Many Small Programs in CS1: Usage Analysis from Multiple Universities

Joe Michael Allen, University of California, Riverside

Joe Michael Allen is a Ph.D. student in Computer Science at the University of California, Riverside. His research interests include STEM education, specifically educational games for building skills for college-level computer science and mathematics.

Prof. Frank Vahid, University of California, Riverside

Frank Vahid is a Professor of Computer Science and Engineering at the Univ. of California, Riverside. His research interests include embedded systems design, and engineering education. He is a co-founder of zyBooks.com.

Mrs. Kelly Downey

I have a bachelors and masters degree in electrical engineering. After working in industry, I found a passion for education. I am currently a lecturer at UC, Riverside for the computer science department.

Mr. Kris Miller
Dr. Alex Daniel Edgcomb, Zybooks

Alex Edgcomb is Sr. Software Engineer at zyBooks.com, a startup spun-off from UC Riverside that develops interactive, web-native learning materials for STEM courses. Alex is also a research specialist at UC Riverside, studying the efficacy of web-native content and digital education.

Many Small Programs in CS1: Usage Analysis from Multiple Universities

Abstract

In 2017, we introduced a teaching method called many small programs (MSPs) in the CS1 courses at our university. Instead of teaching via one large programming assignment (OLP) each week, MSPs allow the instructor to assign multiple programming assignments, for example 5 or more, each week instead. Our previous studies have shown that MSPs can improve the student experience by reducing stress and increasing student satisfaction in the course. Furthermore, MSPs have been shown to improve student grade performance in CS1, especially on the coding portion of exams. In a follow-up study, we gained insight on how students were using MSPs, and learned that students use MSPs in ways beneficial to their learning. Students spend sufficient time working on MSPs each week, start working on MSPs earlier, complete more MSPs than required (given a weekly full-credit threshold), take advantage of pivoting (switch to another program if stuck on the current one), and use MSPs more to study for exams. We have shared these findings with universities around the nation; causing other universities to switch from teaching CS1 with OLPs to MSPs. Given data on student MSP submissions from other schools, we extend our work to include MSPs taught at other universities. We perform similar analysis and found that students being taught via MSPs from other universities also use MSPs in beneficial ways. Students spend sufficient time working on MSPs each week, they start working on MSPs early, and they complete a majority of assigned MSPs each week.

1. Introduction

Student success in introductory programming courses (known as CS1) is critical to keeping students in computer science (CS), training students in other majors who need some programming, 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 [6, 8]. Watson and Li [11] report that over the past 30 years, CS1 classes have a 30% non-passing rate. Beaubouef and Mason [4] state that drop rates between 30%-40% is now the norm for many CS programs. These issues have drawn the attention of education researchers to find ways to improve CS1.

1.1 Autograded programs

One improvement approach makes use of modern 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 [10], Web-CAT [12], and numerous homegrown auto-graders at various universities [3, 9]. 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 [13], Mimir [7], and Matlab Grader [6] 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 200 courses (mostly CS1) have started using an auto-grader that did not before.

1.2 One Large Program (OLP)

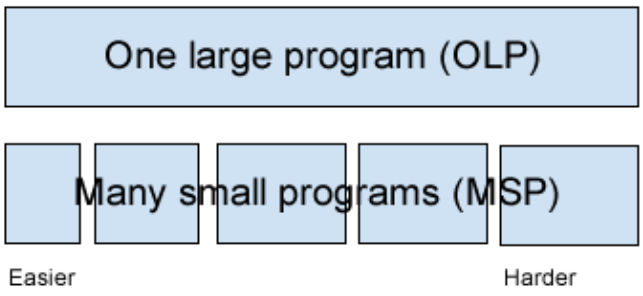
Traditionally, CS1 courses are taught by giving students one large programming assignment (OLP) to complete each week. These assignments tend to be large in size (require a solution between 50 - 100 lines of code), cover multiple new concepts at once, may require students to complete several interconnected parts, and tend to have lots of text to explain the problem. Since CS1 is an introductory course and is taken by students who have never programmed before, such large programs can be intimidating; leading to increased stress and procrastination.

At our university, and likely at most others, OLPs given to students in CS1 often form much of the class grade. Additionally, some instructors may choose to combine weekly OLPs with smaller coding problems as warm-up assignments or practice homework problems. Either way, programming assignments in CS1 often account for the most challenging and stressful part of the course -- and is also where much of the learning occurs. For this reason, we chose to focus our research on improving how programming assignments are used in CS1.

1.3 Many Small Programs (MSPs)

The ease of using modern-day program auto-graders to create and grade programming assignments combined with the benefit of giving students immediate, detailed score feedback encouraged us to consider a new teaching approach called many small programs (MSPs). Opposed to assigning students one large program each week, we now assign students many smaller programs each week instead. Figure 1 shows a high-level comparison between an OLP vs. MSP approach. Notice how there are multiple MSPs ranging from “easier” to “harder.” Previously without auto-graders, the extensive resources required to grade so many programs may have deterred instructors from considering an MSP approach. Or, perhaps the difficulty of using older auto-graders to create and maintain such programs may have deterred this approach as well.

Fig. 1. One large program (OLP) vs. many small programs (MSPs) per week.



Compared with OLPs, MSPs focus on teaching one specific concept at a time, are built using minimal text, are short in size, and they tend to have a small solution size, 20-50 lines of code. Additionally, MSPs have other unique benefits to help students. First, due to their small size, students may find them less intimidating; causing students to be more confident and start working earlier. Next, since students are given multiple programming assignments to work on, if they get stuck on one, they have the option to "pivot" or move on to another MSP and then come back later to finish the current one. Finally, students get more practice on new concepts being taught due to having more problems to complete each week.

1.4 Research Objectives

In this work, we try to further understand how students use MSPs. Previously, we looked at MSP usage for CS1 courses at our university. We found positive results; that students were using MSPs in ways that were beneficial to their learning. We now extend that work to include CS1 courses taught at other universities and with different programming languages to see if those results are consistent with what we previously concluded.

Section 2 discusses our previous work. Section 3 describes our methodology, including details about the CS1 courses and universities we included in this study. Section 4 presents our usage analysis of MSPs from other universities. Section 5 provides a general discussion on this work. Section 6 concludes.

2. Previous Work

2.1 Student Satisfaction and Performance

Prior to this work, we conducted two studies on MSPs; both yielding positive results. In our first study [1], we found that teaching via MSPs in CS1 can improve student satisfaction and grade performance. To gauge student satisfaction, we gave students a "stress survey" at the end of the quarter. 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"). The OLP group consisted of 2 class sections; totaling 171 students who were taught only via OLPs and the MSP group consisted of 1 class section, totaling 76 students who were taught only via MSPs. We computed p-values for all questions using a one-tailed t-test. Table 1 summarizes the results of the "stress survey" and shows that students who were taught via MSPs responded more favorably than students who were taught via OLPs in almost every question asked.

Table 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 MSP group.

Question	OLP group average	MSP group average	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

Note that for the question "The weekly programming assignments were enjoyable," there was a significantly more positive response from the students who had MSPs vs. the students who had OLPs. Furthermore, students who were taught via MSPs performed better that students who were taught via OLPs on the exams given in course, especially on the coding portion of the exams. Table 2 summarizes a grade analysis done at the end of the course term. Students who had MSPs performed higher in almost all categories, especially on the coding portions of exams.

Table 2: OLP vs. MSP 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 ($p < 0.0056$).

	OLP group %	MSP 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 activities	88.4%	87.1%	0.317

2.2 Usage Analysis

In our follow-up study [2], we performed a usage analysis only considering CS1 taught at our university. We found that students use MSPs in ways beneficial to their learning. In summary, our analysis concluded that students spend a good amount of time, about 2 hours a week, working on MSPs. Given a week to complete MSPs, students begin working about 2.5 days ahead of the due date with 37% of students starting 3 days ahead. Given a scoring threshold, we found that students willingly complete more MSPs than they were required each week for additional practice and learning. Over half, 54%, of students use MSPs to practice for exams. Many students also take advantage of additional benefits of MSPs like pivoting or switching between MSPs before completing them. Finally, we looked at CS2 performance and found that MSPs do not harm students when they reach a CS2 where they are required to complete OLPs.

3. Methodology

This section presents details on how the multi-university follow-up study in this paper was conducted. We discuss the universities included in this analysis and details about the program auto-grader that was used for MSPs.

3.1 CS1 University Metadata

We looked at CS1 courses taught using MSPs 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 3 provides details about the universities included in this study.

Table 3: Metadata on the 10 universities included in this study. Details include the programming language being taught, number of students in the class, number of MSPs assigned, number of submissions collected, and number of develops collected.

	Prog Language	#Students	# MSPs	# Submissions collected	# Develops collected
University 1	C++	20	98	3177	5635
University 2	Python	81	69	19244	19707
University 3	C++	30	19	2397	3416
University 4	C++	14	61	1675	5104
University 5	Java	11	51	643	3535
University 6	C++	234	77	21451	40573
University 7	Python	333	43	88981	103089
University 8	C++	79	25	7315	9298
University 9	Java	56	59	7454	18505
University 10	Java	321	65	40320	96721

3.2 zyBooks Program Auto-grader

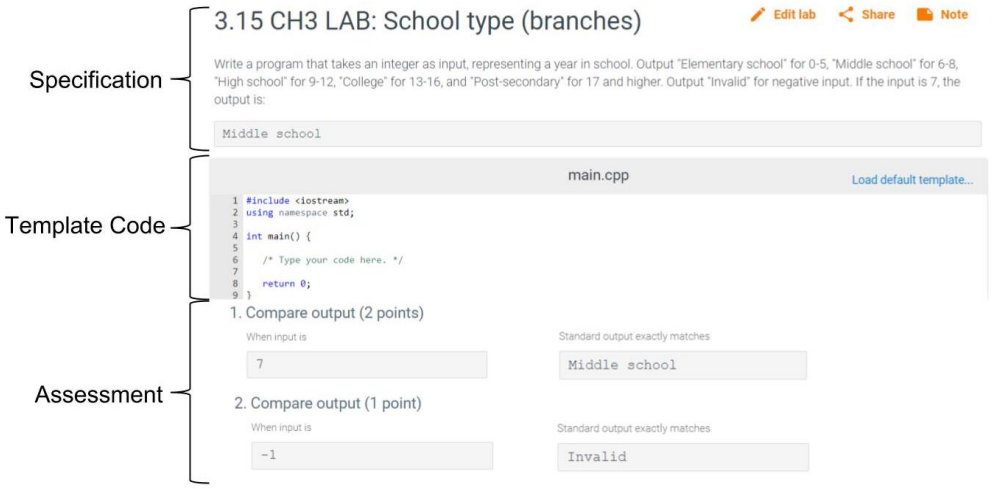
All universities included in this study use zyBooks' auto-grader for programming assignments. In 2016, zyBooks [13] released a web-based program auto-grading system, known as "zyLabs", that emphasizes ease of use for students and for instructors. Students using zyBooks can code directly in the web-browser, and are not required to have a third party IDE for program development (though file uploads are supported as well). Instructors using zyBooks can create new assignments by using a web interface that does not require additional scripting or coding. This means no special training is needed for instructors or TAs that wish to create or revise programming assignments. In our initial work, we created 61 new MSPs, each requiring about 30 minutes to think up, create, and implement.

The process to create a new programming assignment using zyBooks involves filling out a web form. Each assignment has four parts: a title and prompt, compiler flags, a code template (optional), and test cases. The instructor begins by entering a title for the lab and creating a text prompt for the assignment. Next, the instructor can set various compiler flags such as number of submissions allowed, submission metering, etc. The instructor can then choose to provide a starting code template for students to use when working on the assignment. Finally, the instructor can create test cases and assign points to each test.

zyBooks currently supports "input/output" tests as well as unit tests. An “input/output” test case involves input values paired with expected output values. If a program should output the square root of its input, then an “input/output” test case may have an input of 9 and an output of 3. The instructor can create any number of test cases and assign any point value to each test case.

Figure 2 shows the zyBooks' auto-graded programming assignment interface. The specification section consists of the lab title and the programming assignment prompt. Note that the lab in Figure 2 is an MSP and has a short and concise prompt. Next, an instructor can choose to provide template code for students to use in the template code section. Note that in this example, main is provided as well as basic include statements. Finally, the assessment section shows basic input/output test cases for the students' program to pass. Grading is immediate and the students receive immediate score feedback based on the test cases their program passes.

Fig. 2. Example programming assignment (in this case, an MSP program).



4. MSP Usage Analysis

Section 4 provides analysis of a variety of research questions about MSPs. Section 4.1 provides details on our data collection method and sections 4.2 through 4.4 answer common questions related to MSP usage in CS1 courses.

4.1 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 submissions and develops for each programming assignment from zyBooks and consolidated them into a single spreadsheet. A submission 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 submissions have metadata that describes the lab title, a userID (anonymized and generated from zyBooks), the submission score, the max score possible for the submission, and a timestamp with the date and time. A develop has the same metadata as a submission but without a score and a max score. In this study, 1,179 students were considered with 567 MSPs included. In total, we collected 192,657 submissions and 302,406 develops.

4.2 How much time do students spend working on MSPs?

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

4.2.1 Analysis and Procedure

To calculate the total time students spent on each MSP, we looked at the timestamp metadata for each develop and submission. We calculated the time spent between submissions 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 actually 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 submission.

4.2.2 Results

Figure 3 summarizes the average time spent by students on each MSP. Note that since all MSPs are included in this calculation, including the "easy" introductory MSPs 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 MSPs. Across all universities observed, students spend an average of 12 minutes working on MSPs, with universities 1 and 6 slightly pulling the averages down.

Fig. 3. Average time spent by students on each MSP. Students with 0 submissions or 0 time spent were excluded from calculations.

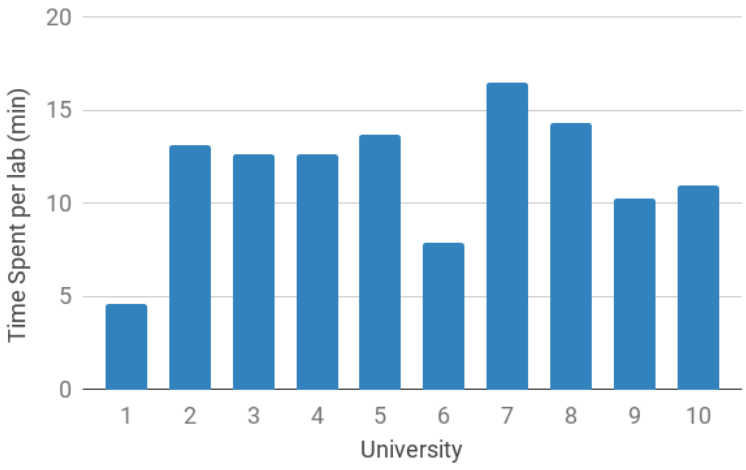


Figure 3 shows that many universities share similar time spent averages. Assuming that students are given 5 MSPs each week, students should be working on programming assignments for about 1.5 hours. There are a few exceptions, like universities 1 and 6, but a more detailed analysis is required to determine why the time spent per lab averages are so low.

4.3 How many days before the due date do students start MSPs?

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

4.3.1 Analysis and Procedure

To calculate when a student began working on an MSP, we first determined each students' first submission timestamp. Once we knew the first submission, using this in combination of knowing the MSP due date, we computed the number of days each student began working on MSPs before the due date. Note that zyBooks does not keep track of MSP due dates, so we had to calculate due dates based on student submissions. We found the two most active dates for submissions (days that had the most submissions by unique users) and then chose the later date as the due date for that MSP. We checked our due date calculations by comparing our results with known due dates of prior MSPs and had ~90% accuracy. Most inaccuracies were +/- a day and affected MSPs assigned during the beginning of the term -- likely caused by students adding/dropping the course at the start of the term.

4.3.2 Results

Figure 4 summarizes the average number of days students began working on MSPs 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.

Fig. 4. Average T-X days prior to the due date students began working on MSPs.

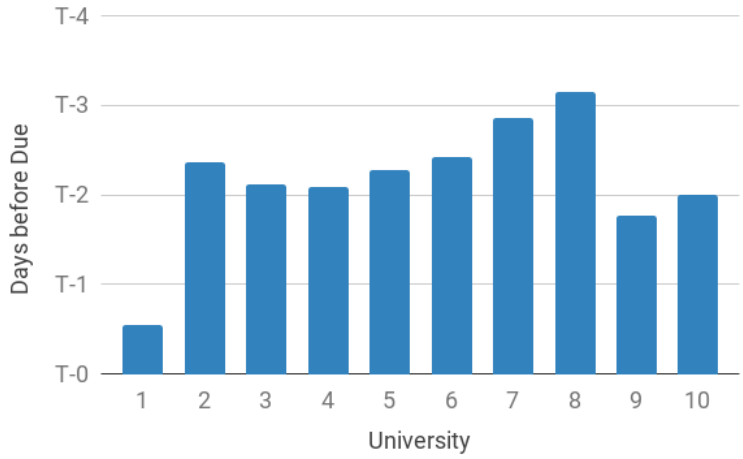


Figure 4 shows that on average, students begin working on MSPs about 2 days before the due date, with the exception being university 1 which seems to have most students starting on the due date. Note that this is the best approximation we can make without specifically knowing the due dates for all MSPs. Also note that the data given could be slightly off as we do not know the duration students have to work on MSPs - one week, three days, etc.

4.4 How do students score on MSPs

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

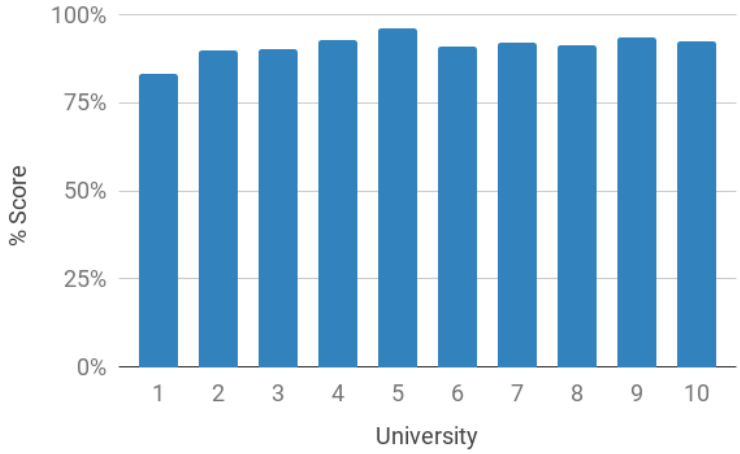
4.4.1 Analysis and Procedure

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

4.4.2 Results

Figure 5 summarizes the average percentage score on each MSP. The x-axis is the university and the y-axis is the MSP percentage score.

Fig. 5. Average percentage score on MSPs.



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

5. 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 students using MSPs 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 analysis, compared with the several of our previous work, we still believe this to be a good indication that MSPs 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.

6. Conclusion

In this work, we performed several analyses of MSPs 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 MSPs in beneficial ways. We see that students are spending sufficient time working on weekly MSPs (~1.5 hours), that students begin working about 2 days before the MSP deadline, and students are scoring high grades on MSPs (91% avg). Based on the positive results of this work, we are encouraged to continue improving MSPs and studying them more in depth. Already, we have switched all CS1 offerings at our university to using MSPs and we encourage others to consider using MSPs in their CS1 course as well.

References

[1] Joe Michael Allen, Frank Vahid, Kelly Downey, and Alex 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.
[2] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris 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.
[3] Autolab. http://www.autolabproject.com/. Accessed: January, 2019.
[4] Theresa Beaubouef and John Mason. 2005. Why the high attrition rate for computer science students: some thoughts and observations. SIGCSE Bull. 37, 2 (June 2005), 103-106. DOI: http://dx.doi.org/10.1145/1083431.1083474
[5] Päivi Kinnunen and Lauri Malmi. 2006. 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. DOI: http://dx.doi.org/10.1145/1151588.1151604
[6] Matlab Grader. https://grader.mathworks.com/. Accessed: January, 2019.
[7] Mimir. https://www.mimirhq.com/. Accessed: January, 2019.

[8] Andrew Petersen, Michelle Craig, Jennifer Campbell, and Anya Tafliovich. 2016. 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. DOI: https://doi.org/10.1145/2999541.2999552
[9] Submitty. http://submitty.org/. Accessed: January, 2019.
[10] Turing's Craft: CodeLab. https://www.turingscraft.com/. Accessed: July, 2018.
[11] Christopher Watson and Frederick W.B. Li. 2014. 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. DOI: http://dx.doi.org/10.1145/2591708.2591749
[12] Web-CAT. http://web-cat.org/. Accessed: January, 2019.
[13] zyBooks. https://www.zybooks.com/catalog/zylabs-programming/. Accessed: January, 2019.