem sets. As a result, they observed an improvement in scores for tests, lab exercises, the final exam, and the overall semester score. Layman et al. [36] note that students are more interested in practical and socially-relevant assignments. As a

follow-up, Layman et al. examined 200 CS1 programming assignments and found that only 34% had a practical or socially-relevant context. Thus, the authors recommend improving the applications of the programming assignments that instructors use. Guzdial [31] changed their CS1 course to focus on media applications. Guzdial describes a new course where students learn programming with Python and create programs for manipulating sound, images, and movies.

1.2.6 AUTOMATED HOMEWORK GRADING SYSTEMS

Most closely related to our work is the increase of automated homework grading systems and the introduction of small coding problems for homework or extra practice. Automated homework systems have benefits such as easy assignment creation and grading, quick and accurate feedback to students, and freeing of instructors' time. Universities and companies have built automated homework grading systems and are studying how to effectively use them [1][18][23][69]. In addition, smaller coding problems are being introduced into classrooms. Systems such as CloudCoder [14], CodingBat [15], Pearson's MyProgrammingLab [46], and Problets [54], help instructors design small coding problems used as homework, warm-up, or extra practice in the classroom.

1.2.7 OTHER

There are many other approaches that are being researched to improve CS1. Studio-based learning [32][33] emphasizes student communication, collaboration, and critical thinking skills; showing improvements in student attitude and content mastery. Denny [17] viewed exam question creation as a tool for learning and had students author

questions that could appear on their exam. Kumar [35] developed online tutors for their course and evaluated student experience, opinions on learning, and other feedback. Edgcomb et al. [21] replaced static textbooks with interactive textbooks at three universities and found substantial improvements in exam scores, project scores, and overall letter grades. Alfaro and Shavlovsky [2] created a system that allows students to submit and collaboratively review and grade homework. They found that students had a higher incentive to turn in higher quality work and homework reviews. Stone and Madigan [63] allowed their students to choose their course projects from a comparable set of alternatives. Allowing students to choose their projects increases student success rate and enhances student perception of understanding the material.

1.3 MSP Teaching Approach Introduction

1.3.1 One Large Program (OLP) Teaching Approach

Traditionally, students are taught CS1 via a one large program (OLP) teaching approach. An OLP approach involves instructors giving students one large programming assignment each week to complete. This approach is manageable for instructors since they only need to create, maintain, and grade a small number of assignments each term. Typically, OLP programming assignments tend to share the following characteristics: they cover many concepts at once, may require students to complete multiple parts, usually have lots of text, and by nature, typically require a solution of 50 to 100 lines of code, or more.

For example, our university has long taught CS1 by using an OLP approach. Figure 1.2 shows the lines of code (LOC) for OLP programming assignments given at

our university during the Spring 2017 quarter. We used 10 OLP programming assignments with an average of 101 LOC with our largest assignment having 210 LOC. For students who have never programmed before, such solution sizes can be intimidating, thus, increasing stress or procrastination.

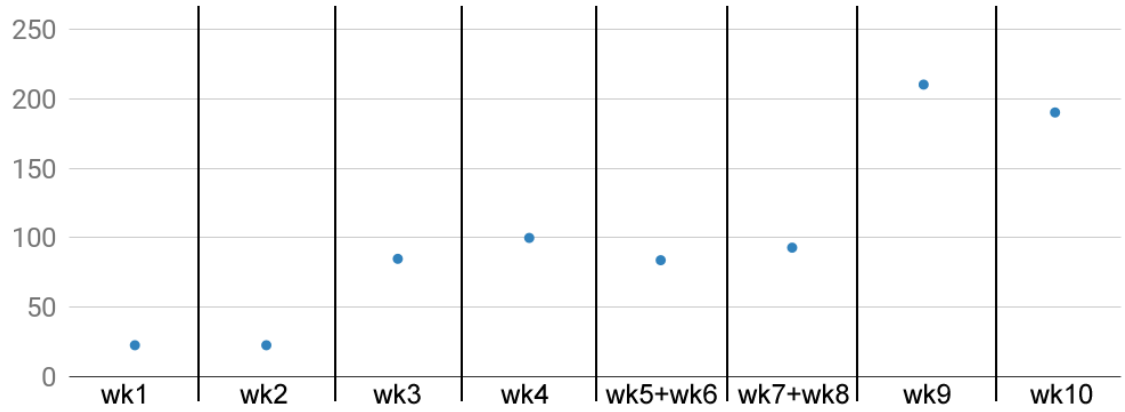

Figure 1.2: Instructor solution LOC for each OLP programming assignment from a CS1 course taught during Spring 2017. Average LOC is 101, max 210.

For our university, and likely for most others, OLP programming assignments in CS1 often account for much of the class grade. Some instructors assign simpler warm-up/practice programs and/or smaller coding homework problems, worth a smaller grade percentage. The weekly OLP programming assignments typically represent the hardest, most-stressful part of CS1 for students, and is also where much of the learning occurs. For this reason, we decided to focus our research on improving the way programming assignments are used in CS1.

1.3.2 Auto-graded Programs

To begin improving weekly programming assignments, we took advantage of modern-day program auto-graders. A program auto-grader automatically runs students' programs against test cases, scoring each program. Many auto-graders let a student

directly submit a program and see score feedback immediately, including which test cases failed, with the student allowed to submit multiple times. Program auto-graders have existed for decades, such as CodeLab [65], Web-CAT [68], and numerous homegrown auto-graders at various universities [6][64]. However, most early auto-graders required high expertise to create new auto-graded assignments, involving specialized scripting. In contrast, modern commercial auto-graders like zyBooks [72], Mimir [41], and Matlab Grader [39] enable creation of new assignments in minutes, entirely via web forms, with no specialized scripting. Due to the ease of use, there has been a dramatic increase in program auto-grader use in CS1 and other courses. For example, since zyBooks' auto-grader was released in 2016, over 250 courses (mostly CS1) have started using an auto-grader that did not before.

1.3.3 MANY SMALL PROGRAMS (MSP) TEACHING APPROACH

The ease of creating and grading new programming assignments in modern auto-graders, coupled with the learning benefit of students getting immediate fine-grained score feedback, motivated us to consider a new teaching approach we refer to as the many small programs (MSP) teaching approach. Instead of assigning students one large programming assignment each week, we assign students multiple smaller programs, or lab activities, each week instead. Typically, we give students between 5-7 lab activities to complete each week. Figure 1.3 is a high-level depiction of an OLP programming assignment versus an MSP programming assignment. Without auto-graders, the extensive resources to grade so many programs deterred instructors from considering an MSP

approach. Or, with older auto-graders, the difficulty of creating and maintaining so many programs also deterred the approach.

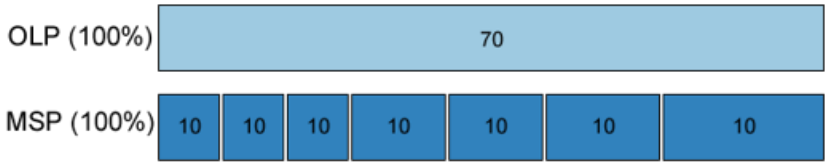

Figure 1.3: High level depiction of an OLP programming assignment versus an MSP programming assignment.

Compared to an OLP programming assignment, MSP lab activities teach one specific concept at a time, are short in size, have short, concise prompts, and tend to have smaller solution sizes. Figure 1.4 shows the lines of code (LOC) for each MSP lab activity used at our university during the Spring 2017 quarter.

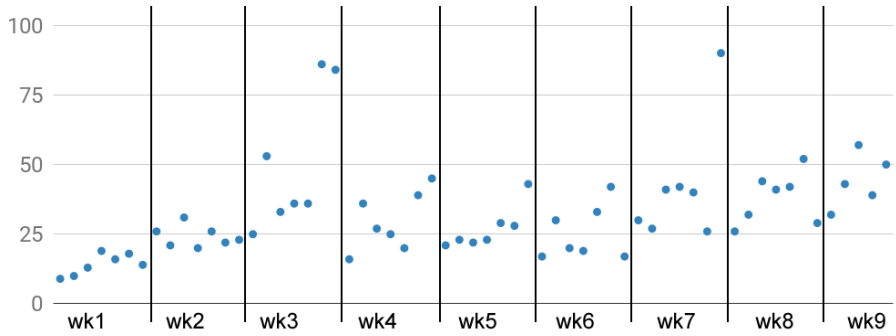

Figure 1.4: Instructor solution LOC for each MSP lab activity from a 2017 CS1 course taught at our university. Horizontal lines have been added to separate weekly MSP assignments. Average is 32, max 90. Outliers in weeks 3 and 7 are due to tall if-elseif-else trees.

MSP lab activities have several other immediate benefits. Students may find them less intimidating due to their small size. There may be less inertia to get started on the smaller programs, with students building confidence from the initial easier programs. Students may benefit from the option of moving on to another program if stuck, then coming back later to finish the incomplete program ("pivoting"). Finally, students get to

have repeated but different learning experiences on the week's concepts (like "loops"), meaning more practice.

1.3.4 MSP Lab Activity Details

For additional insight, Figure 1.5 shows a sample MSP lab activity. Each MSP lab activity consists of a title, a prompt, an instructor solution, and a set of test cases (with assigned points) to grade the submitted code by. The prompt is short and concise; never more than three sentences and contains specific instructions for the student to complete the assignment. Sometimes the prompt can include helpful hints for students as well. Instructor solutions are small and roughly between 10 - 50 LOC total with comments and proper code styling. Test cases are typically input/output tests such that the program expects an exact output for a given input. There is also the option to have unit tests where the auto-grader tests a student's function directly. Point values are assigned to each test case -- we found three to four test cases to be sufficient, typically totaling to 10 points per assignment.

Title:

Count input length without spaces, periods, or commas

Prompt:

Given a line of text as input, output the number of characters excluding spaces, periods, or commas.

Ex: If the input is "Listen, Mr. Jones, calm down." (excluding the quotes), the output is:

21

Solution:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  string userText;
  unsigned int i;
  int charCount;


  getline(cin, userText);  // Gets entire line, including spaces.

  charCount = 0;
  for (i = 0; i < userText.length(); ++i) {
    if ( (userText.at(i) != ' ') && (userText.at(i) != '.') && (userText.at(i) != ',') ) {
      charCount += 1;
    }
  }

  cout << charCount << endl;

  return 0;
}
```

Test Cases:

Compare output (3 points)	Compare output (3 points)	Compare output (4 points)
When input is: Listen, Mr. Jones, calm down.	When input is: Howdy!	When input is: abcd,,,,efgh....ijkl ....
Output exactly matches 21	Output exactly matches 6	Output exactly matches 12

Figure 1.5: Sample MSP lab activity with a title, prompt, instructor solution, and test cases.

1.4 COMMON TERMS USED IN THIS DISSERTATION

An *OLP approach* (One Large Program Teaching Approach) is the traditional teaching approach where students are given one larger programming assignment to complete each week.

An *MSP approach* (Many Small Programs Teaching Approach) involves assigning students multiple *lab activities* (programming assignments) to complete each week. A weekly *MSP assignment* is a grouping of all MSP lab activities assigned to students for the given week. For example, we typically assign students 5-7 MSP lab activities each week, so all 5-7 lab activities collectively form an MSP assignment. Students that have taken a CS1 taught via an MSP approach are considered *MSP-trained* CS1 students.

We primarily use zyBooks' auto-grader for all our weekly MSP assignments. A *develop* is when a student runs their code through zyBooks' compiler for testing without grading and a *submit* is when a student "turns in" their code for grading. An *activity run*, or run, is either a develop or a submit.

Chapter 2. WEEKLY PROGRAMS IN A CS1 CLASS: EXPERIENCES WITH AUTO-GRADED MANY SMALL PROGRAMS (MSPS)

2.1 INTRODUCTION

Weekly programming assignments form a large part of the students' experience in a CS1 course. At our university, our CS1 course followed the traditional one large programming assignment per week (OLP) model with a few small warm-up programs, for over 20 years. Our CS1 serves about 350 students per quarter (including majors and non-majors). We used a program auto-grader for the past 10 years. We also used commercial online auto-graded homework problems (which are even smaller coding exercises) for about 15 years. Students are encouraged to collaborate on the warm-ups and homework problems but are not permitted to collaborate on the weekly OLP assignments. Students may, however, get help during instructor or teaching assistant (TA) office hours or via discussion board posts to the class. We use pair programming at times as well. Our lectures have included small-group collaborative programming (a form of flipped classroom) since the late 1990s. Student evaluations indicate that the course is reasonably well-liked, though many students indicate that the course is hard, time-consuming, and stressful. We also check for and detect overly-similar submissions using MOSS, a system for determining the similarity of programs [43]. Investigating and pursuing academic dishonesty (typically 10-20 per quarter) is a time-consuming and unpleasant part of the instructor's job.

About 10 years ago, faced with increasing enrollments and shrinking funds, we developed an in-house program auto-grader. This dramatically reduced the time TAs spent grading; freeing TAs to spend more time teaching and handling larger class sections. Students also appreciated the immediate score feedback and the ability to resubmit right away for a higher score. The auto-grader did not improve student evaluations, exam performance, or academic dishonesty

In 2016, zyBooks [72] released a web-based program auto-grading system that emphasized ease of use for both students (allows direct coding in the browser or file upload) and instructors (creating new assignments via a simple web interface; requiring no scripting or coding). With this system, any of our instructors or TAs could easily create new assignments with no training. Creating each weekly large programming assignment required only about 60-90 minutes; opposed to many hours in the past. Thus, we created a new set of weekly assignments and warm-up assignments for winter 2016. We continued revising assignments quarterly instead of yearly, or less.

The ease of creating new program assignments coupled with students getting immediate, fine-grained score feedback enabled us to consider a new option. We implemented a teaching approach that involves assigning students multiple smaller assignments each week instead of the traditional OLP approach. We refer to this new method as the many small programs (MSP) teaching approach.

Chapter 2 describes an experiment in which, for one of three class sections, we taught CS1 via an MSP approach and compared survey results and grade performance results to the other two class sections being taught via an OLP approach. We provide

results of student surveys showing significantly happier students. We provide exam results showing improved performance. Based on the results, our department changed all CS1 class sections to use an MSP approach for the following quarter.

2.2 METHODOLOGY

2.2.1 CS1 COURSE DETAILS

The experiment was conducted in our CS1 course at the University of California, Riverside. The CS department is typically ranked in the top 60 by the U.S. News and World Report. The course usually serves about 350 students per quarter (three quarters per year, plus summer) with four class sections containing 80-100 students per section. In 2013, our university made one class section completely online. The online section is run identically to the physical class sections with the lectures and lab sessions carried out via synchronized online meetings that require real-time attendance. Four instructors rotate to teach the course, all with over five years of experience and strongly-positive student evaluations. In a given quarter, two instructors teach the course, each with their own class sections. Each section consists of three hours of instructor-led "lecture" per week. A typical lecture consists of short talks, coding examples, and small-group coding activities. All sections have two scheduled-lab hours per week led by a TA. Instructors and TAs hold weekly office hours. An online discussion board is used for questions / answers. This experiment was conducted during the Spring 2017 quarter. The CS1 course had about 250 students split into three sections. Most students were non-computing majors.

2.2.2 THE EXPERIMENTAL GROUP

The experimental group was one online section of our CS1 containing 76 students. The control group was the other two in-person sections containing 166 students. Most features were kept the same for all three sections except the weekly programming assignments, the midterm percentage, and allowing collaboration as described in 2.2.6. All sections took the same midterm and final exams.

2.2.3 COURSE TASKS AND GRADES

In all class sections, students were assigned three tasks each week. (1) Reading tasks that consisted of completing small activities and answering questions (multiple choice, true/false, short-answer) found in our online textbook. Readings were due before lecture. (2) Homework tasks were small auto-graded coding exercises, typically about 15-20 per week, usually typing a few lines of code in a template program, like writing an if-else statement or a for loop. (3) Programming assignments required students to apply that week's topics by writing one or more full programs.

The control and experimental groups had the same grade percentage points for reading tasks (7.5%), homework tasks (7.5%), in-class participation (5%), and the final exam (35%). Grade percentage points differed for programming assignments (control 25% vs. experimental 15%) and the midterm exam (control 20% vs. experimental 30%).

The experimental group was given 7 MSP lab activities per week versus the usual one large programming assignment per week (plus warm-ups) in the control group. Students could earn 0-10 points, per MSP lab activity, depending on how many test cases their program passed. Students in the experimental group were told that 50 points yielded

100% for the week. No extra credit was given for earning more than 50 points in a week. Both groups used the same program auto-grader with immediate score feedback. Neither group had limits on the number or rate of submissions.

2.2.4 PROGRAM AUTO-GRADER

We used a program auto-grader published by zyBooks [72]. zyBooks' auto-grader is a fully web-based system that makes creating and grading assignments simple. An instructor creates a new lab activity by clicking a button that opens a web form. The instructor enters a title and a text specification (with some formatting available). The instructor chooses some configuration options such as compiler flags, number of submits allowed (we selected unlimited), and whether submits are metered (we did not meter). A code template can be provided for students as well (we usually provided a basic template having includes and the main() function).

Next, the instructor creates test cases. An "input/output" test case involves typing input values paired with expected output values. For example, if a program should square its input, an "input/output" test case might have an input of -5 and an output of 25. The instructor can create any number of test cases and assign any point value to each test case. Test cases can also be configured to ignore output whitespace, indicate that the output need only start with the expected output (or end with it), and more. Another kind of test case is a unit test where a student's function/method can be called directly to check the returned result. Figure 2.1 shows a sample MSP lab activity generated via zyBooks' program auto-grader. The specification section details the MSP lab activity's prompt and title, the template code section is where sample code is provided and where the students

interact with zyBooks' IDE, and the assessment section is where all the test cases are located.

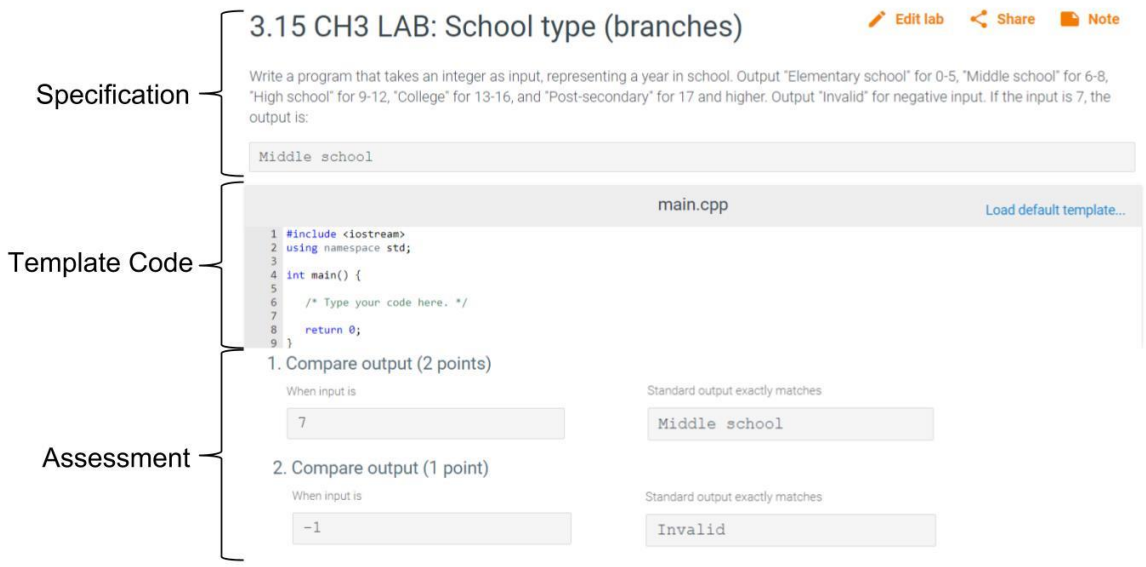

Figure 2.1: Sample MSP lab activity generated via zyBooks' program auto-grader.

2.2.5 Many Small Program (MSP) Assignments

The course covered input/output, variables/assignments, branches, loops, functions, and vectors. C++ was the language used in the course. All these topics were taught within nine weeks (the 10th week covered various topics not involving programming assignments). Solution sizes ranged from 10-50 lines of code. Each week's MSP assignment consisted of 2 easy, 3-4 medium, and 1-2 hard MSP lab activities.

2.2.6 Collaboration on Many Small Programs

With each MSP lab activity being lower stakes in the experimental group vs. the control group, the experimental group was told that they could collaborate versus the control group whose students were allowed to discuss programs conceptually, but not to show their programs to each other. The experimental group was told that similar or

identical submissions were allowed (though they should indicate collaborators or other helpers in comments). The only thing considered academic dishonesty would be having someone else write their program. They were told that half the midterm and final exams consisted of short coding problems of similar difficulty as the programming assignments, and were given sample midterm and final exams illustrating that fact.

2.3 CS1 Student Survey Results

Our main goal was to improve student experience; hoping to reduce attrition and attract students to computing majors. To gauge student satisfaction, we created a "stress survey" to learn more about the students' experience in CS1. This survey asked 18 questions based on a 6-point Likert scale, with responses ranging from Strongly agree (6) to Strongly disagree (1); no option of "Neither agree nor disagree" was given. To reduce bias, some questions were asked such that a more agreeable answer was favorable ("I enjoy the class") and others such that a less agreeable answer was favorable ("I am often anxious about the class"). Questions such that a higher response number is favorable are listed above the bolded line and questions such that a lower response number is favorable are listed below the bolded line. The questions are shown in Table 2.1. Note that the questions were asked to the students in an intermixed order. The survey was given in the 8th week of a 10-week quarter.

Table 2.1: Results of the "stress survey" for Spring 2017. p-values denoted by * are nearing significance ($p < 0.05$) and p-values denoted by ** are significant under the Bonferroni correction ($p < 0.0028$). Most favored the experimental group.

Question	Control group avg.	Experimental group avg.	p-value
I enjoy the class	4.53	4.87	0.046*
This class is an appropriate amount of work per week for the number of units	3.73	4.09	0.073
I was prepared for the midterm exam	3.63	4.18	0.004*
I feel prepared for the final exam	2.78	2.84	0.414
The weekly programming assignments were enjoyable	3.37	4.13	0.001**
The weekly programming assignments contributed to my success in the course	4.58	4.87	0.058
I learned a lot from the weekly programming assignments	4.58	4.94	0.029*
I frequently collaborated with others on the weekly programming assignments	2.74	2.66	0.397
I feel confident in my ability to write a small (< 50 line) useful program	3.98	4.32	0.087
I am often anxious about the class	3.72	3.15	0.020*
I spend a lot of time in the class figuring out system issues rather than learning programming	2.99	2.43	0.022*
The number of tools and websites for this class are somewhat overwhelming	3.15	2.50	0.010*
I have missed a deadline because I thought it was another time	2.48	2.75	0.202
I have looked for class info but couldn't find it	2.19	1.94	0.174
I felt anxious about the midterm exam	4.25	4.18	0.396
I feel anxious about the final exam	4.89	4.37	0.020*
The weekly programming assignments were stressful	4.31	3.93	0.058
The weekly programming assignments were frustrating	4.34	3.99	0.078

In addition, we conducted a combined analysis. Per question, we z-scored all responses. We concatenated the z-scores from the control group into one list, and separately, the experimental group into another list. We compared the lists with the student's T-test using two tails, yielding a p-value of 1.68E-69, which is smaller than the Bonferroni correction value of 0.0028 (0.05 / 18), and thus interpreted as significant.

We converted the average of each list, control group's z-score average was -0.0707 and experimental group's z-score was 0.1563, into a percentage difference via (1) calculating the absolute difference between the average of each list, yielding 0.2270; (2) converting the difference into a percentage using an online tool by Measuring U [40] (two-sided); (3) dividing the resulting percentage by 2 to get the difference from the 50th percentile. In conclusion, we found that the experimental group preferred the class 9% more (p-value = 1.68E-69) than the control group.

We did not expect the experimental group to spend more time on programming assignments per week since although more programs existed, they were smaller. We expect all CS1 students to spend about 2-3 hours per week on their weekly programs and our in-class surveys confirmed that both groups spent about that time. In Table 2.1, the experimental group students indicated they felt the class was an appropriate amount of work per week for the number of units (even more so than the control group, but not quite statistically significant).

We also sought to compare the students' experiences between the current experimental group and the previous time that same instructor taught the course. This was to determine whether the instructor alone might account for the differences. The

instructor provided their teaching evaluations for their previous offering (55 students completed the form) and for Spring 2017 (44 students completed the form). The "Assignments contributed to my learning" response went from 4.4 of 5 (department average was 4.2) to a high 4.72 (dept. avg. was 4.33) in the spring. In fact, nearly all scores went up, including the university's highest priority question "The instructor was effective," which improved from 4.3 (dept. avg. was 4.2) to 4.67 (dept. avg. was 4.25). In fact, the course evaluations were in the 85th percentile for the entire university (30,000 students), which is unusual for a required CS class for non-majors. This data further supports that the change to an MSP approach had a strong positive impact on the students' experiences.

2.4 CS1 Student Grade Performance

We sought to improve the student experience without worsening student performance. We thus compared the experimental and control groups' performance on exams and other course tasks. The exams were half multiple-choice questions and half short coding questions (both in terms of points and approximate time). Table 2.2 shows that the experimental group performed significantly better than the control group on the midterm and final coding questions and slightly better on the multiple choice questions. The experimental group and the control group performed similarly on reading activities and weekly programming assignments. The control group performed better than the experimental group on homework activities.

Table 2.2: Control vs. experimental group averages on exams and other course tasks for Spring 2017. p-values denoted with * are nearing significance ($p < 0.05$) and p-values denoted with ** are significant under the Bonferroni correction ($p < 0.0056$).

	Control group %	Experimental group %	p-value
Final	70.1%	75.7%	0.009*
Final multiple choice	72.9%	75.4%	0.097
Final coding	67.2%	75.9%	0.003**
Midterm	68.2%	79.9%	$p < 0.001$**
Midterm multiple choice	84.4%	86.5%	0.075
Midterm coding	53.6%	73.4%	$p < 0.001$**
Reading activities	97.1%	95.3%	0.153
Homework activities	94.2%	87.6%	0.002**
Weekly programming assignments	88.4%	87.1%	0.317

Although we had no reason to believe that online students would do better, one might question whether the section being online led to the higher scores. Thus, we analyzed the scores for the past three offerings of the CS1 course. Table 2.3 shows that the online section traditionally does not outperform the in-person sections, and in fact, typically performs worse on the exams.

Table 2.3: In-person versus online averages for Spring 2016, Fall 2016, and Winter 2017, for about 1,000 physical and 300 online students, showing the online section traditionally performs worse on exams compared with the in-person section. P-values denoted with a * are nearing significance ($p < 0.05$).

	Physical	Online	p-value
Final	81.1%	78.3%	0.390
Midterm	81.5%	78.3%	0.001*
Reading activities	93.1%	93.0%	0.964
Homework activities	94.3%	92.0%	0.030*
Weekly programming activities	89.1%	83.2%	$p < 0.001$

We also note that the instructor who taught the experimental group section had taught some of the online sections in the past (three times over the past three years).

These instructor's previous online sections never outperformed the other sections; instead usually performed slightly worse; consistent with Table 2.3. This data increases our confidence that the improvements seen in the online section of Spring 2017 is indeed due to the introduction of an MSP approach.

2.5 DISCUSSION

The experimental group had three related changes: MSP approach versus an OLP approach, allowing collaboration on those programs, and those programs being worth 15% rather than 25% (with more weight on the midterm). One might ask if one of the latter two factors caused the different survey and performance results. However, we had previously experimented with lowering the program percentage points and increasing exam percentage points, but such a change was highly unpopular. Students complained they spent too much time on programs worth little. We also experimented with collaboration before but saw drops in exam scores as students over-relied on their classmates for help.

Instead, we view the three changes as tightly interrelated. The MSP lab activities are less intimidating, meaning students are less likely to seek inappropriate help and more likely to attempt the programs themselves. The MSP lab activities are more focused, allowing students to see their skills improving (e.g., each loop gets easier to write) and to see how those skills will help on exams. Those factors enabled us to allow collaboration since students would not immediately cheat such a system, instead first making attempts because the programs are approachable and their usefulness clearer. (One might note that the experimental group's answer to the survey question "I frequently collaborated…" is

not higher than the control group). Likewise, those factors allowed us to reduce the program percentage, without students getting upset as in the past.

We note that, although the experimental group only needed 50 of 70 points or 71% program completion, the group obtained 87% completion, nearly identical to the control group's 88%. Students voluntarily completed more than necessary, further suggesting students found the MSP lab activities approachable and useful. Also, no student asked for an extension.

One might ask, given the rampant cheating common in CS1, is allowing collaboration really going in the right direction? We think so. We believe most students want to learn and will do so given what they understand to be a fair and valuable learning experience. The positive benefits of student collaboration are well known and we wish to encourage such collaboration in CS1.

Ultimately, we believe this is a case of technology enabling a new approach and perspective. The traditional OLP model may never have been best for students but was what instructors could manage. When programs were graded by hand (and when creating auto-graded assignments was time-consuming), instructors could not conceive of giving so many program assignments per week. Plus, with students not getting scores back immediately, the idea of "you only need 50 out of 70 points" was less feasible since students would not know their scores for a week or longer. When one looks at other skills, like playing an instrument, instructors do not set up one recital a week. Instead, students spend extensive time on drills, like playing scales, with instructors providing immediate feedback to correct mistakes.

If CS1 is taught only by an MSP approach, when will students learn to write larger programs? Our thoughts:

- Majors will learn to write larger programs in CS2.
- Non-majors, if they need to program in their careers, are more likely to have to write programs similar to the MSP lab activities, like writing a small add-on function for a statistical analysis tool, for google docs, for a database query, etc. If they need to write more substantial programs, they will probably take a CS2 class (or more).
- With the above said, we note that we intentionally ran the experiment in a more "extreme" manner, to see what effect would occur. Going forward, our instructors plan to give one large assignment mid-quarter and one large assignment end-of-quarter, with the other eight weeks using the MSP approach.

Furthermore, we found that the MSP-trained students did just as well as the OLP-trained students for the next term in CS2, which uses the traditional OLP approach and does not allow collaboration. This analysis is discussed more in Section 3.9 and explored in detail in Chapter 4.

We mention that the instructor who taught the experimental section stated that they "are never going back." The instructor has always had positive experiences teaching, and (like the other CS1 instructors) has above-average student evaluations and positive student comments. Still, the instructor said this was the best CS1 teaching experience they had in 20 years.

2.6 CONCLUSION

New technology, namely a program auto-grader with rapid/easy assignment creation, enables replacing the traditional CS1 one large program approach by a many small programs approach. The MSP approach involves numerous smaller, focused programs due each week accompanied by a scoring threshold (in our case, 50 of 70 points for the week yields full credit). The MSP approach is less intimidating for students. Students can build confidence and skill on the easier lab activities and then work on the harder ones; skipping around until they earn enough points. The MSP approach enabled us to allow collaboration. The MSP approach allowed us to decrease the program percentage and increase the exam percentage contribution to the total course grade, to be better assured in testing what students know due to the controlled testing environment (versus programming assignments where even with a no collaboration policy, instructors cannot be certain who is doing the programming). The MSP approach led to significantly greater satisfaction and less stress among the students. The approach also yielded improved performance on the exams. The approach led to improved experiences for the instructor and TA, including not having to spend any time on academic dishonesty. As a result, our department switched all sections to primarily use the MSP approach, with continued success. We encourage other departments to consider the approach.

Chapter 3. An Analysis of Using Many Small Programs in CS1

3.1 Introduction

Our initial analysis [4] on using an MSP teaching approach in CS1 concluded that such an approach could result in happier, less-stressed students without hurting student performance. In fact, we saw that an MSP approach led to improved code-writing scores on exams, likely due to students having more practice on focused concepts. After sharing the results of our initial analysis, all CS1 courses at our university switched from an OLP approach to an MSP approach the following fall 2018 quarter. Not only did our university switch CS1 teaching approaches, other universities began incorporating an MSP approach in their own CS1 classes as well -- many cloning our programming assignments (with permission), and dozens of others have asked to view our MSP lab activities and are considering adopting the approach. Since an MSP approach was becoming a large part of our CS1 curriculum, we wanted to understand more about how students were interacting with the new MSP lab activities. Chapter 3 seeks to answer common questions related to MSP assignments.

3.2 Methodology

3.2.1 Course

The study was conducted in our CS1 at the University of California, Riverside (UCR). UCR's CS department typically ranks in the top 60 by U.S. News and World Report. The university operates on the quarter system. Each academic year is divided into

three "regular" 10-week quarters (fall, winter, spring) and one compressed 5-week summer session. Throughout the academic year, the CS1 course serves around 300-500 students each quarter. The course is required for all computing majors and for various engineering, science, and math majors, such that about half the students are computing majors and half are non-computing majors. The course topics include basic input/output, assignments, branches, loops, functions, and vectors. The weekly structure of the course includes three hours of instructor-led lecture, two hours of TA-led labs, interactive online readings, and auto-graded homework assignments. The course teaches C++ as the programming language. The course has a midterm during week six and a final after week 10. Each exam's points come half from multiple choice questions and half from free-response coding questions. The course uses active learning and peer learning in lectures.

3.2.2 DATA COLLECTION

We analyzed data from a Spring 2017 76-student section of our CS1 course that used an MSP approach. Our CS1 used an online textbook published by zyBooks for all class readings, activities, and lab activities. At the quarter's end, we collected all student submits and develops for lab activities from zyBooks and combined them into one spreadsheet. A submit is defined as when the student "turns in" their assignment for grading. A develop is defined as when a student runs their code through the zyBooks compiler for testing without grading (development was done in the built-in zyBooks coding windows; students were not introduced to an external development environment). Each student submit has metadata about the lab activity title, a userID (anonymized and generated from zyBooks), the submit score, the max score possible for the submit, and a

timestamp. A develop has the same metadata as a submit but without a score and a max score. For this study, we collected data from the 76 students for 61 MSP lab activities. In total, we collected 16,106 submits and 48,186 develops a total of 64,292.

3.3 How Much Time Do Students Spend Working on MSP Assignments?

We generally expect students to spend between 2-3 hours per week working on their programming assignments. Our past surveys and analyses showed students on average spending about 2 hours, the average pulled down by students who submit few or no programs (of course some students spend more than 3 hours as well). We designed the MSP assignments to take about the same total time per week as the traditional OLP approach. We seek to answer the question: how much time do students spend working on MSP assignments?

To calculate the total time students spent on MSP assignments, we used each timestamp for a develop or submit, calculated the difference between each timestamp, and summed the differences. We excluded a difference that exceeded 10 minutes, assuming the student took a break. Note that our calculations are thus an underestimate, as some breaks may have actually involved the student working or researching, and we also cannot capture time spent understanding and working on the MSP assignments before the first develop or submit.

Figure 3.1 summarizes the average time spent by students on MSP assignments per week, as calculated above. The x-axis is the week number and the y-axis is the time spent in minutes. On average, students spent 17 minutes per MSP lab activity and 120

minutes per MSP assignments each week. This excludes week 1 (which had easy introductory programs) and week 9 (which had fewer lab activities to complete). The two most challenging weeks were week 4 covering loops, and week 8 covering vectors. The dips in weeks 6 and 7 are due to several MSP lab activities having students rewrite earlier MSP lab activities, but using user-defined functions.

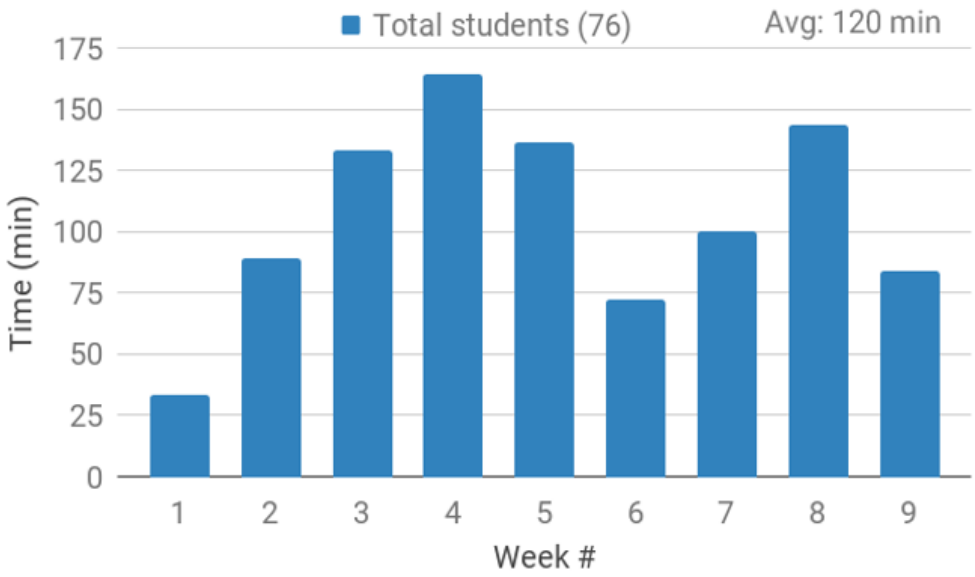

Figure 3.1: Average time spent by students each week on MSP assignments. Students with 0 submits or 0 time spent were excluded from calculations.

We compared our analyses with a survey during lecture of week 8 that had 21 questions, one of which being "The average hours per week spent on all zyLab programming assignments that week was?" with response options 1-2, 2-3, 3-4, …, 10+. Figure 3.2 summarizes student responses. 67 students responded. A weighted average yields about 5 hours per week, which is higher than our calculated time of 2 hours a week. This higher value may be due to various factors including: our calculations being an underestimate as mentioned earlier, students may overestimate or overreport time

spent, the survey's options may bias students towards selecting higher values, and the weighted sum may unintentionally round up.

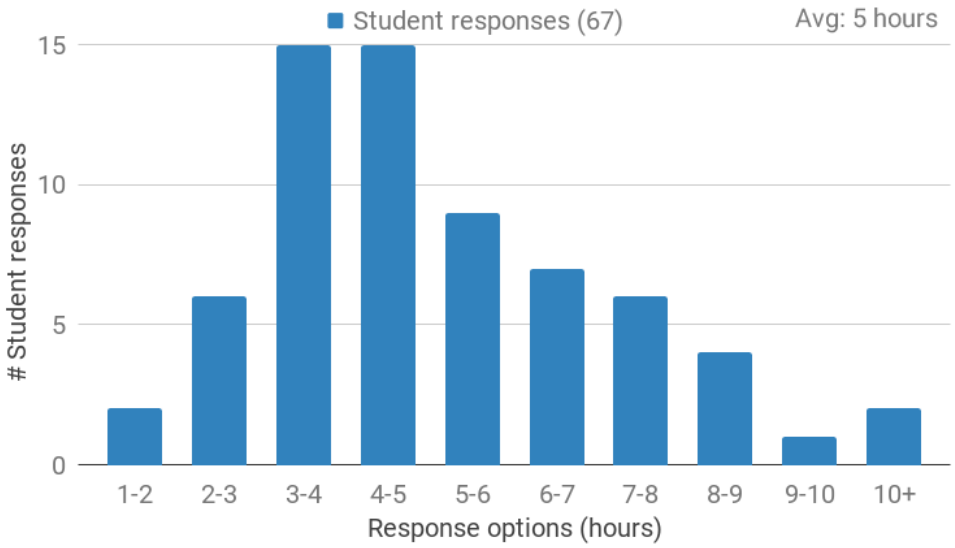

Figure 3.2: CS1 Spring 2017 survey responses (67 students) for "The average hours per week spent on all zyLab programming assignments that week was?" A weighted sum yields an average of 5 hours per week.

Figure 3.3 shows the time spent per MSP lab activity, using a box-and-whisker plot. The x-axis is the MSP lab activity (61 total) and the y-axis is the time spent in minutes. Dashed lines separate MSP assignments by week. The y-axis is capped at one hour (60 minutes). Students who did not attempt the given MSP lab activity are excluded from the calculations.

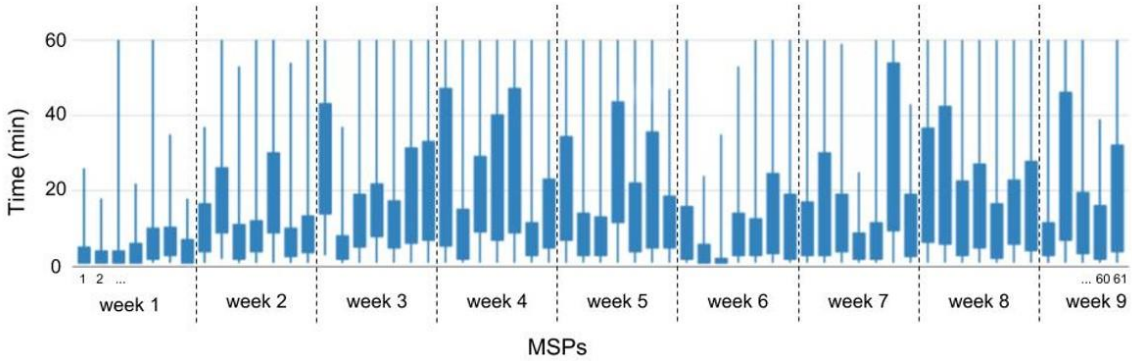

Figure 3.3: Box-and-whisker plot of student time spent for each MSP lab activity. On average, students spent 17 minutes per MSP lab activity excluding weeks 1 and 9.

3.4 How Many Days Before the Due Date Do Students Start MSP Assignments?

We released each week's MSP assignment on Tuesday, due the following Tuesday at 9:00 pm. That week's readings and lectures (Tuesday and Thursday, 80 minutes each) taught the concepts covered by that week's MSP lab activities. That week's 2-hour lab (Thursday) also taught those concepts, with about 30 minutes at the end for students to work on the MSP lab activities and ask questions. We seek to answer the question: how many days before the due date do students start working on MSP assignments?

Figure 3.4 summarizes the average number of days students began working on MSP assignments before the due date. The average was computed by finding students' first submit for a lab activity belonging to the MSP assignment in the given week, computing the days between the first submit and the assignment's due date, calculating the percent of students that started T-7, T-6, …, T-0 days before the due date, and then averaging across all MSP lab activities. The x-axis is the number of days prior to the due date. Using "NASA countdown-like" terminology, we use "T-2" to mean two days before the due date (or Sunday). The y-axis is the average percent of students that fall under each category. Week 1 is excluded from these calculations since week 1 MSP lab activities were very easy.

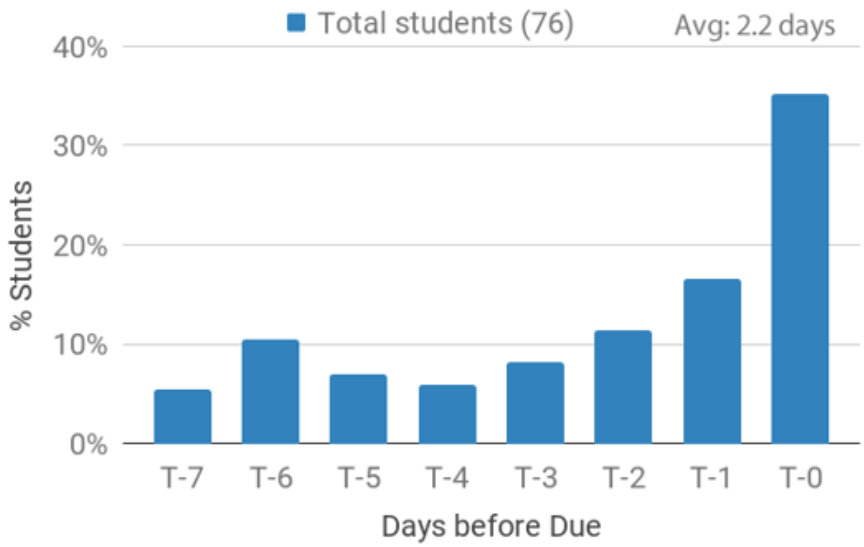

Figure 3.4: Percent of students who began MSP lab activities each week T-X days prior to the due date - Spring 2017.

To our pleasure, 37% of students (28) started 3 days ahead or more; however, the other 63% of students started only 2 days ahead or less, with 35% of students (27) starting on the due date. Students on average began 2.2 days ahead of the due date.

Figure 3.5 shows start times for the other two CS1 sections that quarter, which used OLP assignments. Those students began on average 2.1 days ahead of the due date. Only 28% (48) started 3 days ahead or more, and 25% (43) started on the due date. Note that the due dates were different between the sections, but this comparison still gives valuable insight.

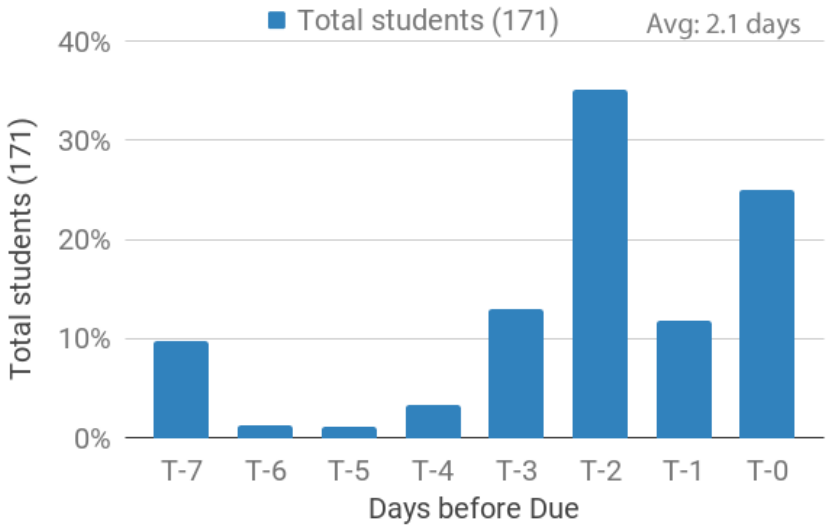

Figure 3.5: Percent of students who began OLP assignments each week T-X days prior to the due date - Spring 2017.

We had hoped that MSP assignments' less-intimidating nature would have led to earlier starts by most students. MSP assignments had a mild impact on students starting earlier, but many students still started on or near the due date. We believe starting earlier is good practice, and thus decided to try to encourage earlier starts. The MSP approach made such encouragement easy. In our Fall 2018 course, we simply included the following policy in our syllabus: "To discourage procrastination, you will be required to complete at least 20 points out of the 50 points each week by Sunday at 10 pm", which is 2 days prior to the Tuesday, 10 pm deadline. That small change led to substantial modification in student behavior, with start dates shifting from 2.5 (weeks 2 – 5) to 5.3 days before the due date.

3.5 What Percent of MSP Lab Activities Do Students Complete Each Day?

Section 3.4 showed when students started, defined as achieving at least 1 point on the MSP lab activity (out of 10 points). In section 3.5 we analyze total completion percent per day. We seek to answer the question: what percentage of MSP lab activities do students complete each day?

Figure 3.6 summarizes the completion rate of MSP lab activities per day. The x-axis is the number of days prior to the due date and the y-axis is the completion percentage. The top bar is the percent completed on that day and the bottom bar is the cumulative completion prior to that day. Recall that only 50 of 70 points (71%) were required for full credit.

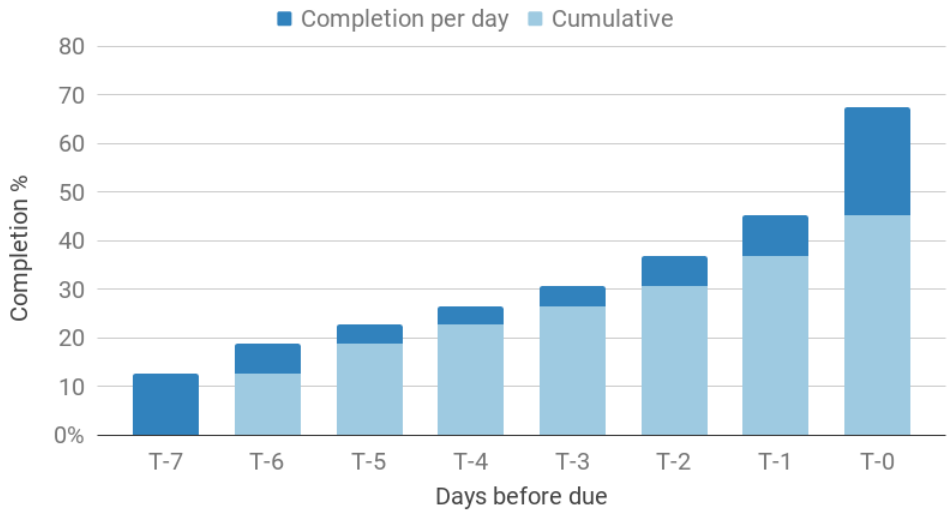

Figure 3.6: MSP lab activity completion T-X days prior to the due date. The top bar is the percent completed on that day, and the bottom bar is the percent completed prior to that day.

Figure 3.6 shows a gradual increase in the completion rate throughout the week. The completion rate increases 5-10% each day except for the last day (T-0) which has

about a 20% increase. Because students need only complete 50 of 70 points, some MSP lab activities have 0% completion, pulling down the averages shown.

3.6 Will Students Complete More MSP Lab Activities Than Required?

Each week, students were assigned 7 MSP lab activities (10 points each) and were only required to complete 50 points of 70 to score 100% on the MSP assignments for the week. No extra credit was given for exceeding 50 points. We refer to the 50-point cutoff as the full-credit threshold. We seek to answer the question: do students willingly complete more MSP lab activities than required?

Figure 3.7 shows the percent of students that scored equal to or above the full-credit threshold each week. The bottom bar is the students that completed above the threshold and the top bar is the students that completed equal to the threshold. In weeks 1, 2, 3, and 6, a higher percentage of students scored above the threshold than equal to the threshold. Across the quarter, an average of 40% of students scored above the threshold.

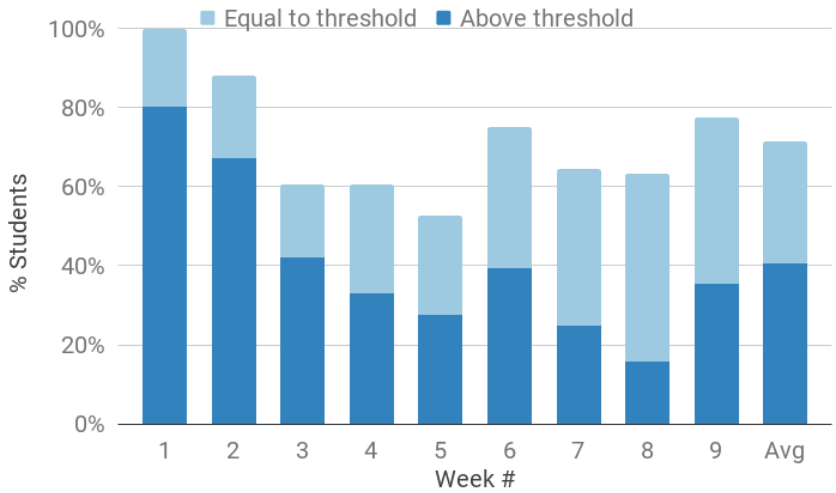

Figure 3.7: Percent of students who completed equal to or above the full-credit threshold each week.

Figure 3.8 provides a more detailed analysis via a bubble chart. The x-axis is the week number and the y-axis is the total points scored per week. The bubble size represents the number of students that scored that number of points. For example, the largest bubble in week 1 is labeled 53 because 53 students scored 70 points on MSP lab activities for that week. Note that students who scored 0 points for the week are not included because those students likely dropped the class or decided not to submit labs for the week. The dashed line represents the full-credit threshold for each week. Note that week 9's threshold is lower since only five MSP lab activities were given to students. On average, students who scored more than the full-credit threshold scored an additional 13 points. As each MSP lab activity is worth 10 points, this translates to completing an additional 1.3 MSP lab activities each week.

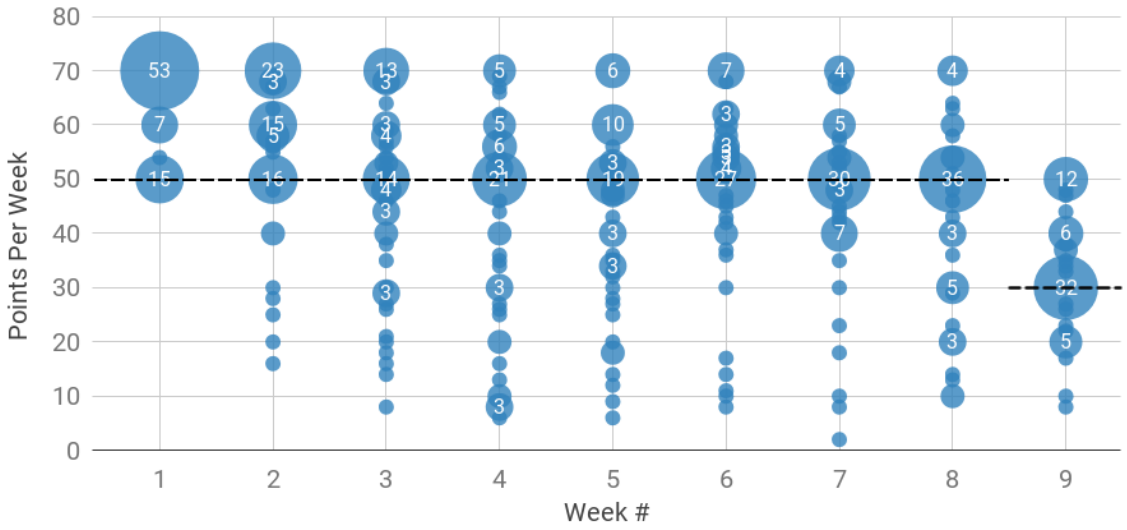

Figure 3.8: Points students scored each week. Students who scored 0 points for the week are excluded. Dashed line indicates max points for the week.

We were pleased to find that so many students were able to meet the full-credit threshold and that a substantial number were willing to do more than the minimum

required work. The results from Figure 3.7 and Figure 3.8 suggest that students find MSP lab activities helpful or enjoyable since they complete more than required even without an extra credit opportunity.

3.7 Do Students Take Advantage of Switching Among MSP Lab Activities When Stuck (pivot)?

Pivoting is when a student partially completes an MSP lab activity (e.g., scores 6 of 10 points) and then decides to work on a different MSP lab activity. Typically, with traditional OLP programming assignments, students only have the option to work on the program until completion. If stuck, a student has few or no options. With the MSP approach, students can pivot to another lab activity while working to score additional points or even learn from another lab activity to help themselves with the current lab activity they are struggling on. We seek to answer the question: do students take advantage of the opportunity to pivot, and if so, how often?

A student run (either a submit or develop) is defined as a pivot if all following rules are met:

- The current run is not the student's first run for the week
- The current run is for a different MSP lab activity than the previous run
- The current run is for an MSP lab activity that has not been completed
- The previous run has not been completed
- The current run and previous run are for the same MSP assignment

Figure 3.9 shows the percent of students who pivoted at least once in a given week. The x-axis is the week number and the y-axis is the percent of students that pivoted that week.

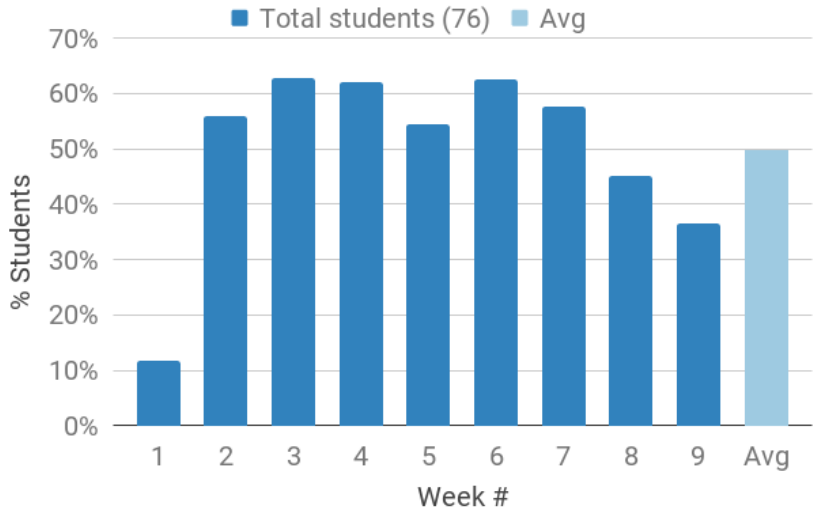

Figure 3.9: Percentage of students who pivoted at least once in a given week. An average of 50% of students pivoted at least once each week.

We found that students pivot on average 1.3 times each week. The highest number of pivots was one student who pivoted 12 times in week 4. Week 1 had few pivots due to the MSP lab activities being easy. With more challenging programs beginning in week 2, students made much use of pivots. Students who pivoted at least once a week pivoted on average 2.5 times.

For insight, we highlight three actual pivoting scenarios.

3.7.1 Pivot at 0% - Week 8 (vectors)

A student attempted MSP lab activity 5 (LA5) three times but received 0 points on all submits. Instead of continuing LA5, the student switched to LA7 and scored 10

points. The student did not return to complete LA5. The student scored 50 points on the MSP assignment for the week, meeting the 50-point full-credit threshold.

3.7.2 Single Pivot - Week 3 (branches)

A student worked on MSP lab activity 4 (LA4) and scored 8 points. The student switched to LA6 and scored 10 points. The student did not return to complete LA4. The student scored 48 points on the MSP assignment for the week, nearly meeting the 50-point full-credit threshold.

3.7.3 Multiple Pivots (3 or more) - Week 4 (loops)

A student worked on MSP lab activity 4 (LA4) and scored 2 points. The student switched to LA5 and scored 10 points. The student returned to LA4 and improved their score from 2 points to 8. The student moved to LA7 and scored 9 points. The student then worked on LA6 and scored 10 points. Finally, the student returned to LA4 and improved their score from 8 points to 10. The student scored 69 points on the MSP assignment for the week, exceeding the 50-point full-credit threshold and nearly hitting the 70-point max.

Students seem to take advantage of the pivot benefit that an MSP approach offers, especially when a threshold is used. 94% of students (71 students) pivoted at least once throughout the 10-week quarter. As a result, we hope to do future work to investigate whether students who pivot score higher than those who do not, whether there are any detriments to pivoting, and whether students who pivot return and solve the MSP lab activities they switched away from. A more in-depth analysis of pivoting can be found in Chapter 8.

3.8 Do Students Use MSP Lab Activities to Study For Exams?

Given that MSP lab activities are short, concise, and focus on a single concept, we seek to answer the question: do students voluntarily redo MSP lab activities to prepare for exams?

Given the dates for the midterm and final exams, we defined criteria to determine if a student used an MSP lab activity for exam practice. We said that a student used an MSP lab activity for exam practice if the student had, for that MSP lab activity, a submit or develop timestamp that was after the MSP assignment's due date and within one week prior to the exam. The midterm occurred during week six of the quarter and the final occurred at the end of the quarter.

Table 3.1 shows the results of how many students used MSP lab activities for practice and how many unique MSP lab activities were used to study. 54% of students (41) used MSP lab activities to study for either the midterm or final. 98% of all MSP lab activities (60) were used by at least one student to study for an exam.

Table 3.1: Student use of MSP lab activities for exam preparation.

Total number of students	76
Total number of MSP lab activities	61
% of students that used MSP lab activities to study for the midterm	38%
% of students that used MSP lab activities to study for the final	37%
% of students that used MSP lab activities to study for either exam	54%
% of MSP lab activities that were used to study for the midterm	97%
% of MSP lab activities that were used to study for the final	90%
% of MSP lab activities that were used to study for either exam	98%

We are pleased to see many students using MSP lab activities to study for exams. For comparison, we looked at the other two sections of CS1 from Spring 2017, which used an OLP approach. Only 10% of students (17) used OLP assignments to study for exams.

3.9 Do MSP-Trained Students Do Poorly in a CS2 Using an OLP Approach?

A common concern regarding using an MSP approach in CS1 is the impact MSP assignments will have on students when they reach CS2 using an OLP approach. We seek to answer the question: how do students taught programming in CS1 via an MSP approach (MSP-trained) fare in CS2, compared to students taught programming in CS1 via an OLP approach (OLP-trained)?

We gathered data from our CS2 course from Winter 2017 through Spring 2018 (5 quarters). We determined which students took CS1 using an MSP approach and which took CS1 using an OLP approach. To be conservative, we excluded students who did not take CS1 at our university. We found 241 students that took CS1 via an MSP approach and 312 students that took CS1 using an OLP approach. In total, 553 students who took CS2 at our university were considered in our analysis.

Figure 3.10 shows CS2 performance results. The x-axis shows the class work categories we analyzed (participation activities, labs, programming assignments, midterm exams, final exam, and total grade in the class) and the y-axis is student grade performance. OLP-trained students are the light bars on the left and MSP-trained students are the dark bars on the right.

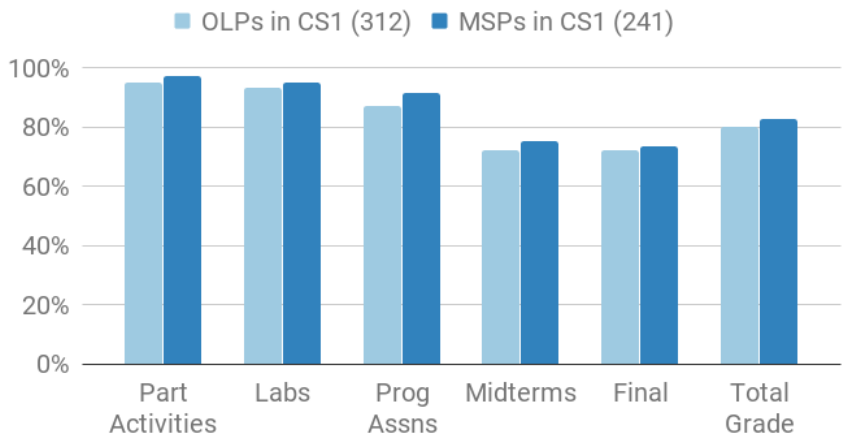

Figure 3.10: CS2 performance for MSP-trained students versus OLP-trained students. MSP-trained students do no worse, and in fact do slightly better.

Figure 3.10 shows that MSP-trained students perform similarly, and in fact slightly better, than OLP-trained students. Note that the purpose of this analysis is not to claim an MSP approach in CS1 leads to better performance in CS2. Instead, the analysis shows that an MSP approach does not harm students in CS2. We hope to do further research to better understand the effects that using an MSP approach in CS1 has on students in CS2.

3.10 Conclusion

Modern easy-to-use auto-graders enable new teaching approaches in CS1 courses, like using an MSP approach instead of an OLP approach for weekly programming assignments. Our previous research showed that using an MSP approach in CS1 yielded happier students and better grades in the course. This paper analyzed how students use MSP assignments. We conclude that students are making good use of MSP assignments to aid in their learning process: Students spend sufficient time working on MSP assignments each week, begin working on MSP lab activities earlier than for OLP

assignments, complete more MSP lab activities than necessary with a full-credit threshold, take advantage of pivoting between MSP lab activities, and use MSP lab activities to study for exams. We also see that MSP-trained students do just as well, even slightly better, than OLP-trained students in a CS2 that uses an OLP approach. Our department now uses an MSP approach in all CS1 sections, and we are aware of dozens of other schools that have switched to an MSP approach as well.

Chapter 4. Does a Many Small Programs Approach in CS1 Hurt Student Performance in CS2?

4.1 Introduction

When hearing about using an MSP approach in CS1, a common criticism / concern among instructors is that CS1 students who learn programming using an MSP approach may do poorly in a CS2 course that requires students to complete larger programs. Thus, Chapter 4 addresses the question, "Does an MSP approach in CS1 hurt student performance in a CS2 requiring larger programs?"

Chapter 4 summarizes a study that considered five quarters of one university's CS2 course, comparing performance of 417 CS1 students taught via an OLP approach (OLP-trained) and 241 CS1 students taught via an MSP approach (MSP-trained) who then took the CS2 course. The results show that MSP-trained students do not perform worse than OLP-trained students in CS2, and in fact perform slightly better. These results hold for programming assignments, midterms, finals, and more. The results also hold when controlling for gaps between quarters, and hold equally for males and females. The study suggests that instructors can embrace an MSP approach in CS1 to obtain happier, less-stressed students, who may perform better in CS1 too, without fear of putting those students at a disadvantage in CS2 (and who may gain a slight advantage).

4.2 METHODOLOGY

4.2.1 COURSE

This study was conducted at the University of California, Riverside, whose CS department typically ranks in the top 60 by U.S. News and World Report. The CS1 and CS2 courses are offered quarterly (fall, winter, spring) in 10-week "regular" quarters, plus during compressed 5-week summer sessions as well. The CS1 course serves about 300-500 students per regular quarter, taken about equally by computing majors and by non-majors who are mostly engineering, science, and math. The CS1 course uses C++ and topics include basic input/output, assignments, branches, loops, functions, and vectors. The CS2 course serves about 100-200 students per quarter, mostly computing majors and some engineering, science, and math majors. The CS2 course also uses C++ and topics include object-oriented programming, recursion, pointers, linked lists, abstract data types, software development principles, and development of larger programs. Both the CS1 and CS2 courses have three hours of lecture and two hours of TA-led labs per week, and require weekly online interactive readings and auto-graded homeworks. Both have midterm and final exams, each consisting of multiple choice and coding problems. Both use active and peer learning techniques in lecture.

4.2.2 DATA COLLECTION AND ANALYSIS

We collected gradebooks with all students who took CS2 during the past five regular quarters and did not drop/withdraw, totaling 658 students. We excluded the 36 students who did not take the final exam. For each included student, we determined whether the student was trained from an OLP or MSP offering of our CS1, via rosters

obtained from prior CS1 offerings. Figure 4.1 shows the CS1 and CS2 offerings that contributed to the study. Note that in Spring 2017, CS1 had two OLP-trained sections and one MSP-trained section. If we could not find the student in our CS1 rosters, that likely means they took CS1 at another school (we have many transfer students), or took our CS1 long ago. For those students, we assume they had an OLP training, which is almost surely the case and consistent with our knowledge of CS1 classes taught by the community colleges that mostly feed into our program. However, we also performed the analysis excluding such students (105 students excluded) and found results to be the same.

When our instructors switched to an MSP approach in Fall 2017, most decided to include two larger programs, one in week 5 and one in week 10, with the other 8 weeks having five MSP lab activities each. We have found no difference in CS2 student performance for "pure" MSP-trained students and "predominantly" MSP-trained students. Thus, we still use "MSP" to refer to an approach where nearly all weeks use MSP assignments, but where 1-2 weeks may use OLP assignments.

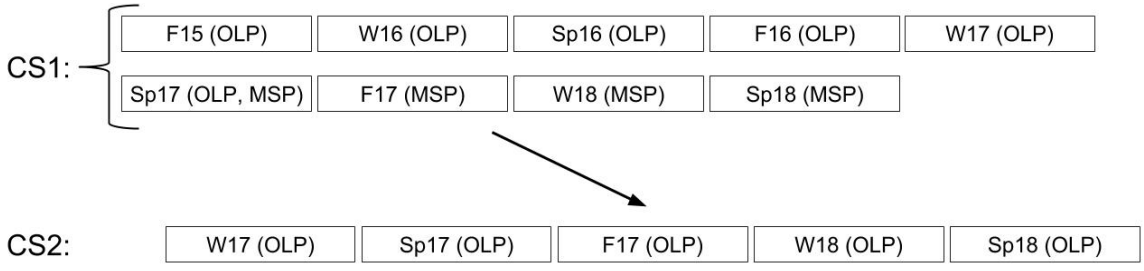

Figure 4.1: CS1 and CS2 class offerings considered.

4.3 Main Results

4.3.1 CS2 Programming Assignments

Figure 4.2 summarizes the main performance of CS2 students on the class' seven weekly programming assignments (some assignments spanned multiple weeks). The left, dark-blue bars represent OLP-trained students (417 students), and the right, light-blue bars represent MSP-trained students (241 students).

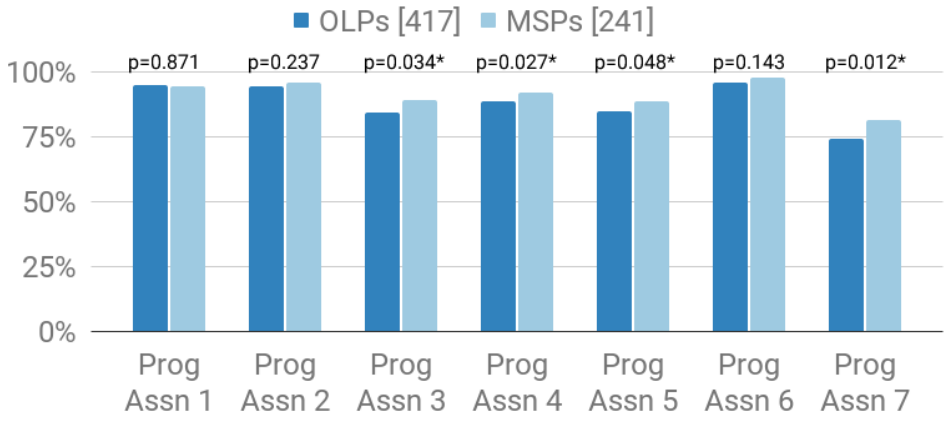

Figure 4.2: CS2 student performance on all seven large CS2 programming assignments. MSP-trained students did not perform worse than OLP-trained students (and in fact did slightly better). p-values are shown above each column. p-values denoted with * are nearing significance ($p < 0.05$).

The data shown in Figure 4.2 shows that using an MSP approach in CS1 does not hurt students in a CS2 requiring larger programs. In fact, MSP-trained students did slightly better. As the research question was not whether MSP-trained students do better in CS2, we do not focus on statistical significance; however, we do report p-values in Figure 4.2. More importantly, as the commonly-voiced concern by instructors is that MSP-trained students may do worse, the data shown in Figure 4.2 proves this concern to be false.

4.3.2 CS2 Midterm and Final Exams, and More

In addition to programming assignment performance, one may wonder how students fared on other aspects of the course. Figure 4.3 shows performance on the midterm and final exams, which consisted of multiple choice and coding questions. Again, MSP-trained students did not do worse (and instead did slightly better).

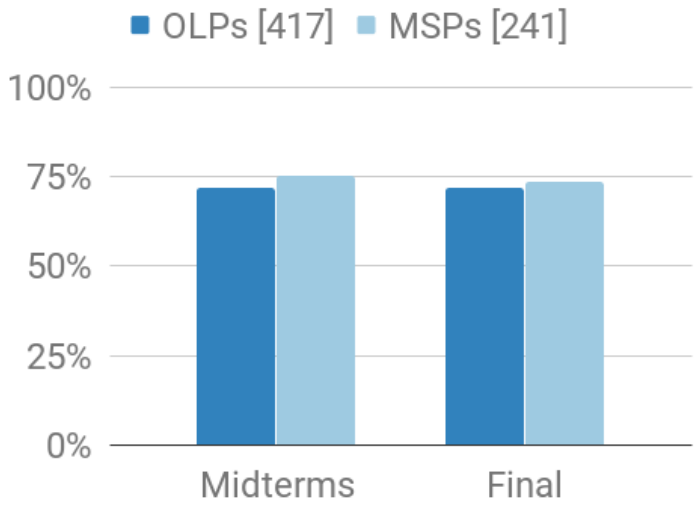

Figure 4.3: CS2 student performance on midterm and final exams. MSP-trained students did not perform worse (and in fact performed slightly better).

Likewise, Figure 4.4 shows student performance in other aspects of the course, including completing online participation activities before lectures and small in-lab "warm up" programming activities. For convenience, Figure 4.4 shows the programming assignment and midterm/final data from above and ends with the students' total grade in the CS2 course. The data shows MSP-trained students do no worse in any category.

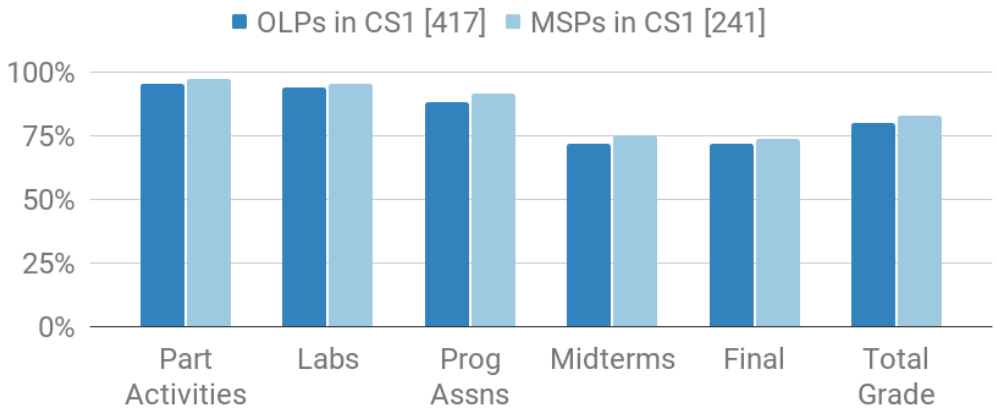

Figure 4.4: CS2 student performance on all aspects of the CS2 course. MSP-trained students do no worse in any aspect.

Earlier, we mentioned that if we could not find a CS2 student in our CS1 rosters, we assumed an OLP training. To be safe, we re-analyzed the data excluding such students. Figure 4.5 shows that the results are nearly identical to when those students are included.

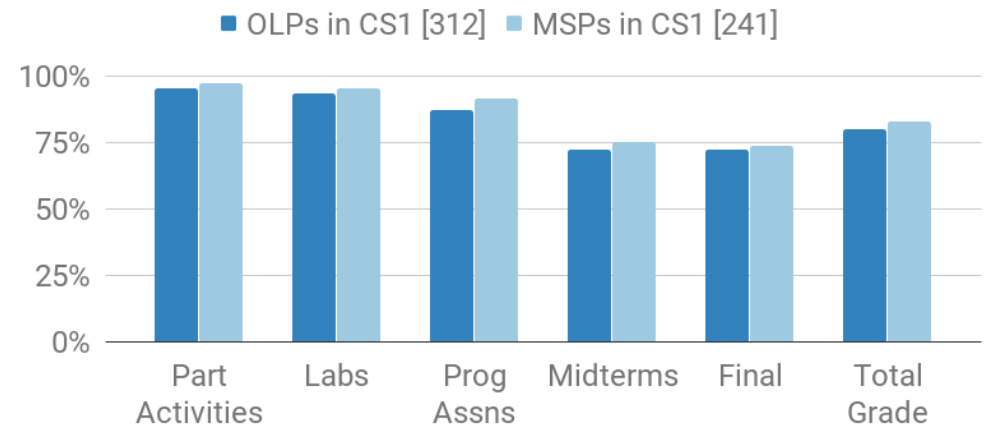

Figure 4.5: CS2 student performance considering only students known to have taken CS1 at our university. Results are the same: MSP-trained students perform no worse in any aspect.

4.4 Considering Gaps Between CS1 and CS2 Terms

One threat to the study's validity is the following: Because the MSP approach CS1 offerings were all more recent than the OLP approach CS1 offerings (see Figure 4.1), then the OLP-trained students' performance in CS2 may have been degraded due to gaps between the quarters in which they took CS1 and CS2.

Thus, we reanalyzed the data by considering only students who had no gap -- they took CS1, and then took CS2 in the next regular quarter, such as taking CS1 in Fall 2017 and then CS2 in Winter 2018. Figure 4.6 provides results, showing that MSP-trained students did no worse in any categories (and in fact did slightly better).

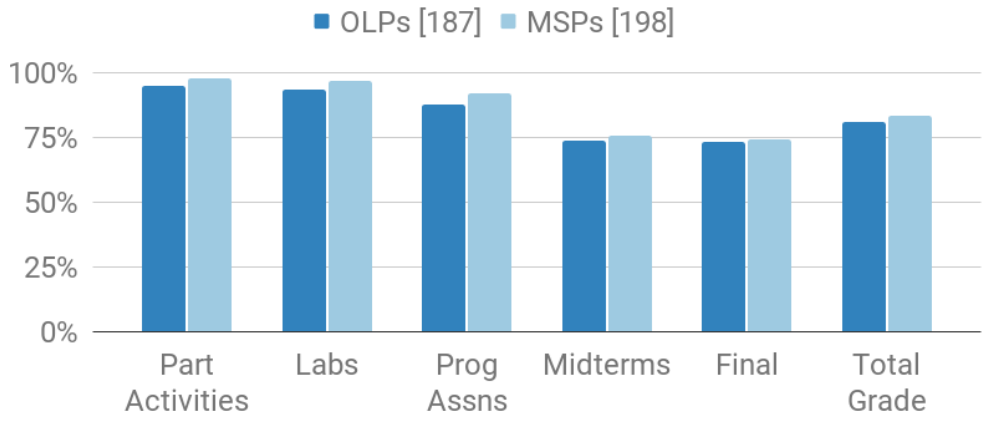

Figure 4.6: CS2 performance for students having no gap between taking CS1 and CS2.

We further analyzed the data for students having a one-quarter gap, and also for a two-quarter-or-more gap. The results, in Figure 4.7 and Figure 4.8, show MSP-trained students generally did not do worse, and again improved in some class areas. As the point is that MSP-trained students do not do worse, we do not analyze such improvements further here, but the results suggest future work as to why MSP-trained students did better -- perhaps the MSP approach's repetition lengthens retention of programming skill.

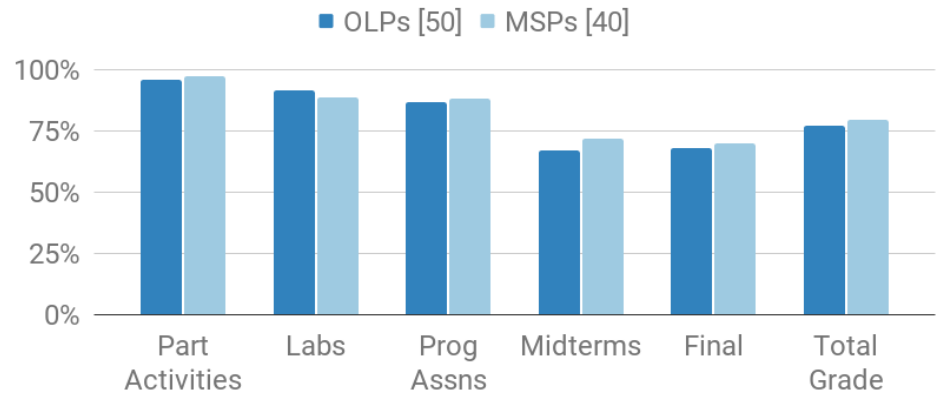

Figure 4.7: CS2 performance for students having a one-quarter gap between CS1 and CS2.

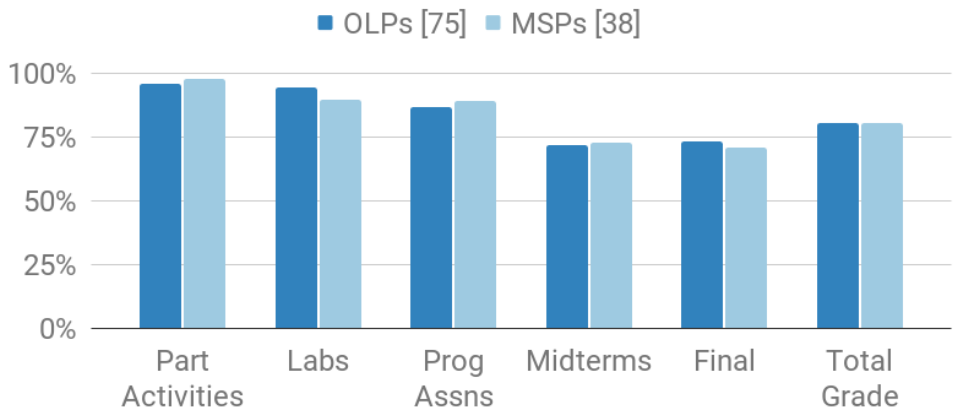

Figure 4.8: CS2 performance for students having a two-quarter or more gap between CS1 and CS2.

4.5 Considering Gender

Much recent emphasis has been on possible differential impact of classroom approaches on females. Thus, we re-analyzed the data considering gender. We did not have gender data in the gradebooks, and thus had to approximate gender based on names that had a high probability of being male or female. We excluded names that were not obviously male or female names. We understand this analysis is not perfect (especially considering gender identity), but believe it is better than no analysis. Figure 4.9 shows our findings.

The analysis shows that females with OLP and MSP CS1 training displayed the same general performance in CS2 as seen for the entire population of students, with MSP-trained students again not doing worse (instead doing slightly better) in CS2.

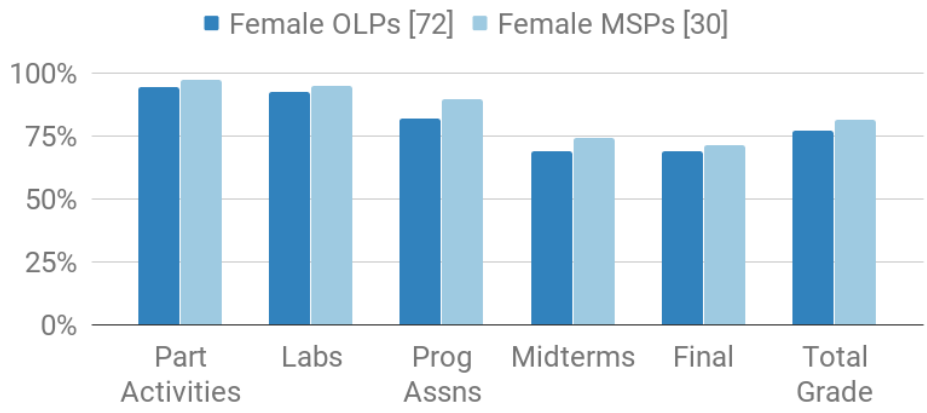

Figure 4.9: CS2 student grade performance considering gender. Data shows that MSP-trained CS1 females display similar performance in CS2, in fact performing slightly better.

We conducted a similar analysis for males, and for the non-obvious names category too. The results match the above analyses. Thus, the effect of MSP versus OLP CS1 training on CS2 performance seems to not have a strong gender component.

4.6 Conclusion

A many small programs (MSP) approach in CS1, largely enabled by modern program auto-graders, was previously shown to lead to happier, less-stressed students, who also performed better on the code-writing portion of exams, compared to the common one large program (OLP) per week approach. However, numerous instructors expressed concerns that MSP-trained students would do poorly in a CS2 requiring larger programs. Our study of 5 quarters of CS2 performance, for 471 OLP-trained students and 241 MSP-trained students, showed the MSP-trained students do no worse in CS2 (and

actually do slightly better), in CS2's programming assignments, exams, and all other course aspects. The results hold even when considering gaps between CS1 and CS2 quarters and hold for either females or males. We conclude that an MSP approach can be used in CS1 without fear of harming the students in CS2, and possibly helping them do better in CS2. Future research may include quantifying those possible benefits in CS2, and introducing an MSP approach in the beginning of CS2 for a smoother and stronger start.

Chapter 5. MANY SMALL PROGRAMS IN CS1: USAGE ANALYSIS FROM MULTIPLE UNIVERSITIES

5.1 INTRODUCTION

In 2017, we introduced a many small programs (MSP) teaching approach at our university. Instead of teaching via a one large programming assignment (OLP) each week, the MSP approach allows for the instructor to assign multiple programming assignments, for example 5 or more, each week instead. Our previous studies [4] have shown that an MSP approach can improve the student experience by reducing stress and increasing student satisfaction in the course. Furthermore, an MSP approach has been shown to improve student grade performance in CS1, especially on the coding portion of exams. In a follow-up study [5], we learned that students use MSP assignments in ways beneficial to their learning: students spend sufficient time working on MSP assignments each week, start working on MSP lab activities earlier, and more. We shared these findings with universities around the nation; causing other universities to switch from teaching CS1 using an OLP approach to an MSP approach. Given MSP lab activity data from other universities we extend our analysis to include MSP lab activities from other universities. We perform similar usage analyses and found that MSP-trained students from other universities also benefit from an MSP approach.

5.2 CS1 UNIVERSITY METADATA

We looked at CS1 courses taught using an MSP approach from 10 universities. To maintain anonymity of the universities included in this study, we do not include the name

of the institutions. The universities varied in size with some having classes of more than 300 students, while others had a class size of 20 students. Many universities taught CS1 using C++, while others taught via Python and Java. Table 5.1 provides details about the universities included in this study.

Table 5.1: Metadata on the 10 universities included in this study. Details include the programming language being taught, number of students in the class, number of MSP lab activities given, number of submits collected, and number of develops collected.

	Prog Language	#Students	# MSP lab activities	# Submits collected	# Develops collected
University 1	C++	20	98	3,177	5,635
University 2	Python	81	69	192,44	19,707
University 3	C++	30	19	2,397	3,416
University 4	C++	14	61	1,675	5,104
University 5	Java	11	51	643	3,535
University 6	C++	234	77	21,451	40,573
University 7	Python	333	43	88,981	103,089
University 8	C++	79	25	7,315	9,298
University 9	Java	56	59	7,454	18,505
University 10	Java	321	65	40,320	96,721

5.3 DATA COLLECTION

We analyzed data from 10 different CS1 classes taught at different universities. Each university used an online textbook published by zyBooks for programming assignments. After the course was completed, we collected all student run activity (submits and develops) for each lab activity from zyBooks and consolidated them into a single spreadsheet. A submit is defined as when the student "turns in" their assignment for grading. A develop is defined as when a student runs their code through the zyBooks compiler for testing without grading. Student submits have metadata that describes the

lab title, a userID (anonymized and generated from zyBooks), the submit score, the max score possible for the submit, and a timestamp with the date and time. A develop has the same metadata as a submit but without a score and a max score. In this study, 1,179 students were considered with 567 MSP lab activities included. In total, we collected 192,657 submits and 302,406 develops.

5.4 How Much Time Do Students Spend Working on MSP Lab Activities?

At the University of California, Riverside, we expect students to spend between 2-3 hours each week working on MSP assignments. When using the MSP approach, we assign students 7 MSP lab activities each week, requiring them to only earn 70% of the points for a 100% score on programming assignments each week. With this setup, we found that students generally spend 17 minutes on each MSP lab activity and thus spend at least 85 minutes a week (~1.5 hours) working on MSP assignments. Additionally, we found that many students tend to complete more MSP lab activities than required and thus spend around 2 hours or more working each week. We created the MSP assignments to take students about the same total time per week as the traditional OLP approach.

5.4.1 Analysis and Procedure

To calculate the total time students spent on each MSP lab activity, we looked at the timestamp metadata for each develop and submit. We calculated the time spent between submits by calculating the difference between each timestamp and then summed the differences. In our calculations, we excluded differences that exceeded 10 minutes, assuming that the student took a break from working. Note that our calculations are thus

an understatement, as some breaks may have been the student working or researching the problem. Additionally, we cannot capture the time the students spent working or thinking about a problem before the first submit or develop.

5.4.2 Results

Figure 5.1 summarizes the average time spent by students on each MSP lab activity. Note that since all MSP lab activities are considered in this calculation, including the "easy" introductory MSP lab activities which require minimal time, the averages are likely an undercalculation. The x-axis represents the university and the y-axis is the time spent in minutes. Across all universities observed, students spend an average of 12 minutes working on MSP lab activities with universities 1 and 6 slightly pulling the averages down.

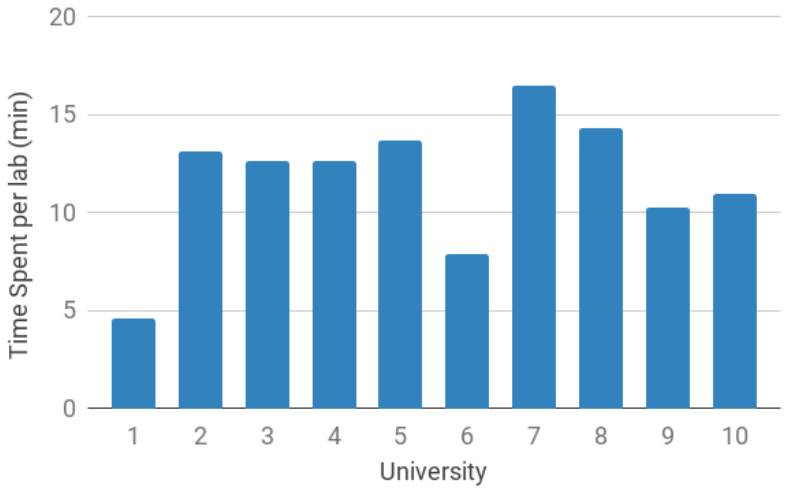

Figure 5.1:Average time spent by students on each MSP lab activity. Students with 0 submits or 0 time spent were excluded from calculations.

Figure 5.1 shows that many universities share similar time spent averages. Assuming that students are given 5 MSP lab activities each week, students should be working on programming assignments for about 1.5 hours. There are a few exceptions,

like universities 1 and 6, but a more detailed analysis is required to determine why the average time spent per lab for universities 1 and 6 are much lower compared to the other 8 universities.

5.5 How Many Days Before the Due Date Do Students Start Working on MSP Lab Activities?

For CS1 taught at our university, students were given one week to work on MSP lab activities. Given this setup, we found that students began working about 2.5 days before the due date. For example, if MSP lab activities were given to students on a Tuesday (due the following Tuesday), students began working on Sunday. Our previous study showed that MSP lab activities were helpful to get students to begin working earlier, but is this the same for other CS1 courses?

5.5.1 Analysis and Procedure

To calculate when a student began working on MSP lab activities, we first determined each students' first activity timestamp. Once we knew when the first activity happened, using this in combination with knowing the MSP lab activity due date, we computed the number of days each student began working on MSP lab activities before the due date. Note that zyBooks does not keep track of MSP lab activity due dates, so we had to calculate due dates based on student activity. We found the two most active dates for runs (days that had the most submits and develops by unique users) and then chose the later date as the due date for that MSP lab activity. We checked our due date calculations by comparing our results with known due dates of prior MSP lab activities and had ~90% accuracy. Most inaccuracies were +/- a day and affected MSP lab

activities given at the beginning of the term -- likely caused by students adding/dropping the course at the start of the term.

5.5.2 Results

Figure 5.2 summarizes the average number of days students began working on MSP lab activities before the due date. Using "NASA countdown" terminology, we use "T-2" to mean two days before the due date. The x-axis is the university and the y-axis is the number of days before due.

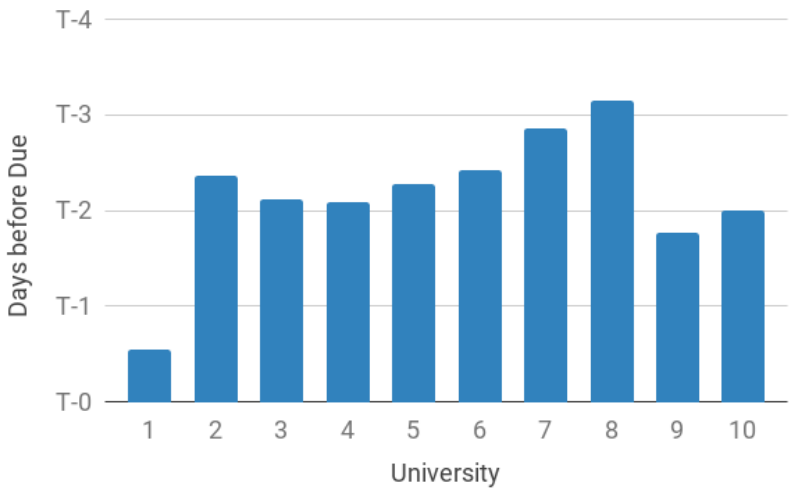

Figure 5.2: Average T-X days prior to the due date students began working on MSP lab activities.

Figure 5.2 shows that on average, students begin working on MSP lab activities about 2 days before the due date, with the exception being university 1 which seems to have most students starting on the due date. Note that this is the best approximation we can make without specifically knowing the due dates for all MSP lab activities. Also note that the data given could be slightly off as we do not know the duration students have to work on MSP lab activities - one week, three days, etc.

5.6 How Do Students Score on MSP Assignments?

Though we want each MSP lab activity to be challenging, allowing students to learn programming, we also need to be sure that the MSP lab activities are not excessively challenging. We want to know how students are performing on each MSP lab activity.

5.6.1 Analysis and Procedure

To calculate student scores on each MSP lab activity, we used the metadata for max score and current score. First, we found each students' highest scoring submit for each MSP lab activity and divided that value by the lab activity's max possible score. This results in the highest submit percentage for each student. Finally, we average across all students and all MSP lab activities.

5.6.2 Results

Figure 5.3 summarizes the average percentage score on each MSP lab activity. The x-axis is the university and the y-axis is the MSP lab activity's percentage score.

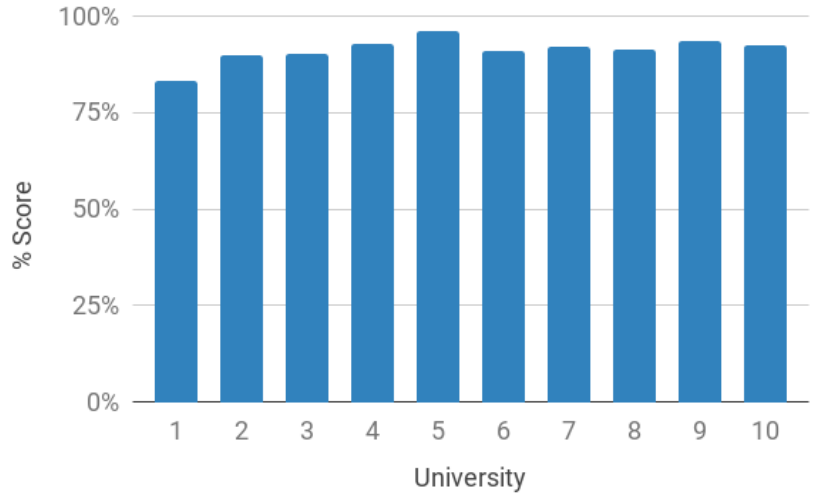

Figure 5.3: Average percentage score on MSP lab activities.

We are pleased to see that students are performing well on MSP lab activities. On average, across all universities, students score 91% on MSP lab activities. This high score indicates that students are completing MSP lab activities and are (hopefully) learning how to program successfully.

5.7 DISCUSSION

Based on the results of the analysis performed in this work, we can conclude that MSP usage is similar across all universities. We can see that MSP-trained students at other universities are getting the same benefits that we observed in the CS1 courses taught at our university. Though we were only able to perform three different analyses, compared with the several of our previous work, we still believe this to be a good indication that an MSP approach can be used in CS1 and students will benefit. For future work, we hope to extend this analysis and obtain the data necessary to perform the additional analysis done in our other work. Given the large amount of data used in this extension, we are now more confident in the previous results we obtained when only considering our university.

5.8 CONCLUSION

In this work, we performed several analyses on MSP lab activities from other universities. We found that the results we obtained in this work are similar to the results we observed in our previous work only considering CS1 at our university. This analysis improved our confidence that students are using MSP lab activities in beneficial ways. We see that students are spending sufficient time working on weekly MSP assignments (~1.5 hours), that students begin working about 2 days before the MSP lab activity's

deadline, and students are scoring high grades on MSP lab activities (91% avg). Based on the positive results of this work, we are encouraged to continue improving MSP lab activities and studying them more in depth. Already, we have switched all CS1 offerings at our university to using an MSP approach and we encourage others to consider using an MSP approach in their CS1 course as well.

Chapter 6. An Analysis of Using Coral Many Small Programs in CS1

6.1 Introduction

Previous work [3] detailed in Chapter 5 showed that an MSP approach can work across universities and across different programming languages. As we continued to apply new intervention techniques to try and improve our CS1 even further, we decided to incorporate a Coral-first approach alongside an MSP approach. 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 an MSP approach which involves assigning students multiple smaller assignments instead of only giving them one large assignment each week. In Chapter 6, 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.

6.2 Coral Programming Language

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 [16]. Coral is free to use and resembles popular commercial programming languages like C++, Java, and Python, allowing for a smooth transition between languages.

Figure 6.1 shows an example of an introductory program written in C++, Java, Python, and Coral. This program requires the student to prompt the user for an integer input, declare a variable "wage" and assign the input to "wage", increment the input value by 10, and then output the final result of "wage" to output. Although this is an intro programming assignment, C++, Java, and Python have many nuances and intricacies that a new programmer should not need to worry about -- function calls, strange symbols (stream operators '<<', semicolons ';', braces '{}'), class methods ('Scanner.nextInt()'), and more. Coral is simple to read and built to look like pseudocode so students can understand the logic easier without focusing on language semantics.

C++

```
#include <iostream>

using namespace std;

int main() {

  int wage;

  cout << "Enter wage: "
  cin >> wage;

  wage = wage + 10;
  cout << "New wage: ";
  cout << wage;

  return 0;
}
```

Java

```
import java.util.Scanner;

public class Main {

    public static void main(String []args){

        Scanner myScanner = new Scanner(System.in);

        System.out.println("Enter wage: ");
        int wage = myScanner.nextInt();

        wage = wage + 10;

        System.out.println("New wage: ");
        System.out.println(wage);

    }
}
```

Python

```
print ('Enter wage:', end='')

wage = int(input())
wage = wage + 10

print('New wage:')
print(wage)
```

Coral

```
integer wage

wage = Get next input
wage = wage + 10

Put "New wage: " to output
Put wage to output
```

Figure 6.1: Sample introductory program written in C++, Java, Python, and Coral (listed left to right).

The Coral 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.

Figure 6.2 shows the online, web-based Coral simulator. The simulator comes with an area for coding on the left, and on the right are areas for real time variable values, user input, and program output. There is also the option to execute the program immediately or step through the program's execution to see real time updates to variable values. Figure 6.3 shows Coral's equivalent visual flow chart language.

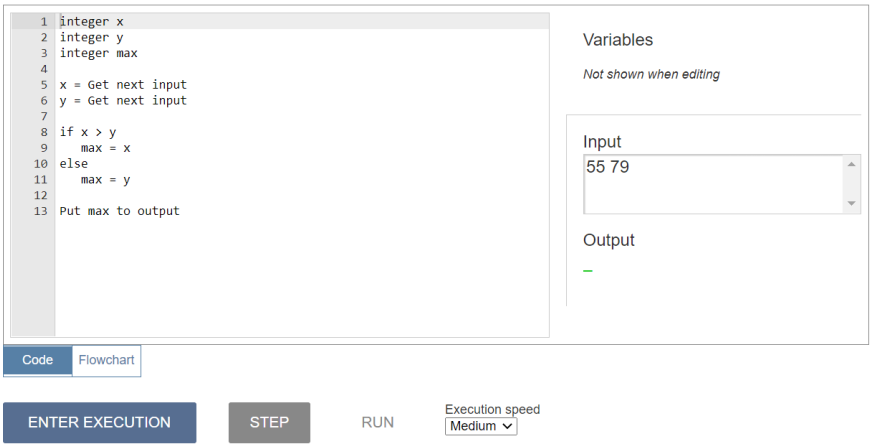

Figure 6.2: Coral's online web-based simulator.

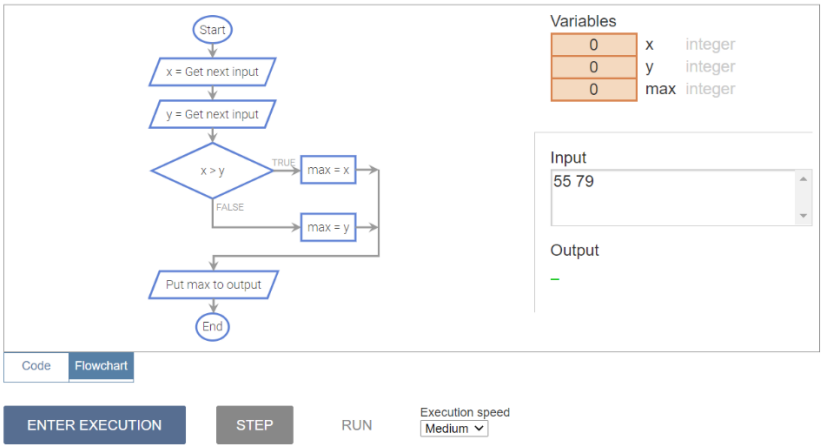

Figure 6.3: Coral's online web-based visual flowchart simulator.

The authors of Coral published an initial work showing Coral's ease of use and we decided to apply the language in our CS1 [20]. We had considered using other

introductory programming languages like Snap [61] or Scratch [58], but we found they are not designed for a CS1 class. We began using Coral at the start of the class and then switched to C++ after 3 weeks.

6.3 METHODOLOGY

6.3.1 COURSE

We analyze a Spring 2020 CS1 course taught at the University of California, Riverside. The CS1 course 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). The CS1 course 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.

6.3.2 EXPERIMENT DETAILS

For one CS1 class section we taught Coral for the first 3 weeks and then switched to C++ (hybrid Coral/C++ group) 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 the 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.

6.3.3 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.

6.4 Student Grade Performance

We gathered gradebooks for each class 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.

6.4.1 Results

Figure 6.4 shows our results. The pure C++ group data is shown on the left, dark-blue bars and hybrid Coral/C++ group data is shown on the right, light-blue bars. The grade percentage is on the y-axis and the week number is on the x-axis. A total grade average column is added to the end of the chart. Table 6.1 summarizes the average grades for all class categories.

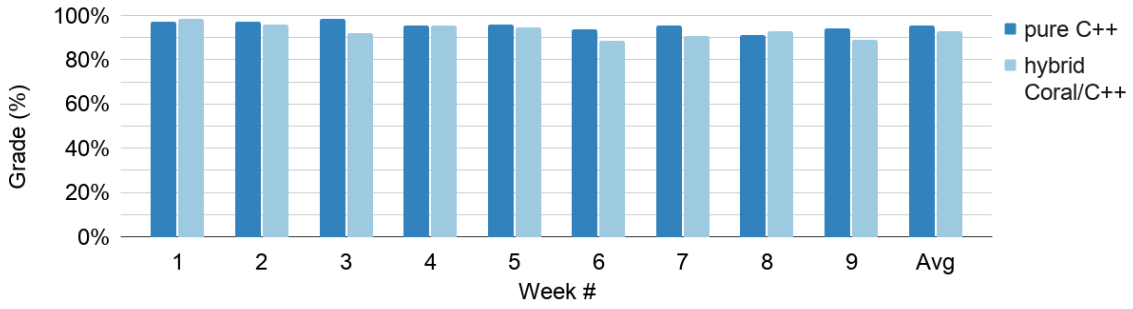

Figure 6.4: Grade performance results: Both the pure C++ group (avg. 97%) and the hybrid Coral/C++ group (avg. 95%) scored equally well on MSP assignments.

Table 6.1: Student grade performance on all categories of our CS1 class. Students that did not take the midterm exam or the final exam are excluded from calculations. p-values denoted with * are nearing significance ($p < 0.05$).

Class category	Pure C++	Hybrid Coral/C++	p-value
Total class grade	89%	96%	$p < 0.001$*
Final exam	86%	91%	0.009*
Midterm exam	84%	97%	$p < 0.001$*
Participation activities	96%	96%	0.727
Challenge activities	95%	96%	0.875
Lab activities	97%	95%	0.066

Figure 6.4 shows that both the pure C++ group (97%) and the hybrid Coral/C++ group (95%) do equally well on weekly MSP assignments. Table 6.1 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.

6.5 Time Spent Metrics for Weekly MSP Assignments

We expect students to spend between 2-3 hours working on MSP assignments 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.

6.5.1 Results

Figure 6.5 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. The pure C++ group data is shown on the left, dark-blue bars and the hybrid

Coral/C++ group data is shown on the right, light-blue bars. Students that did not attempt any MSP lab activities for the given week were excluded from the calculations.

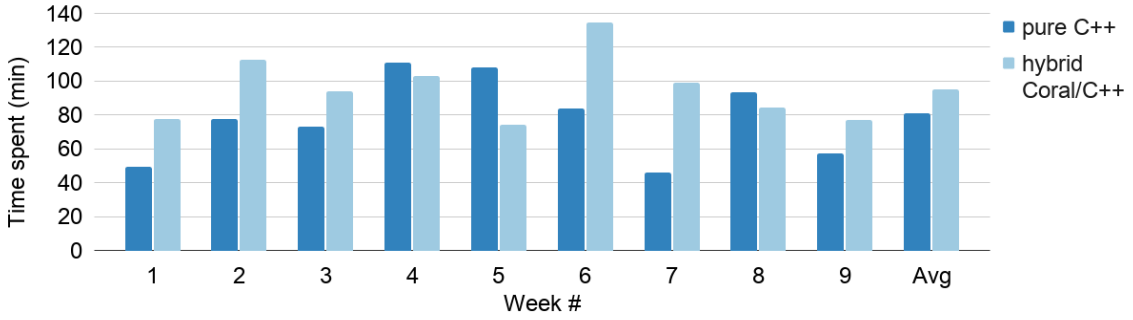

Figure 6.5: Time spent results: The hybrid group (avg. 95 min) spends slightly more time working on MSP assignments each week than the pure C++ group (avg. 81 min).

Figure 6.5 shows that the pure C++ group (81 minutes) spends less time working on MSP assignments each week than the hybrid Coral/C++ group (95 minutes).

6.6 Activity Run Metrics for Weekly MSP Assignments

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.

6.6.1 Results

Figure 6.6 displays our results. Develop runs are indicated by the dark-blue bars at the bottom and the submit runs by the light-blue 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. Students that did not attempt any MSP lab activities for the given week were excluded from the calculations.

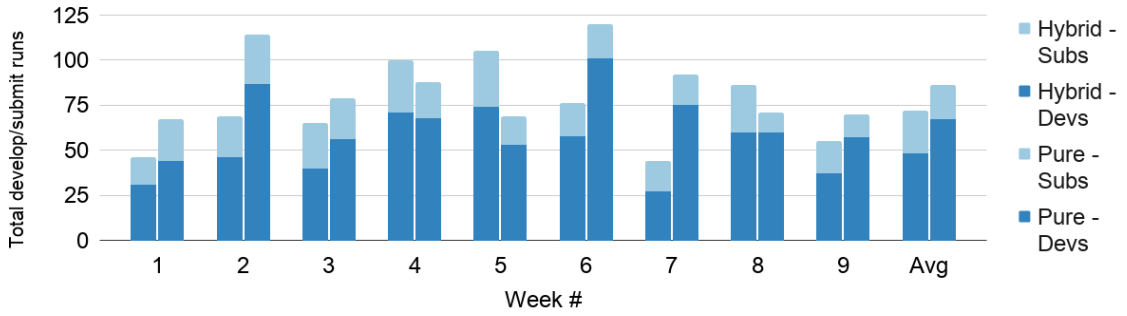

Figure 6.6: Activity run results: The pure C++ group (avg. 48dev / 24sub) develops less and submits more than the hybrid Coral/C++ group (avg. 67dev / avg. 16 sub).

Figure 6.6 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.

6.7 Start Date Metrics for Weekly MSP Assignments

Each MSP 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. To calculate students' average start date each week, we found each students' earliest activity timestamp for a lab activity from the given MSP assignment, calculated the difference between that and the due date, and averaged the differences.

6.7.1 Results

Figure 6.7 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. The pure C++ group data is shown on the

left, dark-blue bars and the hybrid Coral/C++ group data is shown on the right, light-blue bars. Students that did not attempt any MSP lab activities for the given week were excluded from the calculations.

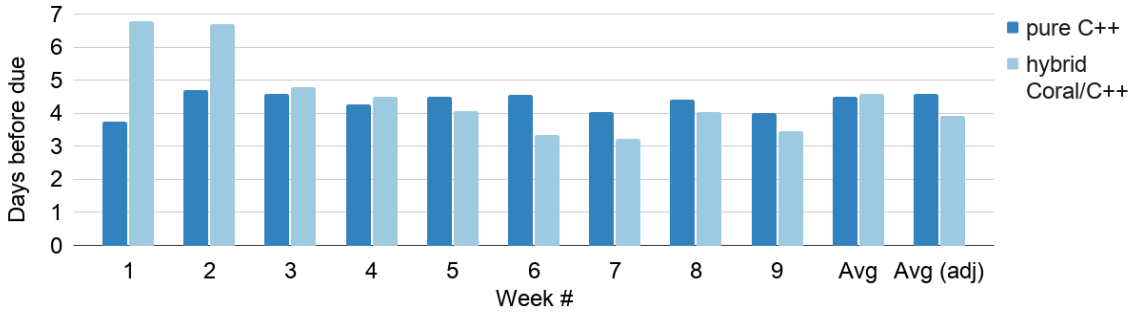

Figure 6.7: Start date results: The pure C++ group (avg. 4.5days / 4.8days adj.) begins working earlier than the hybrid Coral/C++ group (avg. 4.6days / 3.9days adj.).

Figure 6.7 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 6.7 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).

6.8 Pivot Metrics for Weekly MSP Assignments

A pivot is when a student switches from one lab activity to another without completing (scoring 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.

6.8.1 Results

Figure 6.8 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 given in week

6. The pure C++ group data is shown on the left, dark-blue bars and the hybrid Coral/C++ group data is shown on the right, light-blue bars. Students that did not attempt any MSP lab activities for the given week were excluded from the calculations.

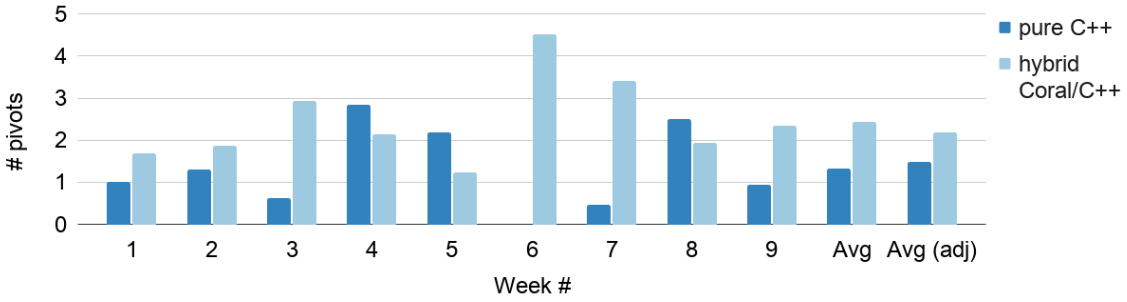

Figure 6.8: Pivot results: The hybrid Coral/C++ group (avg. 2.4 / 2.2adj.) pivots more than the pure C++ group (avg. 1.3 / 1.5adj.) each week.

Figure 6.8 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).

6.9 Discussion

This is the second time that we have taught CS1 via a hybrid coral-first approach combined with an MSP approach. The first time was in Fall 2019 and overall things went well but we had learned a lot from our experience. The first time we used a hybrid approach, we had taught Coral for the first 5 weeks of the 10-week quarter and then switched to C++ during the latter 5 weeks. Our major mistake was covering the entire CS1 content first in Coral (i.e. cramming 10 weeks of material in 5 weeks) and then covering the same content again in C++: input/output, variables, branches, loops, functions, and vectors. By the time students were learning C++ during the second half of

the quarter, they were exhausted as they were repeating the same content and had additional lab activities to complete. Students were still performing well, but we noticed a much smaller completion percentage of MSP assignments and class content during weeks 5-10 compared with the other class sections using a pure C++ MSP approach. We also got lots of student feedback via surveys saying things like "It was a great way to introduce concepts but I wish we spent more time using C++" or "Pros: I learned faster for C++. Same concept so C++ similar to it were easy to understand. Con: Time constraint. I think C++ needs more time to study instead of just 5 weeks." Our students seemed to grasp the benefits Coral provided, however improvements needed to be made. As such, this led us to improve our approach and use Coral for the first 3 weeks only and then transition to C++ as described in Chapter 6.

6.10 CONCLUSION

In this work, we shared our experience using a hybrid Coral/C++ MSP 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.

Chapter 7. Concise Graphical Representations of Student Effort on Weekly Many Small Programs

7.1 Introduction

Beyond seeing final submissions, many instructors want insight into how students went about the process of writing their code -- when did they start, how often did they test, how correct was their code along the way, how much time did they spend overall, etc. As such, some now require students to use version control software like github, to at least see some versions of the code during development. However, program auto-graders provide a distinct opportunity for such insight, having grown tremendously in use in recent years, including new commercial tools like zyBooks [72], Gradescope [30], Mimir [41], Vocareum [66], CodeLab [65], and MyProgrammingLab [46]. Some of those also have development environments so that all a student's programming activity can be recorded: "develop" runs while the student is still developing and testing their code, and "submit" runs where they submit code for auto-grading. Non-commercial systems also record develop runs and/or submit runs, like Runestone [57] and BlueJ [12]. Such recording opens new possibilities for instructors to gain the desired insight in student coding.

Meanwhile, hundreds of schools, including ours, have converted to an MSP approach (zyBooks alone reports over 200 schools; many more exist). Thus, not only do we want insight into our students' programming process, but we want that for multiple MSP lab activities per week -- to see which they started on, how they switched between

programs, and so on. A table of statistics is too hard for an instructor to process and loses too much information. Thus, in 2018, we began developing a script to process the log files from the popular auto-grader that we use and convert to a graphical representation that we call "programming workflow charts". We have found those charts provide instructors with tremendous insight, allowing a quick determination of how a class is doing (starting on time? spending sufficient time?), but also to quickly see a particular student's effort (such as when a student comes to office hours for help, or is requesting an extension) -- and even to detect some cheating cases. We even pull up the charts for the class and use them as a springboard to dive into a particular student's code (if they offer). Students find the workflow charts "cool", and we believe such charts, if used properly in a class, may even reduce some cheating in the future due to showing students that instructors can see their effort.

Chapter 7 describes the goals of such a representation, the evolution of our representation to its current status, various design trade-offs, our current usage, and numerous possible future uses in CS1 classes. We plan to create a website for any instructor to upload such log files to gain insight on their own class' performance.

7.2 METHODS

7.2.1 DATA COLLECTION

To collect the data required to generate our workflow charts, we obtained from zyBooks log files for all MSP assignments. The file was in csv format and contained all develop and submit runs for every lab activity in our class. A develop run is when a student tests their code in the built-in IDE without receiving a grade. A submit run is

when the student submits their code for grading. Each student activity entry contains metadata such as the title of the lab activity, the user ID, a timestamp, a link to the source code for that run, and for submit runs also contains a score, a max score, and a list of test cases including which were passed or failed.

7.2.2 Time Spent Calculations

An integral calculation for all workflow charts is the time spent by students on each lab activity. To calculate time spent, we gathered all student activity and calculated the difference between timestamps. Each difference was then summed together to yield a final calculation of the total time spent. Note, that if the difference between two timestamps exceeded 10-minutes, we excluded the time from our calculations to be conservative as the student likely took a break or went to work on something else. Furthermore, we cannot capture the time a student spent working before their first activity. As such, our data is likely an understatement.

7.2.3 Construction of Programming Workflow Charts

To generate each programming workflow chart, we first gathered all student activity for the quarter. From the metadata, using a combination of userID, labID, time spent, and knowing the week each lab activity was assigned, we grouped student activity to do the necessary calculations. To determine time spent see process listed in Section 7.2.2. To compute the number of develops and submits, we counted each activity and if there is a score associated with the activity, then the activity is counted as a submit, otherwise a develop. We calculate the percent scored by finding the highest submit score among all activities for that program. Finally, we separate activity by "workflow blocks,"

indicating that the student switched working between lab activities or there was a 10-minute gap observed between activity (10 minutes was chosen arbitrarily).

7.3 The Evolution of Our Workflow Charts

Our motivation for creating these representations was to understand how students were interacting with the MSP assignments. Based on end-of-the-quarter grades, we had seen that students were earning good grades and doing well on exams, but we lacked insight on questions like: How much time are students working on MSP assignments each week? What days did they work? Were students working on MSP lab activities in the order we listed them, or were they jumping among them? How often were they developing versus submitting?

We decided to pursue a graphical representation of the data to gain quick and concise insights into student effort on weekly MSP assignments. We used a Gantt chart as the initial motivation behind developing our workflow charts. A Gantt chart is a visual view of tasks scheduled over time [26]. Such a chart highlights important information like the start of a task, the end of a task, and the time spent per task in a single view.

In these workflow charts, we are able to see the time students spend per lab activity, the total number of develops and submits per lab activity, the score earned per working session, a summary of all activity for the week, and the pivot patterns students displayed.

Note that some figures in Section 7.3 that show the evolution of our charts may differ in example as we do not have records of all previously used iterations.

7.3.1 VERSION 1 -- CALENDAR VIEW

Figure 7.1 shows Version 1 of our workflow chart. Our initial thought was to display the data using a weekly calendar view to see data on all weekly lab activities for each student each week.

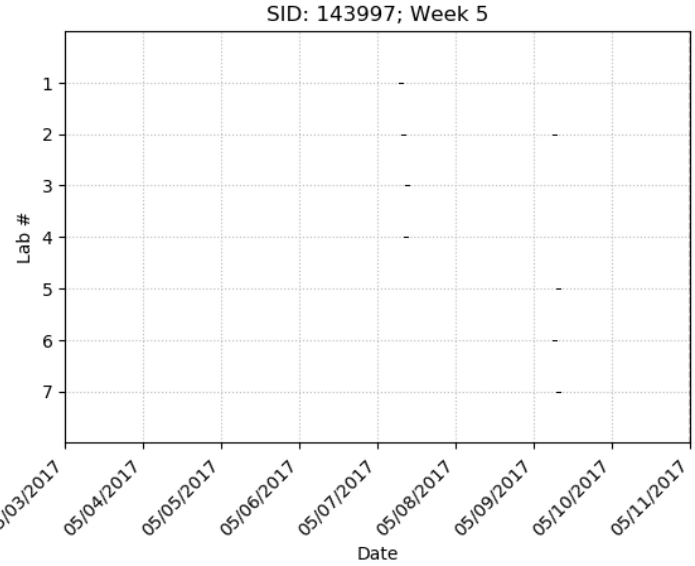

Figure 7.1: Version 1 of the workflow chart. An expanded calendar view with lab activities on the y-axis and days on the x-axis. Horizontal lines added to indicate when students worked.

As we were using the MSP approach, we had assigned students 7 lab activities per week. On the workflow chart, the lab activities are listed on the y-axis in ascending order and dates for the week are listed on the x-axis in ascending order. Horizontal lines are added to indicate the times students spent working on each lab activity. Each chart has a title with the student ID (anonymized) and the week the chart was generated for.

Unfortunately, upon initial inspection, the data is very hard to read and at a quick glance, it may even seem like the student did no work for the given week. In actuality, the data is present, but since the chart covers 7-days, the time increments in which the student worked are so small in comparison that they are almost not even visible on the

chart. Using the calendar view did not work as we intended, and we needed a better way to represent the data in a compressed way.

7.3.2 VERSION 2 -- COMPRESSED CHART

For Version 2, we needed a better way to represent the data for a given week. We compressed the chart by considering total time spent during the week instead of spreading out the data across the entire week as in Version 1. Lab activities are still shown on the y-axis, but the x-axis is now total time spent. Horizontal lines were still used to indicate time spent per each lab activity. Additionally, we put a percentage above each horizontal line to indicate the highest score a student earned after that session of working on that lab. Each chart has a title that summarizes data for the week, including the student's ID, the total time spent working on lab activities for the week, and the total number of develop runs (D) and submit runs (S). Each chart is read from left to right and from top to bottom. Figure 7.2 shows Version 2 of the workflow chart.

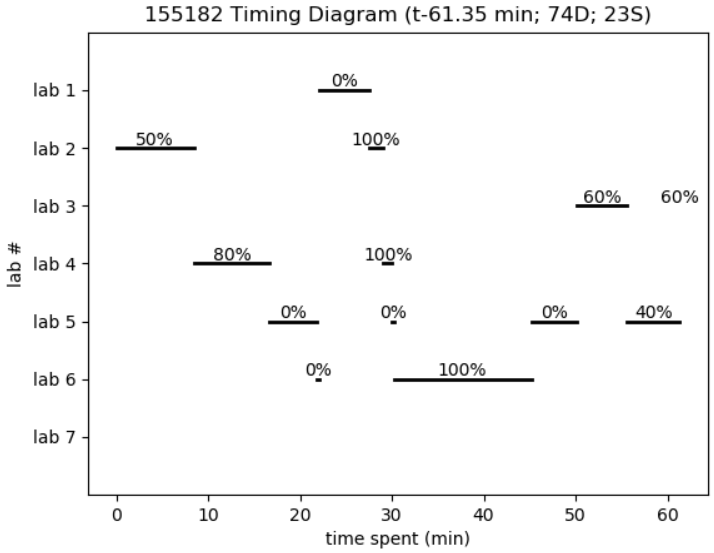

Figure 7.2: Version 2 of the workflow chart. Compressed chart only considering total time spent represented by a black horizontal line per lab activity and a completion score above.

Version 2 of the workflow chart provided insight into a students' workflow (how they worked on each lab activity during the week), but we soon found ways to get more information onto the chart while maintaining readability.

7.3.3 Version 3 -- Color / Score Per Submit Run / Statistics Per Lab Activity

Version 3 of the workflow chart improved clarity and readability. We added color to distinguish data for each lab activity, so when looking at charts for multiple students, an instructor could get a quick sense of which lab took most time -- if seeing a lot of purple, an instructor might know that lab activity 2 was the most time consuming. Next, we added labels on the right of the chart to summarize data for each lab activity, including the lab activity's final score, the time spent, and the total number of develop and submit runs. We added a grid to enable more accurate readings. Finally, we made a change to the way we considered student work sessions throughout the week. This change is represented in the chart by some horizontal lines having multiple final score percentages listed above them. This is explained later in the following paragraphs. The title of each chart was changed for improved readability. Figure 7.3 shows Version 3 of the workflow charts.

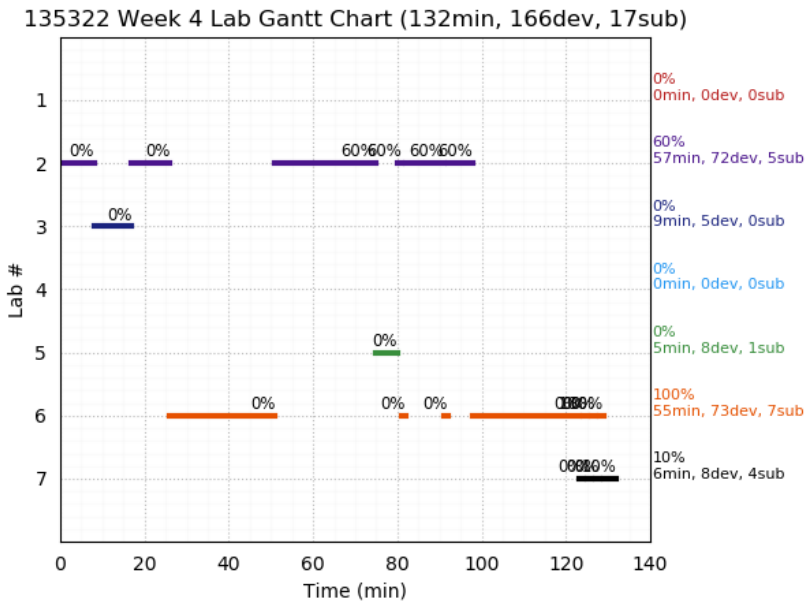

Figure 7.3: Version 3 of the workflow chart, adding color, summary statistics on the right, gridlines, and more submit scores.

Version 3 of the workflow chart required many design considerations. First, when thinking about how to clearly denote which data corresponded to each specific lab activity, we thought of using color, line styles, or a combination of both. Different line styles proved to yield a cluttered appearance, and some were hard to distinguish. They also didn't enable easily seeing the most/least time-consuming labs across multiple students. A tradeoff here relates to some people potentially having less ability to distinguish color, and loss of info when printed in black and white. A second design consideration was related to the grid. Adding the grid added more clarity to the chart, but in earlier iterations, the grid also decreased data visibility. We initially set the grid color to be too dark and also with a higher volume of tick marks that were unnecessary. After testing different color shades and tick mark frequencies, we chose a lighter color for the grid and reduced the tick marks to achieve the accuracy we wanted.

Finally, we changed the way we thought about how to represent students working on each lab activity, referred to as student work sessions. At first, we considered a student work session to end when the student began working on a different lab activity i.e. submitted code for lab activity 1 and then developed code for lab activity 2. Upon deeper analysis, we recognized a common scenario where students would begin working on a lab activity, leave to take a break, and then return to work on the same lab activity. We consider this scenario important to denote, so we considered a work session to also end if the time between two activities was more than a 10-minute threshold. We thus showed the score at the end of every session, which is why the figure above shows multiple 60% values on a single bar of lab activity 2, for example. This distinction does lead to some clutter if the student has many work sessions back-to-back (as can be seen in Figure 7.3 for lab activity 7), but we felt the distinction helped instructors to better understand student workflow patterns.

7.3.4 Version 4 -- More develop/submit details

Version 3 provided the foundation for all the following updates of our workflow chart. As we used these charts for analysis in our teaching each quarter, we noticed a lack of insight on student behavior during each work session. Version 3 summarized data for each lab activity at the end of the week, but not during the week. As such, in Version 4, we wanted our chart to add further insight into student develop and submit runs during work sessions. To accomplish this, we added indicators on the time spent data lines for when a submit took place. These are indicated in a few different styles as seen in Figure 7.4 and Figure 7.5. We also added text data on the number of develop and submit runs

during each work session underneath each time spent line. Finally, we made minor adjustments to the labels on the right of the chart such that each feature was on it's own line for additional clarity.

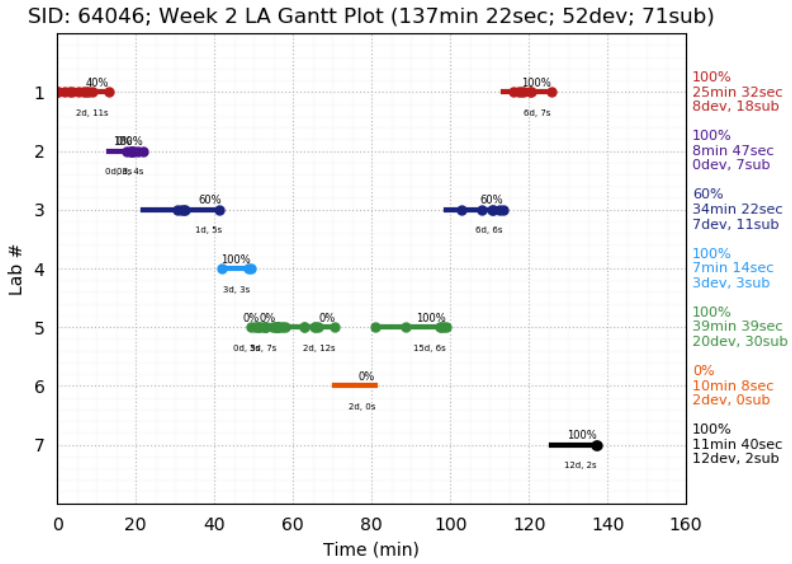

Figure 7.4: Version 4a. Used large filled in points to indicate a submit run, added text to summarize student activity per work session, minor adjustments to chart labels.

Version 4a shown in Figure 7.4 uses large filled in points to indicate submit runs. Using this indication style made it easy to see submit runs but added clutter due to the size of the points. Also, this indication did not show develop runs.

Another approach we took, seen in Figure 7.5 Version 4b, uses a small point with a tail and a character label listed below to denote a develop or submit run. A develop run is indicated with the 'D' character and a submit run is indicated by the 'S' character. By reducing the size of the point and adding a character, the clutter was lessened and the distinction was clear. Unfortunately, with the additional markings, it became difficult to visually separate an 'S' from a 'D.'

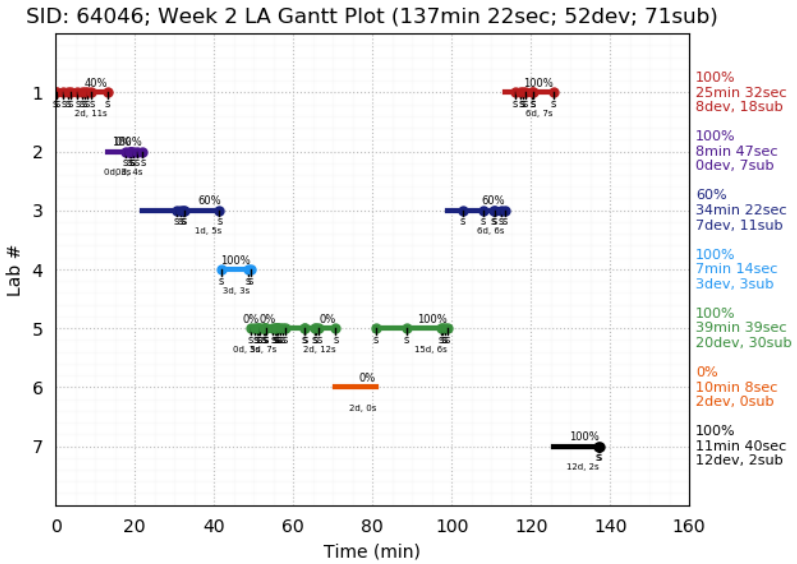

Figure 7.5: Version 4b. Used small points with a 'S' label to indicate a submit run and a 'D' label to indicate a develop run. Other updates are similar to Figure 7.4.

We also experimented using other shapes as indicators like squares, diamonds, stars, 'X's,' and open points, but none worked out. In both versions, the text indications for total develop and submit runs below the data lines were helpful. There were some situations where this data would overlap, making some content difficult to read, but this didn't happen very often.

7.3.5 Version 5 -- Tick Marks for Develop and Submit Runs

In Version 5, we effectively displayed develop runs and submit runs during weekly work sessions. Instead of using points to indicate a develop run or a submit run, we used small tick marks: A tick below the line indicates a develop and a tick above the line indicates a submit run. This indication is simple and quickly understood. Even with a high density of student activity, the chart was still readable. Figure 7.6 shows Version 5.

There was one other design consideration we tested for Version 5. Before putting straight tick marks above and below the data line, we used straight and diagonal tick

marks to indicate a develop run and submit run respectively. This worked when the density of activity was low but became difficult to differentiate a straight tick from a diagonal tick when the density was high.

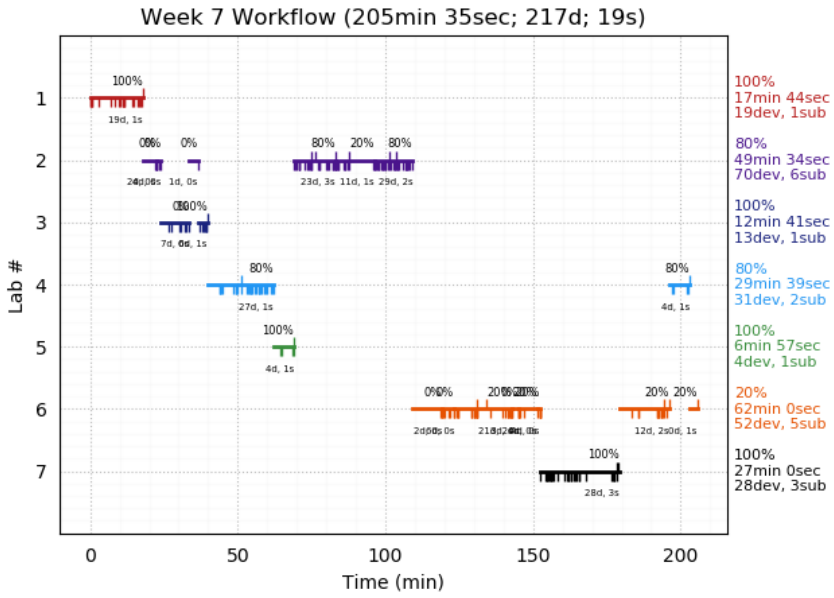

Figure 7.6: Version 5. Added tick marks below the time lines to indicate develop runs, and tick marks above the time lines to indicate submit runs.

7.3.6 Version 6 -- Pivot Indicators

Version 6 introduces the ability to identify pivots. One unique benefit of the MSP approach is the ability to pivot, which is when a student switches from one lab activity to another before scoring 100% on the lab activity being worked on. If a student gets stuck on a lab activity, they can just move on to another, often coming back to finish the earlier lab activity (or having gotten help in the meantime). We added arrows on the workflow chart to indicate when pivoting occurs. Figure 7.7 shows Version 6 of our workflow chart.

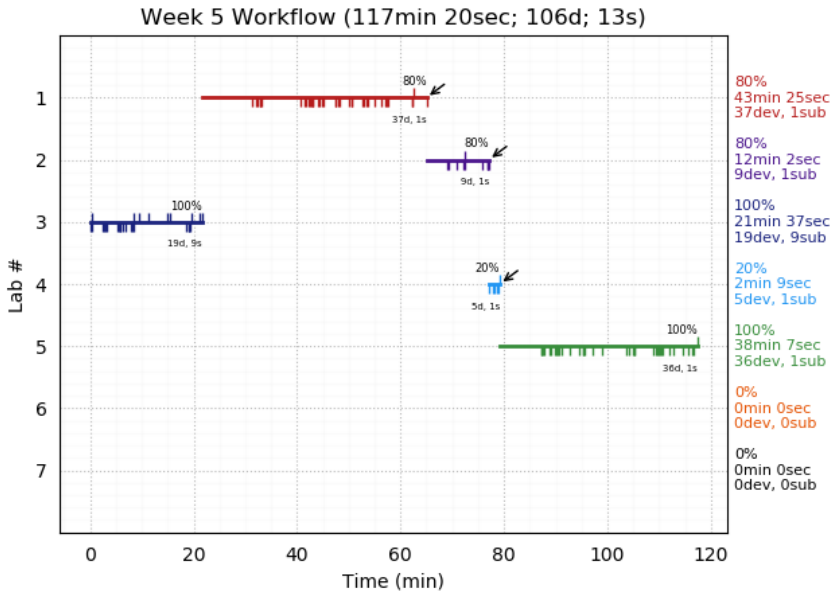

Figure 7.7: Version 6. Added arrows to indicate pivots.

In Figure 7.7, the first pivot can be seen on lab activity 1 since the student only scored 80% and then switched to work on lab activity 2. Pivoting arrows are useful if an instructor is interested in them, but adds a small amount of clutter. As such, we added a flag to control generation of these pivot arrows on the workflow chart.

7.3.7 Version 7 -- Dual Time and Weekly View Charts / Classification Features

In Spring 2021, we updated our workflow charts to Version 7. Version 7 combines a time-like view with a weekly-calendar view. Version 6 showed how long students were working on MSP lab activities but was missing information on when in the week students were working. Are students completing all 5-7 lab activities in one day or are they spreading them out over a couple days? Initially, we solved this issue by having two separate charts and we would compare them side-by-side. One chart displayed time on the x-axis as seen in Figure 7.7 and another would display weekly days on the x-axis

like Figure 7.1. Version 7 combines these two views to now see how long students are working and when students are working.

Furthermore, we introduced simple classification features and also include them in the bottom half of the chart. Some other details that changed in Version 7 are now the charts only show the score when it either improves or at the end of a work session which improves clarity. For the bottom chart, the same colors are used between both charts to indicate which lab activities are shared between the views and a vertical dotted line is added on the weekly-view chart to indicate the deadline for the MSP assignment. Features currently included in this chart are:

- Start: when the student begins working on more than 10% of their total activity [early, on time, late, day of]
- End: when the student finishes working on more than 90% of their total activity [early, on time, late, day of]
- Work type: if the student completes all their work during one day or spreads their work out over more than 2 days [sprint, marathon]
- # Subs: the number of submits the student has compared to the rest of the class [low Q1, avg Q2, high Q3]
- Time spent: the time spent working on the MSP assignment compared to the rest of the class [low Q1, avg Q2, high Q3]
- Suspicious: if the student scores 100% of their points in under 15 minutes [true, false]

Figure 7.8 shows Version 7 of our programming workflow charts.

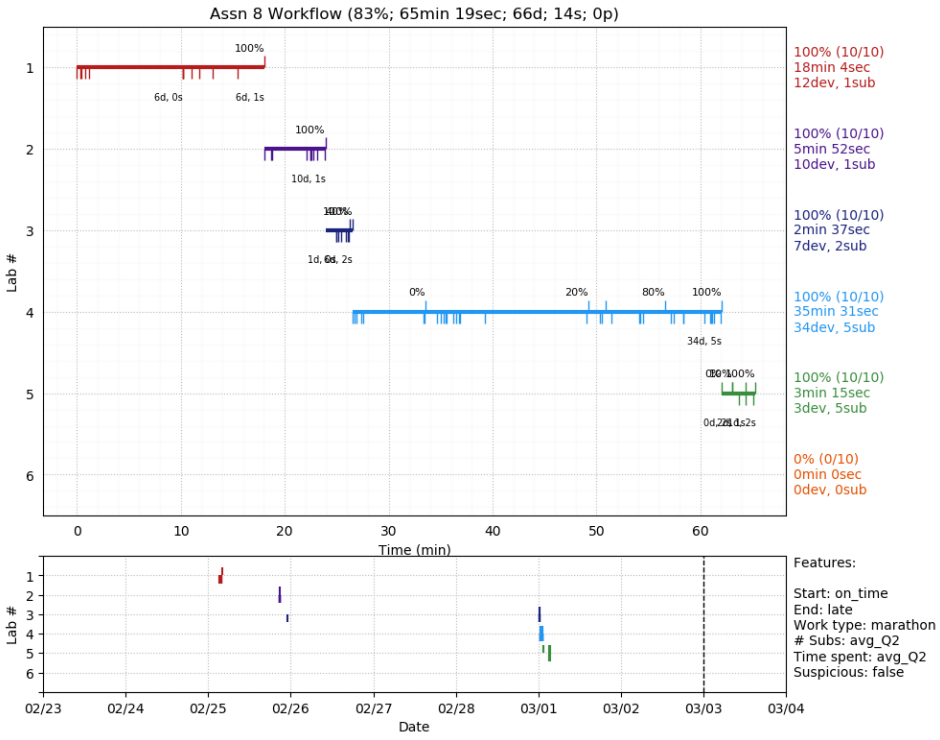

Figure 7.8: Version 7. Time view combined with a weekly view and addition of workflow classification features.

7.4 Current Uses and Discussion

Version 6 (Version 7 as of Spring 2021) of the workflow charts has the information we desired, available at a quick glance. We can see summary data for the week, specific data for each lab activity, and can even see special information like pivots. Section 7.4 discusses our primary uses of these workflow charts.

7.4.1 Understanding Student Effort

From the beginning, our motivation was to create a visual representation of data to understand student effort on our MSP assignments. These workflow charts help us see lots of meaningful data quickly and accurately in a single location. We can pick any week of the quarter and any student and see why they may be struggling or even performing

better than other students in the class. For example, Figure 7.9 is what we consider a 'healthy' workflow chart. The student spent a good amount of time working on the MSP assignments (94 minutes), they scored 71% of the total points (which is considered 100% score for the week using a 70% threshold), and they worked linearly through each MSP lab activity, starting on lab activity 1 and ending on lab activity 5.

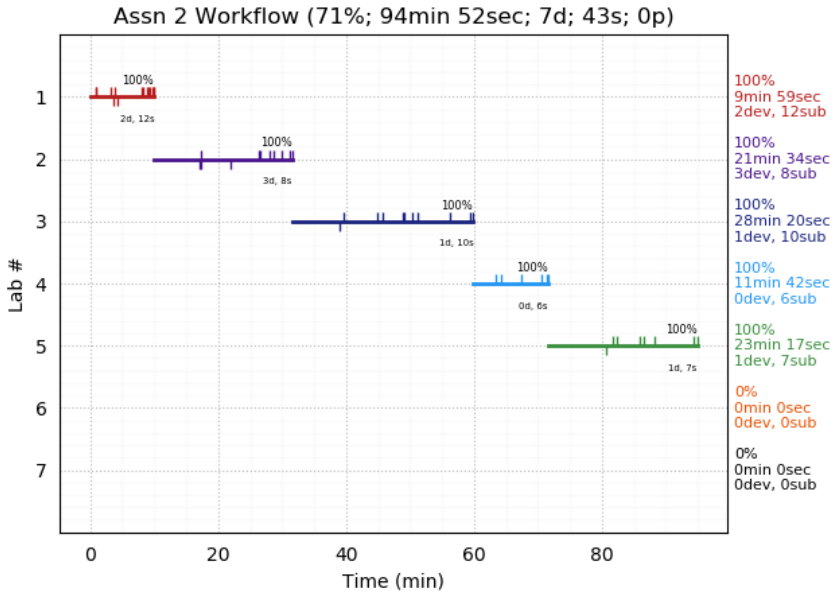

Figure 7.9: 'Healthy' programming workflow chart from a CS1 class.

Figure 7.10 in contrast, shows a student that is likely struggling on MSP assignment 2, specifically on lab activity 2. The student spent 88 minutes (almost half the time) of the total 195 minutes solely working on lab activity 2 but still scored 0%, they pivoted multiple times while working and still did not improve their score, and they ended the week with a 57% instead of the required 70%.

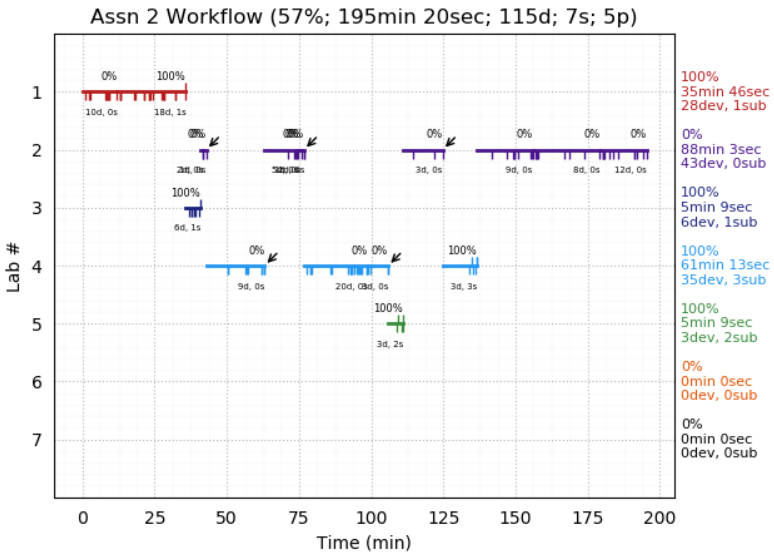

Figure 7.10: Programming workflow chart showing a student likely struggling with lab activity 2.

At our university, we primarily only use an MSP approach in CS1, however, some instructors include a few OLP assignments as well. Figure 7.11 shows that the workflow charts can also provide insight into OLP assignments too. We have already used these charts for many analyses regarding research, individual student considerations, and to generally improve our MSP lab activities and our CS1.

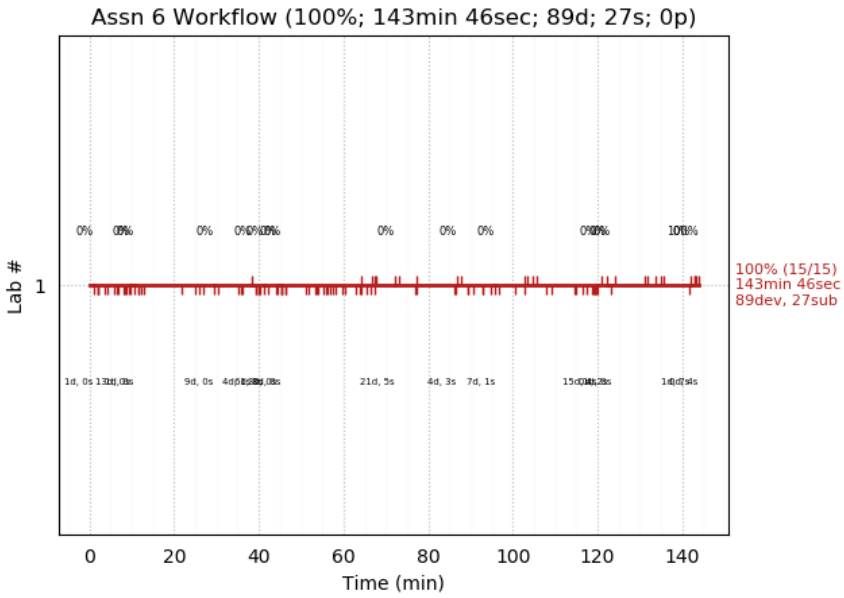

Figure 7.11: Programming workflow chart for an OLP assignment in a CS1 class.

7.4.2 Detecting Unallowed Collaboration

In 2017, we began allowing our students to collaborate when working on lab activities. We allow students to collaborate only if they do a majority of the work and they indicate on their submissions who they worked with. We have a variety of ways to ensure each student is submitting ethical work, and among them are using these workflow charts to visually notice any irregularities. Figure 7.12 shows a potentially 'suspicious' programming workflow chart. This chart is suspicious because the student scored 71% (full credit with a 70% threshold) with only spending 5 minutes total on the given MSP assignment. It is possible that the student had previous experience and was actually able to complete the assignment very quickly, as this is from week 2 when lab activities are fairly simple, or they could have completed their coding outside of the zyBooks IDE and copied their solution in. Either way, instructors can quickly see which workflow charts

need attention and investigate as needed. Recently, we started showing students these generated charts and having them call out any charts that look 'weird' as a class exercise.

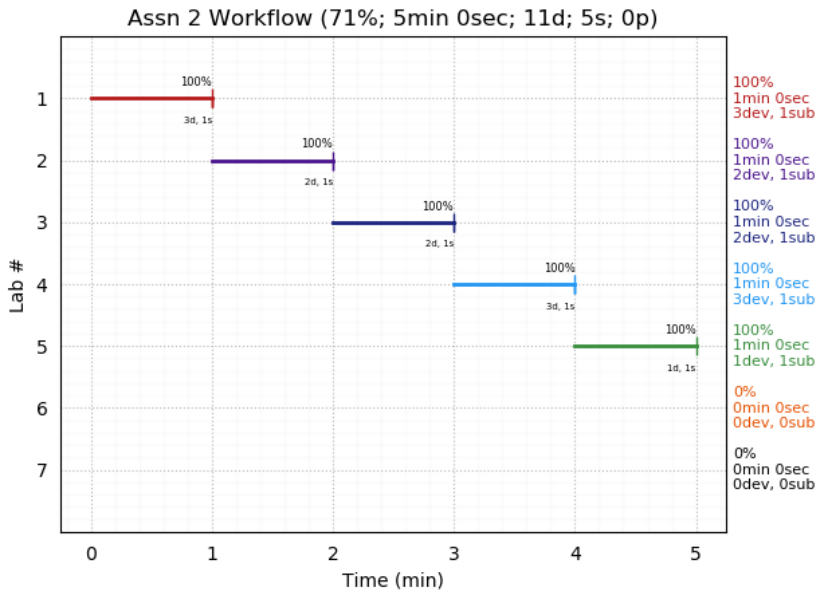

Figure 7.12: Potentially 'suspicious' programming workflow chart in a CS1 class.

7.4.3 Student Classifications

One other way that we have begun using these charts is to create student classifications to help us identify students that may be struggling. In a 10-week quarter, we typically generate over 1,000 workflow charts. If we can use these charts to make meaningful and accurate classifications, then we can identify struggling students early and provide additional resources that will help them succeed. Some classifications that we currently use are when do students begin working, do students complete all lab activities in a single day or spread them out, and how much time do students take to complete all lab activities.

7.4.4 INTERACTIVE WEB PAGE

We are creating a tool to generate a web page to share these workflow charts with our students and the community. Instructors can upload the auto-grader's log files for a week's lab activities, and the charts are automatically generated on an interactive web page. Although our university uses zyBooks' program auto-grader to produce csv log files, our tool can work with any auto-grader so long as it can produce a log file with the same format as needed by the tool. Figure 7.13 shows a high-level description of this process.

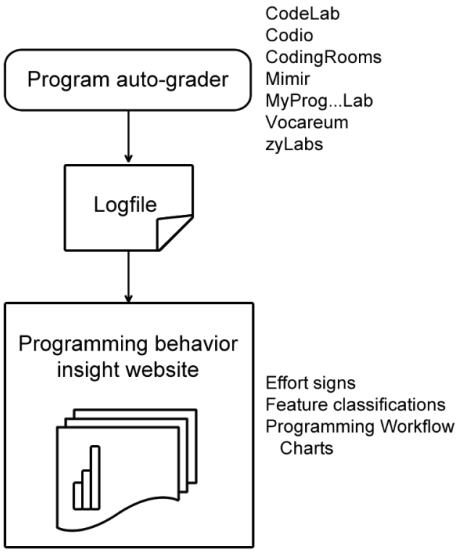

Figure 7.13: High level description of the process to get a log file from a program auto-grader and use the file to automatically generate workflow charts on a web page using our tool.

The current iteration of our website has sorting functionality so instructors can point out key features like students who spend the most amount of time working or students who complete the assignment with the least number of develops or submits. The website also has search functionality to find data for particular students quickly and supports a textual view for quick results and a visual view to see all workflow charts. In

the future, we may investigate integrations directly with the auto-grader so that no log file uploading is necessary. Figure 7.14 is a screenshot of the summary table at the top of the web page, Figure 7.15 is a screenshot of the textual view of the web page, and Figure 7.16 is a screenshot of the visual view of the web page. Figure 7.17 is a small subsection of all the programming workflow charts shown on the web page to understand the vast number of charts we generate and display. The tool is still in beta and only accessible to us and a few instructors at our university.

Programming Workflow Charts

Assignment 8

Assignment averages						
	Timespent (sec)	# Runs	Score (%)	# Develops	# Submits	# Pivots
Assignment Total [290 students]	**1h 21m 19s**	**79**	**93**	**58**	**22**	**2**
- Lab 1 [288 students]	6m 44s	9	99	6	3	0
- Lab 2 [287 students]	20m 17s	19	93	14	5	1
- Lab 3 [281 students]	27m 4s	25	91	19	6	1
- Lab 4 [281 students]	10m 50s	11	96	8	3	0
- Lab 5 [265 students]	19m 1s	18	93	12	5	0
- Lab 6 [11 students]	12m 13s	10	57	4	5	0

Figure 7.14: Screenshot of the current interactive programming workflow chart website: summary analysis table.

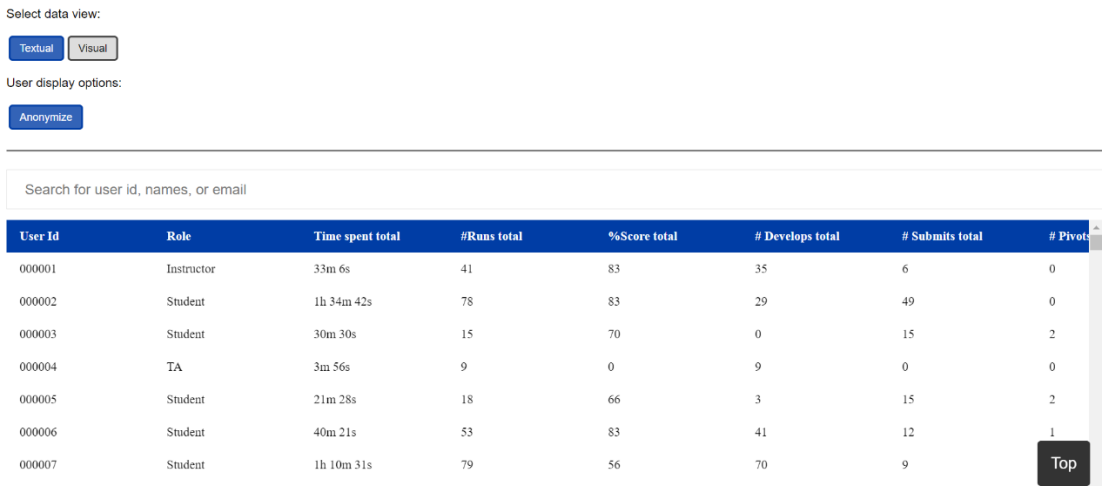

Figure 7.15: Screenshot of the current interactive programming workflow chart website: textual view.

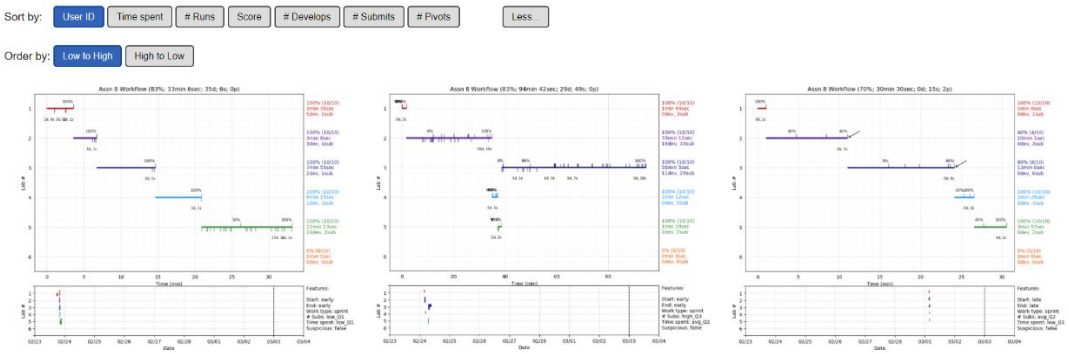

Figure 7.16: Screenshot of the current interactive programming workflow chart website: visual view.

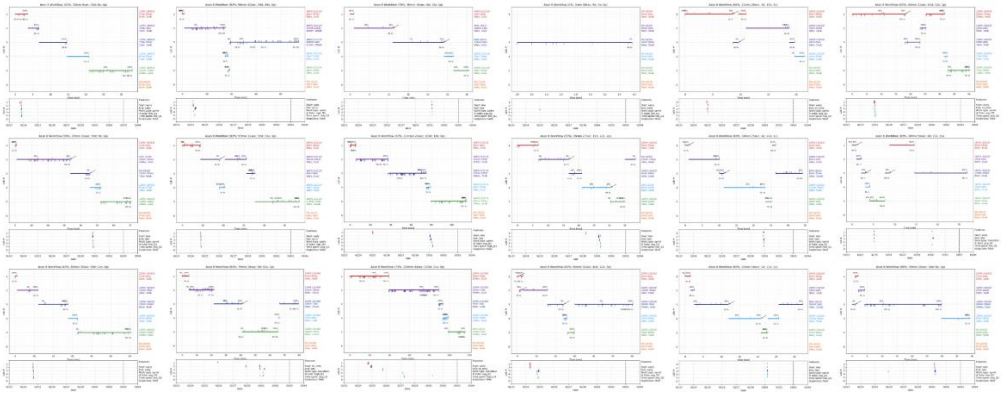

Figure 7.17: Screenshot of multiple programming workflow charts to see the vast number of charts we generate and display.

7.4.5 Future Improvements

These workflow charts have evolved since 2018, but we are still working on further improvements. First, we would like to add an indication of when students began struggling with lab activities compared to their peers. This would make comparison analysis easier when looking at a single week for an entire class. Second, we would like to further develop and use the classification system for these charts. As of now, we are using basic classifications, but if they could be more robust and accurate, we could create impactful intervention techniques for struggling students.

7.5 CONCLUSION

We described the evolution of a graphical representation, called "programming workflow charts", of student effort on weekly MSP assignments. We have used these charts in our teaching each quarter, to help provide insight into our class, get a quick feel for a particular student's effort when they come to office hours (for example), and even to help us decide to investigate potential cheating when a student's workflow chart shows almost no effort but high scores. Chapter 7 focuses on introducing the concept of such charts as a tool for instructors and showing the evolution of the design; that evolution may be of interest in itself, as more education-focused tools focus not necessarily on algorithms or traditional considerations but rather focus heavily on design considerations. We have found such charts quite useful in our teaching, but we encourage future work (and plan to conduct some ourselves) that demonstrate specific benefits, like detecting struggling students, or reducing cheating.

Chapter 8. Analyzing Pivoting Among Weekly Many Small Programs in a CS1 Course

8.1 Introduction

A unique benefit of the MSP approach is the ability for students to pivot, meaning to switch among lab activities if they get stuck. Chapter 8 investigates such pivoting and seeks to answer common questions related to pivoting. We analyze how many students pivot and the number of pivots done each week. Given a full-credit threshold (50 of 70 points on 7 MSP lab activities worth 10 points each with partial credit possible), we examine how students complete the subset of required points. We compare pivot data between a class with a full-credit threshold and a class without. We examine whether students who pivot eventually return to the program from which they pivoted, or if they leave the program unsolved. Finally, we analyze student workflow to observe various pivot patterns. By analyzing student pivot behavior, we hope the community can better understand the pros and cons of pivoting, to help decide whether to adopt an MSP approach and possibly a full-credit threshold.

8.2 Methodology

8.2.1 Course

This study was conducted at the University of California, Riverside, whose CS department typically ranks in the top 60 by U.S. News and World Report. The university operates on the quarter system. Each academic year is divided into three "regular" 10-week quarters (fall, winter, spring) and one compressed 5-week summer session.

Throughout the academic year, the CS1 course serves around 300-500 students each quarter. The course is required for all computing majors and for various engineering, science, and math majors, such that about half the students are computing majors and half are non-computing majors. The course topics include basic input/output, assignments, branches, loops, functions, and vectors. The weekly structure of the course includes three hours of instructor-led lecture, three hours of TA-led labs, interactive online readings, and auto-graded homework assignments. The course teaches C++ as the programming language. The course has a midterm during week six and a final after week 10. Each exam's points come half from multiple choice questions and half from free-response coding questions. The course uses active learning and peer learning in lectures.

8.2.2 DATA COLLECTION

We analyzed data from a Winter 2019 CS1 course section that was taught using an MSP approach. In total, 78 students were in the section used for this analysis. Our CS1 used an online textbook published by zyBooks for all class readings, activities, and programming assignments. At the end of the quarter, we collected all student develops and submits for every lab activity from the class zyBook. A develop is when a student runs their code through the zyBooks compiler for testing without grading and a submit is when the student "turns in" their assignment for grading. Note that all development was done in the built-in zyBooks coding windows; students were not introduced to an external development environment. Each develop has metadata on the lab title, a chapter section, a userID (anonymized and generated from zyBooks), and a timestamp. A submit has the same metadata as a develop, with additional metadata on the score the student earned on

the submit and the max score possible for the lab. In total, we collected data from 78 students for 65 MSP lab activities. We collected 34,316 develops and 14,774 submits for a total of 49,090.

8.3 MSP STUDENT PIVOTING

With the MSP approach, students are assigned multiple lab activities to complete each week. For this class section, students were assigned 7 lab activities, each being worth 10 points, for a total of 70 possible points in a given week. Students were told that they only needed to earn 50 points of 70 each week to earn full credit. We refer to this 50-point cutoff as the full-credit threshold. Since students are given a set of lab activities to complete, they have the unique ability to pivot, or switch among lab activities while working.

A pivot is when a student partially completes a lab activity (e.g., scores 8 of 10 points) and then chooses to work on a different lab activity. More specifically, an activity run (submit/develop) is defined as a pivot if the activity meets all 5 of the following criteria.

- The activity is not the student's first activity for the week
- The activity is for a different lab activity than the previous activity
- The activity is for a lab activity that has not been completed
- The previous activity is for a lab activity that has not been completed
- The activity and previous activity are for lab activities assigned in the same week

8.4 How Many Times Do Students Pivot Each Week?

Each week, students were assigned 7 lab activities to complete; each focusing on the topic taught during that week. For example, week 3 teaches while-loops, so students were given 7 lab activities that focused on loop creation, loop starting/ending conditions, etc. A key question is "How many times do students pivot each week?"

8.4.1 Analysis and Procedure

To answer this question, we first used the rules defined in Section 8.3 to count how many times each student pivoted during each week of the course. To best understand pivot behavior, for each week, we computed the average number of pivots, the minimum, the maximum, the value of the 1st and 3rd quartiles, and the standard deviation. Finally, we computed an average across all weeks to determine the average times students pivot each week. Only nine weeks were included in our calculations since week 10 has no MSP assignments. Students who did not attempt any of the lab activities for a given week were excluded from weekly calculations.

8.4.2 Results

Figure 8.1 is a box-and-whisker plot that summarizes the number of pivots students did each week. Above each whisker are the average number of pivots and the standard deviation. The average number of pivots across all weeks is shown in the top-right corner. The x-axis is the week number and the y-axis is the average number of pivots.

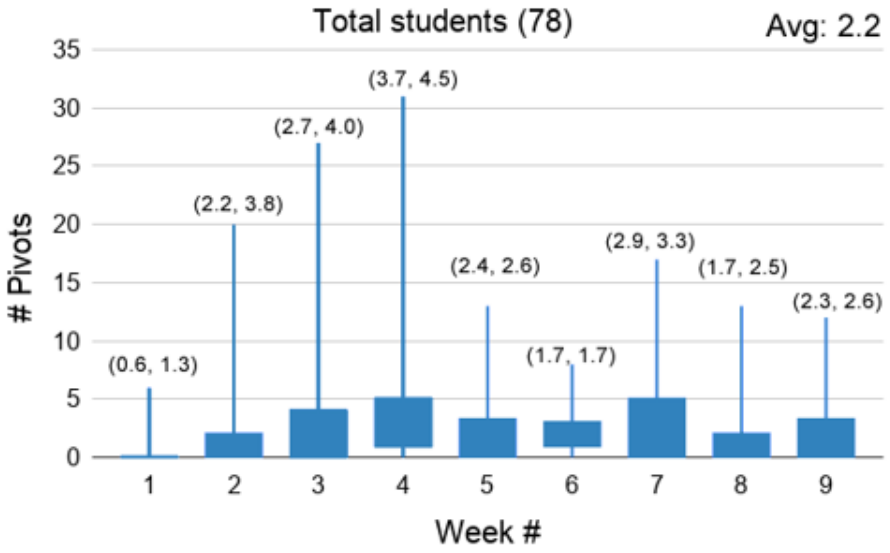

Figure 8.1: Box-and-whisker plot to show the pivots each week with a full-credit threshold. The average pivots and standard deviation appear above each whisker (avg, stdev). Total average pivots is 2.2 per week.

Figure 8.1 shows students pivoted an average of 2.2 times each week. Week 1 has a much lower pivot rate, due to the lab activities being easier. Week 4 has the most pivots (3.7 on average), likely due to students being taught while-loops for the first time, one of the most difficult concepts in the course. The standard deviation is larger in weeks 2-4 when students learn expressions, branches, and loops, and lower later when learning functions and vectors.

8.5 What Percent of Students Pivot Each Week?

Section 8.4 analyzed the average pivots each week. Additionally, we wanted to analyze the percentage of students that pivoted each week. A key question is "What percent of students pivot each week?"

8.5.1 Analysis and Procedure

To calculate the average percentage of students that pivoted each week, we first determined how many students pivoted at least once for each week in the quarter. If the

student pivoted at least once, they were counted as pivoting for that week. Next, we computed the percentage of students that pivoted each week. Finally, we averaged across all the weeks to calculate the average percentage of students that pivoted during the entire quarter. Students that did not attempt any lab activities for a given week were not included in the calculations for that week.

8.5.2 Results

Figure 8.2 summarizes the average percent of students that pivoted each week. The x-axis is the week number and the y-axis is the percent of students.

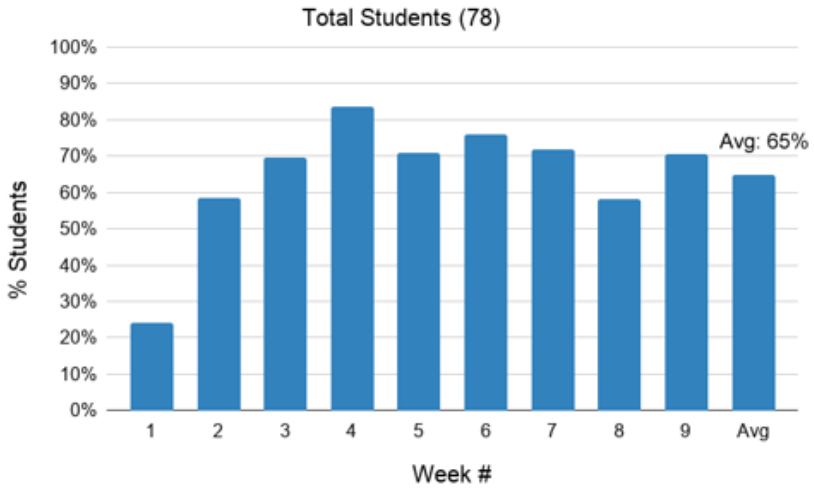

Figure 8.2: Percent of students that pivot each week.

Figure 2 shows that on average, 65% of students pivoted at least once each week. Week 1 has the lowest percent of students that pivoted, likely due to the lab activities for week 1 being quite easy -- students would complete each without getting stuck. Across the remainder of the term, the percent of students that pivoted seems to be consistent. Across the quarter, 95% of the students in the class (74) pivoted at least once on a program.

8.6 What Are Some Observed Pivot Patterns?

Recognizing the ways in which students pivot is important to understand how students utilize the ability to pivot among lab activities each week. To take a closer look at student pivot patterns, we use programming workflow charts to visibly see how students worked on weekly MSP assignments. In Section 8.6, we show multiple programming charts to visually represent how students worked on their MSP assignments during various weeks. A key question is "What are some observed pivot patterns?"

8.6.1 Analysis and Procedure

See Section 7.2.3 to see how each programming workflow chart is constructed.

8.6.2 Results

Section 8.6.2 presents three different workflow charts that demonstrate various student workflow patterns. Each chart summarizes the number of submits and develops, the total time spent, and the total score earned. Analyzing a combination of details about each activity also provides insight on student pivot patterns. Each line in the chart is color-coded to represent a different lab activity for the week. Since students were given 7 lab activities to complete each week, there are 7 different lines on each chart. The x-axis is the time spent in minutes and the y-axis is the lab activity number. Note, the time reported in all the charts is an understatement as previously discussed.

8.6.2.1 Student Pattern 1: 0 Pivots

Figure 8.3 shows a student workflow chart from week 5. During week 5, students were being taught for-loops. This student worked straight through all 7 lab activities, from LA1 (lab activity 1) to LA7. The student spent the least time on LA1, and the most

time working on LA4. On average, excluding LA1, the student spent about 13 minutes working on each lab activity. This student did not pivot while working on this MSP assignment.

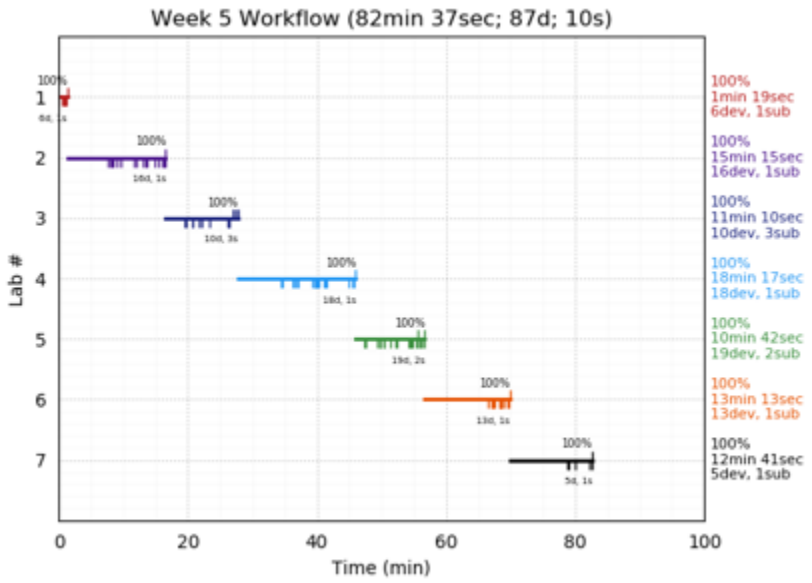

Figure 8.3: Programming workflow chart for a student during week 5.

8.6.2.2 Student Pattern 2: 3 Pivots

Figure 8.4 shows a student workflow chart from week 4. During week 4, students were being taught while-loops. This student began working on LA1 (lab activity 1) and scored 100%. They moved to LA2 and scored 100% after almost 2 hours of working. Next, the student scored 80% on LA3 and decided to pivot away to LA4. The student struggled with LA4, only being able to earn 40% of the points. The student then decided to pivot to LA5, but scored 0% the first time around. The student then pivoted and returned to LA4 and improved their score from 40% to 100%. The student then returned to LA5 and improved their score from 0% to 100% after around 25 minutes of working. They then completed LA6 and returned to work on LA3, improving their score from 80%

to 100%. The student finally attempted LA7, scoring 0 points, and stopped working entirely after that. In total, this student pivoted 3 times; indicated by the arrows in the chart.

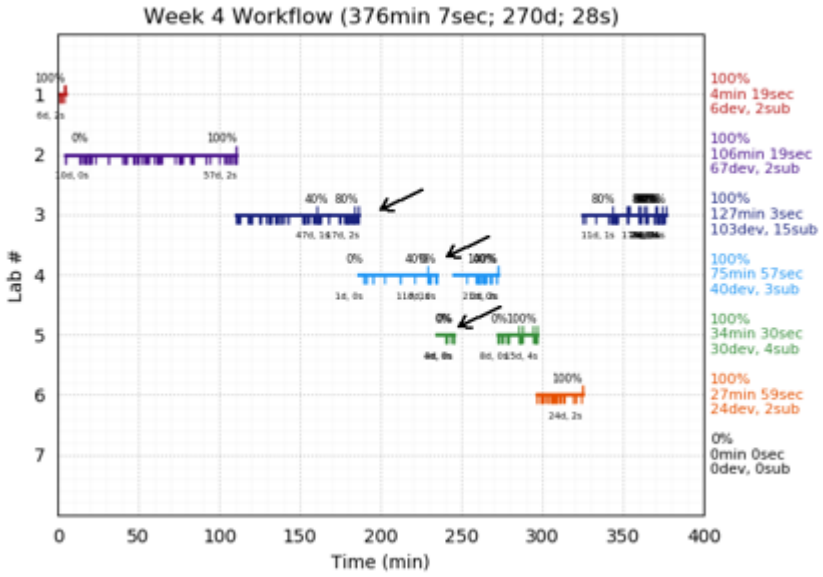

Figure 8.4: Programming workflow chart for a student during week 4.

8.6.2.3 STUDENT PATTERN 3: 10 PIVOTS

Figure 8.5 shows a student workflow chart from week 8. During week 8, students were being taught vectors. This student spent around 3 hours in total working on lab activities for the week. To summarize, this student spent most of their time working on LA1 (lab activity 1), LA2, and LA5. In total, this student pivoted 10 times on this week's MSP assignment. Although the student scored 0% on many of their first attempts, they were still able to pivot between each lab activity and score the needed points to earn 100% for the week.

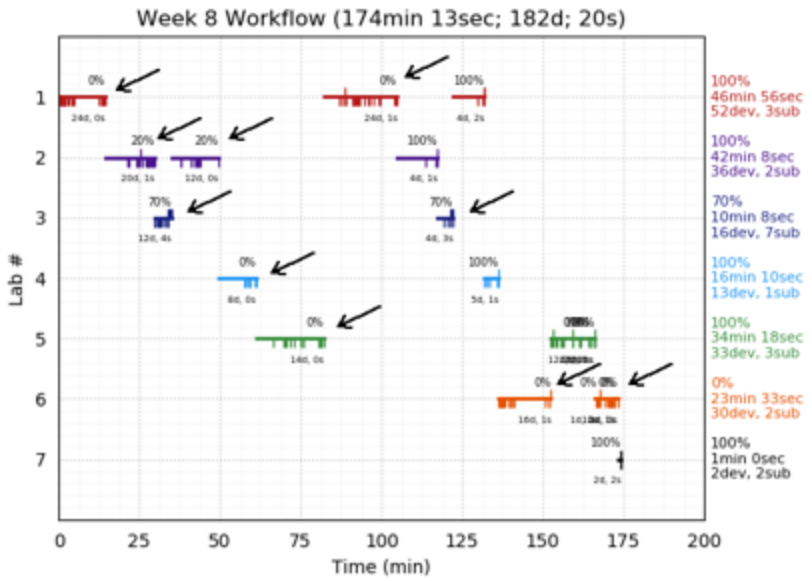

Figure 8.5: Programming workflow chart for a student during week 8.

These workflow charts provide insight into student behavior when working on weekly MSP assignments. Based on these charts, we see that some students do not need to pivot, and can work through all the material straight through, and yet we also see other students make heavy use of pivoting when working on their lab activities. Overall, we can see that a student may initially struggle, but given time and the ability to pivot, they can learn from other lab activities and help themselves improve when returning to previously attempted lab activities for the week.

8.7 Do Students Pivot More or Less Given a Full-credit Threshold?

The class section used in this analysis was given 7 lab activities to complete each week (each worth 10 points, 70 points for the week total), but they only needed to complete 70% of the points to get full credit for the week. We refer to the 70% cutoff as the full-credit threshold.

During Winter 2019, two other sections of CS1 were taught without using a full-credit threshold. The other two sections assigned students 7 lab activities each week, 5 required and 2 optional. Each required lab activity was worth 10 points (50 points total for the week), and the optional lab activities were worth 0 points (no extra credit). A key question is "Does having a full-credit threshold change pivot behavior?"

8.7.1 Analysis and Procedure

To answer this question, we ran similar analyses as presented in Section 8.4 and Section 8.5, but for the other two sections of CS1 offered during Winter 2019. We used these analyses to calculate the average number of pivots each week and the percent of students that pivot when there is not a full-credit threshold. For this analysis, we collected data from the other two class sections being taught. In total, we collected an additional 50,655 submits and 91,774 develops from 182 students over 47 MSP lab activities.

Since we collected data from another class section, there are some potential threats to validity. The other class sections did have a different instructor which could lead to instructor bias. Aside from this, all other class variables were kept the same - they used the same online textbook, used a subset of the same MSP lab activities, followed the same course pacing, and took the same exams.

8.7.2 Results

Figure 8.6 is a box-and-whisker plot that shows the number of pivots students did each week without a full-credit threshold. Above each whisker are the average number of pivots and the standard deviation. Students without a full-credit threshold pivot an average of 1.6 times each week compared to an average of 2.2 pivots each week by the

full-credit threshold students. The x-axis is the week number and the y-axis is the average number of pivots.

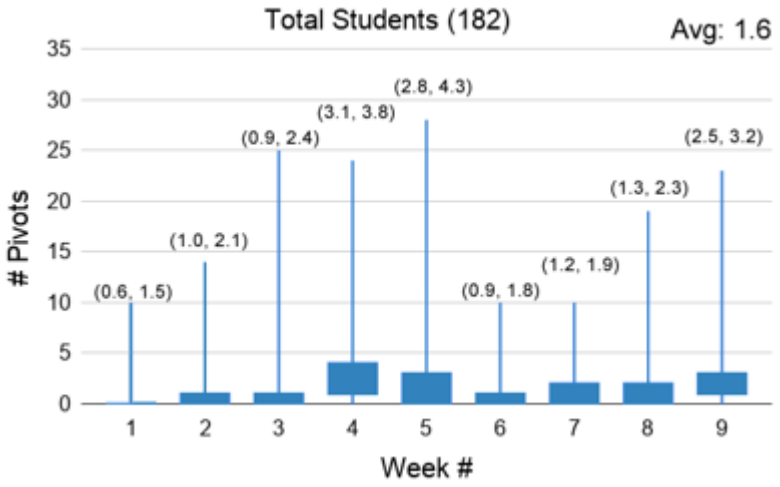

Figure 8.6: Box-and-whisker plot to show the pivots each week without a full-credit threshold. The average pivots and standard deviation appear above each whisker (avg, stdev). Total average pivots is 1.6 per week.

Figure 8.7 shows the average percent of students that pivoted each week in a class without a full-credit threshold. The x-axis is the week number and the y-axis is the percent of students. On average, 48% of students without a full-credit threshold pivot each week compared with an average of 65% by students with a full-credit threshold.

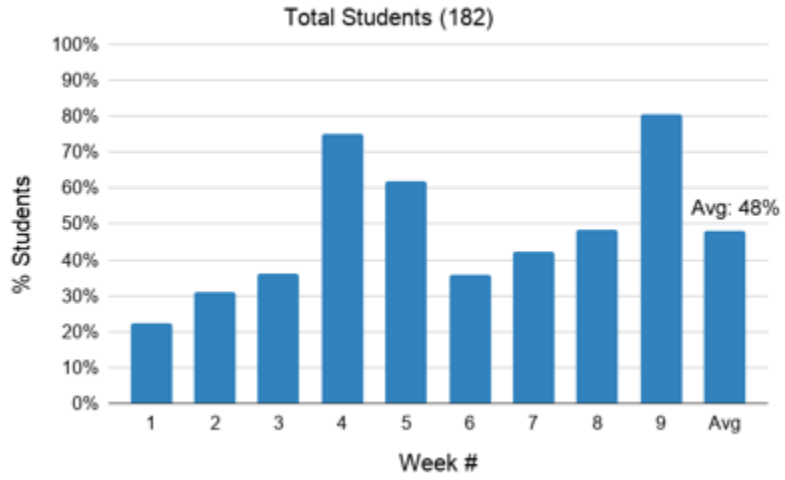

Figure 8.7: Average percent of students that pivot each week without full-credit threshold.

Although both groups had the option to complete 7 lab activities each week, Figure 8.2 compared with Figure 8.7 show that more students pivot when given a full-credit threshold (65% vs. 48%) while Figure 8.1 and Figure 8.6 show that students pivot more frequently when given a full-credit threshold (2.2 vs. 1.6). On first thought, this is likely due to the fact that without a full-credit threshold, students know they need to complete all required lab activities eventually, so they work through all lab activities until completion, but more analysis must be done to confirm. One question for further research is "Does giving students a full-credit threshold increase student agency?"

8.8 Do Students Return to Complete the Original Lab Activity They Pivot From?

One common critique we hear when sharing data on pivoting is that allowing students to pivot could encourage behavior such that students complete the "easy" points of each lab activity, pivot away, and skip the "difficult" parts, thus allowing students to not fully learn programming. To see if this concern is true, one key question is "Do students typically return to the original lab activity they switch from?"

8.8.1 Analysis and Procedure

To address this question, we first define some terminology to categorize student pivot behavior.

- Pivot none (N): student did not pivot from the lab activity
- Pivot away (P): student pivoted from the lab activity and did not return
- Pivot return (PR): student pivoted, returned to the lab activity, but made no improvement in score

- Pivot improve (PI): student pivoted, returned to the lab activity, and improved their previous score
- Pivot complete (PC): student pivoted, returned to the lab activity, and completed the lab activity fully (scored 100%)

We keep track of all activity runs for each lab activity, and then once all activity runs are completed, we can look at all activity runs as a whole to apply the mentioned categories. We isolate the students who pivoted, what they did after pivoting, and if they worked again once they returned. By doing so, we are able to better understand specific pivot behavior.

8.8.2 Results

In total, students completed 4,596 lab activities over the quarter. Of those total lab activities completed, students pivoted on 20% of them. Figure 8.8 is a pie chart that summarizes pivot categories for the subset of lab activities that were pivoted from.

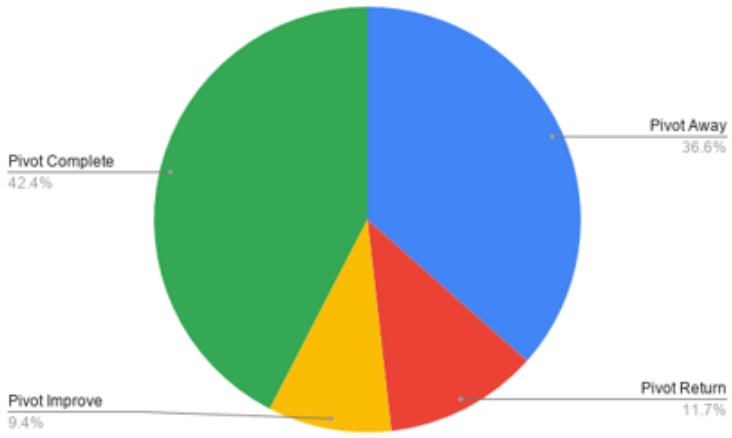

Figure 8.8: Pie chart summarizing student pivot categories.

To summarize, students returned to 11.7% of lab activities, returned and improved on 9.4% of lab activities, and returned to complete 42.4% of lab activities. Overall,

students pivoted away from 36.6% of lab activities and never returned. Figure 8.8 shows that students come back to work on 63.4% of lab activities and improve their score for 51.6% of lab activities. Since students return to a majority of attempted lab activities, this finding shows that students do use pivoting in helpful ways, not harmful to avoid the "hard" parts in a lab activity. Likely, students use pivoting when stuck to either get ahead on other problems, or they are able to self-enlighten themselves by learning from other lab activities and then return to work on the lab activities they were previously stuck on.

8.9 Student Feedback

We surveyed students during week 5 (midway through the quarter) to gather their thoughts on the ability to pivot between lab activities. Using a 4-point Lickert scale (4 is "Strongly agree", 3 is "Slightly agree, 2 is "Slightly disagree, and 1 is "Strongly disagree") we asked students "I find the ability to jump between programming assignments helpful." The average response was 3.23, indicating that students on average were between "Slightly agree" and "Strongly agree."

8.10 Threats to Validity

8.10.1 Different Instructors

In Section 8.7, we look at the other class sections of CS1 being taught without using a full-credit threshold to compare pivoting behavior between students. Since we collected data from other class sections, there could be an instructor bias as the instructor who taught the full-credit threshold group was different from the instructor who taught the other sections. Although there could be a threat to validity, we note that both instructors are very similar in personality, teaching style, previous evaluations, etc. Both

have been teaching for many years together and typically have weekly meetings to share and ensure the class is being run in virtually the same way. Furthermore, all other class variables were kept the same - all sections used the same online textbook, the same lab activities, followed the same course pacing, and took the same exams.

8.10.2 Different Style of an MSP Approach

In Section 8.7, we look at the other sections of CS1 being taught during Fall 2019. The other class sections used a slightly different style of an MSP approach, such that instead of assigning students 7 lab activities and allowing them to choose which ones to complete for their weekly points, the other class section assigned 7 lab activities with 5 being required, offering the other 2 for additional practice (but no extra credit). Although these two methods are slightly different, we do not believe this to have much impact on the results of our experiment.

8.10.3 Outside Code Development

Although our students were only introduced to the zyBooks in-book IDE, this doesn't mean that students could not use their own IDEs to develop their code outside of zyBooks. If this were the case, we would be missing some important data on activity runs as we would have no way to track their develops. Knowing this, it is possible that some of our analysis numbers are slightly off. However, since this is an introductory CS1 course, and most students who take this class are new to programming, it's likely that most students did use the zyBooks' in-book IDE primarily to code. Even if this wasn't true, our numbers would be an understatement and we would expect most of our numbers to increase (i.e. more activities could lead to more pivots, time spent, etc.)

8.11 CONCLUSION

One way we have tried to improve our CS1 course is the use of an MSP approach. A unique benefit of using an MSP approach is that students can pivot, meaning to switch among lab activities when stuck. Since all MSP assignments relate to a core topic each week, pivoting is a unique benefit that allows students to gain insight from other lab activities and then apply that knowledge to solving the previous problems they were stuck on. This paper addressed many common questions about pivoting. We found that students on average pivot 2.2 times each week with most students (65%) making use of pivoting when working on lab activities each week. We explored various pivoting patterns and saw that students can use pivots to solve problems they previously could not. We showed that students, given a full-credit threshold, do pivot more than students not given a full-credit threshold. Finally, we showed that when a student pivots away from a lab activity, they usually return to work on the lab activity again. There is still much more analysis to be done on the ability to pivot using an MSP approach, but this work has shown that students are making good use of the benefits that an MSP approach and pivoting have to offer.

Chapter 9. CONTRIBUTIONS

We introduced a many small programs (MSP) teaching approach for the first time in Spring 2017 with a goal to improve the students' experience in our CS1 without harming grade performance. Four years later, our university only teaches CS1 via an MSP approach and we've gotten positive feedback from instructors, teaching assistants, and most importantly the students.

Over the past 4 years, we have implemented, studied, and improved the way we use an MSP approach in our CS1 classes. Compared to an OLP approach, we have seen that an MSP approach can be used to increase student satisfaction in CS1 and reduce student stress without worsening their grade performance in class. In fact, we have seen that an MSP approach can also yield overall better grade performance, mainly on coding assessments like the midterm and final exams. Furthermore, we have seen students make good use of weekly MSP assignments in regards to their learning. When using an MSP approach, students spend a healthy amount of time working on programming each week, students begin working on their weekly assignments earlier in the week, students purposefully complete additional MSP lab activities when given a full-credit threshold even though no extra credit is given, students use MSP lab activities to prepare for exams, and most importantly MSP-trained students still perform well in a CS2 that assigns large programming assignments under an OLP approach. Through our research, we have seen that an MSP approach can be used across universities and across programming languages with equal success. Finally, we have experimented and can

conclude that an MSP approach can yield positive results when used in conjunction with a Coral-first approach in a CS1 class.

Since implementing our MSP approach, we have designed, created, and maintained over 100 unique C++ MSP programming assignments. With the increased usage of an MSP approach at our university and the growing usage of an MSP approach at other universities across the nation (over 250 that we know about as reported by zyBooks in 2020), we built tools to help analyze and understand how students interact with MSP content. One of the most helpful tools we created is used to generate programming workflow charts to gain a visual understanding of how students work through weekly MSP assignments. Given a log file provided by zyBooks (or any other program auto-grader), we can quickly and automatically generate thousands of workflow charts -- one for every student for every weekly MSP assignment. These tools are readily available to us, but we hope to make them available to all instructors to use to better understand their students and their classes. As such, we are expanding the workflow chart generation tool to create an interactive web page for instructors to upload their own log files to generate and display their classes' programming workflow charts. The web page has search functionality, sorting functionality, and comes with two data views: a textual view that quickly summarizes the data in a table and a visual view to see all workflow charts. This tool is currently in beta and is being worked on by other students. We hope that a stable version will be completed by the end of this year. Currently, the tool can be used by instructors at our university and some others by invitation, but we plan to make it broadly available to all instructors soon.

Our initial goal of improving student experience in CS1 has been made possible due to this MSP approach. Instructors, teaching assistants, and students have all provided good feedback on using this approach in their classrooms. As of 2020, we are aware of hundreds of CS1 classes that have adopted an MSP approach to weekly programming assignments. Since 2017, our university has switched all CS1 classes to an MSP approach and there is no sign of going back to a traditional OLP approach.

On a final note, we compare our CS1 DFW rates when teaching via an OLP approach and via an MSP approach. We collected CS1 DFW data from Fall 2007 - Fall 2019. Comparing DFW rates between a CS1 OLP approach (Fall 2007 - Spring 2017) to a CS1 MSP approach (Spring 2017 - Fall 2019) we see a statistically significant decrease from 15% to 8.4% ($p<0.001$) when using an MSP approach. As Fall 2007 was over 10 years ago and our CS1 has generally improved over the years, we do a similar comparison only including CS1 OLP approach data from Fall 2014 - Spring 2017 (same number of CS1 MSP approach quarters analyzed). With a more recent comparison, we still see a reduced DFW rate from 9.6% to 8.4% ($p=0.35$), although not significant. Based on the success we have had in our own CS1 class and based on all the benefits we have seen an MSP approach offer, we encourage CS1 instructors to try incorporating an MSP approach in their CS1 as well.

References

[1] Ahoniemi, T., E. Lahtinen, and T. Reinikainen, "Improving pedagogical feedback and objective grading," SIGCSE Technical Symposium on Computer Science Edu. pp. 72-76, 2008.

[2] Alfaro, L.de and M. Shavlovsky. 2014. CrowdGrader: a tool for crowdsourcing the evaluation of homework assignments. In Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14). ACM, New York, NY, USA, 415-420. DOI: https://doi.org/10.1145/2538862.2538900

[3] Allen, J.M. and F. Vahid. An Analysis of Using Coral Many Small Programs in CS1, Journal of Computing Sciences in Colleges, 2021.

[4] Allen, J.M., F. Vahid, K. Downey, and A. Edgcomb. 2018. Weekly Programs in a CS1 Class: Experiences with Auto-graded Many-small Programs (MSP). In Proceedings of 2018 ASEE Annual Conference & Exposition. DOI: https://peer.asee.org/31231

[5] Allen, J.M., F. Vahid, A. Edgcomb, K. Downey, and K. Miller. 2019. An Analysis of Using Many Small Programs in CS1. In Proceedings of the 50th ACM technical symposium on Computer science education (SIGCSE '19). ACM, New York, NY, USA, 415-420. DOI: https://doi.org/10.1145/3287324.3287466.

[6] Autolab. http://www.autolabproject.com/. Accessed: July, 2018

[7] Beaubouef, T. and J. Mason. Why the high attrition rate for computer science students: some thoughts and observations. SIGCSE Bull. 37, 2 (June 2005), 103-106, 2005. DOI: http://dx.doi.org/10.1145/1083431.1083474

[8] Bennedsen, J. and M. E. Casperson, "Failure rates in introductory programming: 12 years later," in ACM Inroads, 2019.

[9] Bergin, S. and R. G. Reilly, "The Influence of Motivation and Comfort-Level on Learning to Program," in PPIG, 2005.

[10] Bishop, J.L. and M.A. Verleger. 2013. The Flipped Classroom: A Survey of the Research. In Proceedings of ASEE Annual Conference, 2013. Atlanta, Georgia. https://peer.asee.org/22585

[11] Blaheta, D., "Reinventing homework as cooperative, formative assessment," ACM Technical Symposium on Computer Science Education, pp. 301-306, 2014.

[12] BlueJ. https://bluej.org/. Accessed: August, 2020.

[13] Cliburn, D.C. and S. Miller. Games, stories, or something more traditional: the types of assignments college students prefer. In Proceedings of the 39th SIGCSE technical symposium on Computer science education (SIGCSE '08). ACM, New York, NY, USA, 138-142, 2008. DOI: https://doi.org/10.1145/1352135.1352184

[14] CloudCoder. https://www.cloud-coder.com/. August 2017.

[15] CodingBat. http://codingbat.com/about.html. August 2017.

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

[17] Denny, P. "Generating Practice Questions as a Preparation Strategy for Introductory Programming Exams". In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). ACM, New York, NY, USA, 278-283, 2015. DOI: https://doi.org/10.1145/2676723.2677253

[18] Denny, P., A. Luxton-Reilly, E. Tempero, J. Hendrickx, "CodeWrite: supporting student- driven practice of java," ACM Technical Symposium on Comp. Science Edu., pp. 471-476, 2011.

[19] Deterding S., D. Dixon, R. Khaled, and L. Nacke. From game design elements to gamefulness: defining "gamification". In Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments (MindTrek '11). ACM, New York, NY, USA, 9-15, 2011. DOI: https://doi.org/10.1145/2181037.2181040

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

[21] Edgcomb, A., F. Vahid, R. Lysecky, A. Knoesen, R. Amirtharajah, and M.L. Dorf. 2014. Student Performance Improvement using Interactive Textbooks: A Three-University Cross-Semester Analysis. In Proceedings of ASEE Annual Conference, 2015. DOI: https://doi.org/10.18260/p.24760

[22] Enfield, J. 2013. Looking at the Impact of the Flipped Classroom Model of Instruction on Undergraduate Multimedia Students at CSUN. TechTrends. 57, 6 (November/December 2013), 14-27. DOI: https://doi.org/10.1007/s11528-013-0698-1

[23] Falkner, N., R. Vivian, D. Piper, K. Falkner, "Increasing the effectiveness of automated assessment by increasing marking granularity and feedback units," ACM Technical Symposium on Computer Science Education, pp. 9-14, 2014.

[24] Findlay, S. and P. Mombourquette. Evaluation of a flipped classroom in an undergraduate business course. Business Education & Accreditation. 6. 63-71, 2014.

[25] Findlay-Thompson, S. and P. Mombourquette, "Evaluation of a Flipped Classroom in an Undergraduate Business Course," Business Education & Accreditation, v. 6 (1) p. 63-71, 2014.

[26] Gannt Chart. https://www.projectmanager.com/gantt-chart. Accessed: August, 2020.

[27] Gaughan, J.E. 2014. The Flipped Classroom in World History. The History Teacher. 47, 2 (February 2014), 221-244.

[28] Giannakos, M.N., J. Krogstie, and N. Chrisochoides. 2014. Reviewing the flipped classroom research: reflections for computer science education. In Proceedings of the Computer Science Education Research Conference (CSERC '14), Erik Barendsen and Valentina Dagiené (Eds.). ACM, New York, NY, USA, 23-29. DOI: http://dx.doi.org/10.1145/2691352.2691354

[29] Gilboy, M.B., S. Heinerichs, G. Pazzaglia, "Student Engagement Using the Flipped Classroom," Journal of Nutrition Education and Behavior, 47(1), 109–114, 2014.

[30] Gradescope. https://www.gradescope.com/. Accessed: August, 2020.

[31] Guzdial, M.,. 2003. A media computation course for non-majors. In Proceedings of the 8th annual conference on Innovation and technology in computer science education (ITiCSE '03), David Finkel (Ed.). ACM, New York, NY, USA, 104-108. DOI: http://dx.doi.org/10.1145/961511.961542

[32] Hendrix, D., L. Myneni, H. Narayanan, M. Ross, "Implementing studio-based learning in CS2," ACM technical symposium on Computer science education, 2010.

[33] Hundhausen, C., A. Agrawal, D. Fairbrother, M. Trevisan, "Does studio-based instruction work in CS 1?: an empirical comparison with a traditional approach," SIGCSE ACM technical symposium on Computer science education, pp. 500-504, 2010.

[34] Kinnunen, P. and L. Malmi, "Why students drop out CS1 course?". In Proceedings of the second international workshop on Computing education research (ICER '06). ACM, New York, NY, USA, 97-108, 2006. DOI: http://dx.doi.org/10.1145/1151588.1151604

[35] Kumar, A.N., 2004. Using Online Tutors for Learning - What do Students Think? In Proceedings of the 34th Annual Frontiers in Education (FIE '04). DOI: https://doi.org/10.1109/fie.2004.1408540

[36] Layman, L., L. Williams, and K. Slaten. 2007. Note to self: make assignments meaningful. In Proceedings of the 38th SIGCSE technical symposium on Computer science education (SIGCSE '07). ACM, New York, NY, USA, 459-463. DOI: https://doi.org/10.1145/1227310.1227466

[37] Lee, C.B., S. Garcia, L. Porter, "Can peer instruction be effective in upper-division computer science courses?," ACM Transactions on Computing Education (TOCE) - Special Issue on Alternatives to Lecture in the Computer Science Classroom, Vol. 13 Issue 3, Art. No. 12, 2013.

[38] Leutenegger, S. and J. Edgington. A games first approach to teaching introductory programming. In Proceedings of the 38th SIGCSE technical symposium on Computer science education (SIGCSE '07). ACM, New York, NY, USA, 115-118, 2007. DOI: https://doi.org/10.1145/1227310.1227352

[39] Matlab Grader. https://grader.mathworks.com/. Accessed: January, 2019.

[40] Measuring U. https://measuringu.com/pcalcz/. Accessed: November 2018.

[41] Mimir. https://www.mimirhq.com/. Accessed: August, 2018.

[42] Mok, H.N., "Teaching Tip: The Flipped Classroom," Journal of Information Systems Education, 25(1), 7-11, 2014.

[43] MOSS. https://theory.stanford.edu/~aiken/moss/. Accessed: August 2017.

[44] Nagappan, N., Laurie Williams, Miriam Ferzli, Eric Wiebe, Kai Yang, Carol Miller, and Suzanne Balik. Improving the CS1 experience with pair programming. In Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03). ACM, New York, NY, USA, 359-362. 2003. DOI: https://doi.org/10.1145/611892.612006

[45] Norman, V.T. and J.C. Adams. 2015. Improving Non-CS Major Performance in CS1. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15). ACM, New York, NY, USA, 558-562. DOI: https://doi.org/10.1145/2676723.2677214

[46] Pearson's MyProgrammmingLab https://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/. Accessed: August, 2020.

[47] Petersen, A., M. Craig, J. Campbell, and A.Tafliovich, " Revisiting why students drop CS1". In Proceedings of the 16th Koli Calling International Conference on Computing Education Research (Koli Calling '16). ACM, New York, NY, USA, 71-80, 2016. DOI: https://doi.org/10.1145/2999541.2999552

[48] Porter, L. and B. Simon. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13). ACM, New York, NY, USA, 165-170, 2013. DOI: http://dx.doi.org/10.1145/2445196.2445248

[49] Porter, L., C.B. Lee, and B.Simon. Halving fail rates using peer instruction: a study of four computer science courses. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13). ACM, New York, NY, USA, 177-182, 2013. DOI: http://dx.doi.org/10.1145/2445196.2445250

[50] Porter, L., C.B. Lee, B. Simon, and D. Zingaro. Peer instruction: do students really learn from peer discussion in computing?. In Proceedings of the seventh international workshop on Computing education research (ICER '11). ACM, New York, NY, USA, 45-52, 2011. DOI: http://dx.doi.org/10.1145/2016911.2016923

[51] Porter, L., D. Bouvier, Q. Cutts, S. Grissom, C. Lee, R. McCartney, D. Zingaro, B. Simon, "A Multi-institutional Study of Peer Instruction in Introductory Computing," SIGCSE ACM Technical Symposium on Computing Science Education, pp. 358-363, 2016.

[52] Porter, L., M. Guzdial, C. McDowell, and B. Simon. Success in introductory programming: what works?. Commun. ACM 56, 8 (August 2013), 34-36, 2013. DOI: https://doi.org/10.1145/2492007.2492020

[53] Porter, L., S. Garcia, J. Glick, A. Matusiewicz, C. Taylor, "Peer Instruction in Computer Science at Small Liberal Arts Colleges," ITiCSE ACM conference on Innovation and technology in computer science education, 129-134, 2013.

[54] Problets. http://problets.org/. August 2017

[55] Rodríguez, F.J., K.M. Price, K.E. Boyer, "Exploring the Pair Programming Process: Characteristics of Effective Collaboration," ACM SIGCSE Technical Symposium on Computer Science Education, pp. 507-512, 2017.

[56] Roehl, A., "The Flipped Classroom: An Opportunity To Engage Millennial Students Through Active Learning Strategies," Journal of Family and Consumer Sciences, 2013.

[57] Runestone. https://runestone.academy/runestone/default/user/login?_next=/runestone/default/index. Accessed: August, 2020.

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

[59] Simon, B., J. Parris, J. Spacco, "How We Teach Impacts Student Learning: Peer Instruction vs. Lecture in CS0," SIGCSE technical symposium on Computer science edu., pp. 41-46, 2013.

[60] Simon, B., M. Kohanfars, J. Lee, K. Tamayo, and Q. Cutts. Experience report: peer instruction in introductory computing. In Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10). ACM, New York, NY, USA, 341-345, 2010. DOI: http://dx.doi.org/10.1145/1734263.173438

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

[62] Soh, L. Using game days to teach a multiagent system class. In Proceedings of the 35th SIGCSE technical symposium on Computer science education (SIGCSE '04). ACM, New York, NY, USA, 219-223, 2004. DOI: https://doi.org/10.1145/971300.971378

[63] Stone, J.A. and E.M. Madigan. 2008. The impact of providing project choices in CS1. SIGCSE Bull. 40, 2 (June 2008), 65-68. DOI: https://doi.org/10.1145/1383602.1383637

[64] Submitty. http://submitty.org/. Accessed: July, 2018.

[65] Turing's Craft: CodeLab. https://www.turingscraft.com/. Accessed: July, 2018.

[66] Vocareum. https://www.vocareum.com/. Accessed: August, 2020.

[67] Watson, C. and F.W.B. Li, "Failure rates in introductory programming revisited". In Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE '14). ACM, New York, NY, USA, 39-44, 2014. DOI: http://dx.doi.org/10.1145/2591708.2591749

[68] Web-CAT. http://web-cat.org/. Accessed: July, 2018.

[69] Wilcox, C., "Testing Strategies for the Automated Grading of Student Programs," ACM Technical Symposium on Computing Science Education, pp. 437-442, 2016.

[70] Williams, L., K. Yang, E. Wiebe, M. Ferzli, and C. Miller. Pair Programming in an Introductory Computer Science. OOPSLA Educator's Symposium, Seattle, WA, 2002. DOI: https://doi.org/10.1076/csed.12.3.197.8618

[71] Zingaro, D. Peer instruction contributes to self-efficacy in CS1. In Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14). ACM, New York, NY, USA, 373-378, 2014. DOI: http://dx.doi.org/10.1145/2538862.2538878

[72] zyBooks. https://www.zybooks.com/catalog/zylabs-programming/. Accessed: August, 2018.