Weekly Programs in CS 1: Experiences with Many-Small Auto-Graded Programs

Joe Michael Allen*, Frank Vahid*, Kelly Downey*, Alex Edgcomb†
* University of California, Riverside, CA, † zyBooks, Los Gatos, CA
Email: jalle010@ucr.edu, vahid@cs.ucr.edu, kelly@cs.ucr.edu, alex.edgcomb@zybooks.com

ABSTRACT
We describe an experiment in changing a CS 1 introductory programming course from the traditional 1 larger programming assignment per week to 7 smaller assignments per week: "many-small" assignments. The change was enabled by a program auto-grader in which each new assignment could be created in about 30 minutes, and that gave students immediate score feedback. Students could earn up to 10 points per assignment, and we defined 50 out of 70 possible points as full program credit for the week. With that setup, we allowed collaboration. The change was made for one of four class sections, that section having 80 students, at a research university whose CS 1 course serves about 350 students/quarter (about 1200 students/year, majors and non-majors). Our main goal was to improve the student's experience. Via student surveys, we found significantly higher satisfaction, less stress, and higher confidence among students. Because collaboration was allowed, for the first time in decades, instructors spent no time dealing with academic dishonesty cases, and no student asked for an extension. We of course did not want student performance to worsen. Via analysis of exams (half multiple choice and half coding problems), we found no worsening, and in fact found significant improvement. Also, the instructors and teaching assistant reported their own high satisfaction. As a result, the department will be changing all class sections to utilize the many-small auto-graded programs approach.

CCS Concepts
• **Social and professional topics~CS1** • *Social and professional topics~Student assessment* • *Applied computing~E-learning* • *Applied computing~Interactive learning environments* • *Human-centered computing~Empirical studies in interaction design*

Keywords
CS 1; Programming assignments; Auto-grading; Lab; Student experience.

1 INTRODUCTION

Issues in college-level introductory programming courses (known as CS 1) are well-known: high student stress, dissatisfaction, academic dishonesty, low grades, and high drop rates. A CS 1 course's weekly programming assignments form a large part of the student's experience and are a key source of those issues.

At our university, our CS 1 course followed the traditional one-program-per-week model, with a few small warm-up programs, for over 20 years. Our CS 1 serves about 350 students per quarter (including majors and non-majors). We've used a program auto-grader for the past 10 years. We've 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, and we use pair programming at times as well. Like at many schools, students are not permitted to collaborate on the larger weekly assignment, and instead may get help during instructor/TA office hours or via discussion board posts to the class. Our "lectures" have included small-group collaborative programming (a form of "flipped" classroom) since the late 1990's. Student evaluations indicate that the course is reasonably well-liked by students, though many students indicate that it is hard, time-consuming, and/or stressful. Like many schools, we automatically check for and detect overly-similar submissions. Investigating and pursuing academic dishonesty (typically 10-20 per quarter, sometimes more) is a time-consuming and unpleasant part of the instructor's job.

Auto-graded programs: 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 helping, and enabling TAs to handle larger 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 cheating.

Last year, a publisher released a web-based program auto-grading system that emphasized ease of use, not only for students (including an option to code directly in the web rather than uploading a file), but also for instructors creating new assignments via a simple web interface, requiring no scripting or coding. With the system, any of our instructors or TAs could easily create new assignments, with no training or special skills. Creating each of the larger weekly programming assignments required only about 60-90 minutes, versus many hours in the past and only by a specific TA trained/skilled with the in-house auto-grader. Thus, we created a new set of weekly assignments for winter 2016, along with new warm-up assignments, and revised the assignments quarterly (revisions used to be yearly or less).

Many-small programs: The ease of creating new program assignments, coupled with students getting immediate fine-grained score feedback, enabled us to consider a not-previously possible option. This paper describes an experiment in which, for one of four class sections, we replaced the 1 program per week by 7 smaller programs—what we call "many-small" programs—that focused on the concept being taught that week.

Collaboration (no academic dishonesty): Due to the lower stakes for any one program, we decided to allow students to collaborate. The net result was happier students/instructors/TAs, improved performance, and (for the first time in decades) almost no efforts expended by the instructors related to detecting or punishing academic dishonesty.

In this paper, we summarize the course setup and the program auto-grading system. We highlight the topics of the many-small programming assignments. We provide results of student surveys, showing significantly happier students. We provide midterm and final exam results, showing improved performance. Based on the

results, our department will be modifying all future sections to use the many-small program assignment approach.

2 RELATED WORK

Much work has focused on improving the student experience in CS courses, especially CS 1. Flipped classrooms [3][12][13][21][29] have students do work before class, so class time can be used for working on problems, group activities, discussions, etc. Flipped classrooms have shown various benefits, including improved performance, less drops, and happier students. Studio-based learning [15][16] emphasizes student communication, collaboration, and critical thinking skills, showing improvements in student attitude and content mastery.

Researchers have examined how student collaboration and instruction affects the student experience. Rodriguez [28] examined how pair programming and student collaboration affected learning outcomes, finding that if pair programming is done properly, collaboration increases learning and understanding. Blaheta [4] studied cooperative learning and found that students had a positive reaction. Simon [24] found that peer instruction had a positive impact on student perception of learning. Simon [26] did another study; focusing on three new practices: media computation, pair programming, and peer instructions, finding improved student retention. Work [19][25][31] continues to study the effects of peer collaboration, finding many benefits.

Research has explored effects of changing the programming language or applications. Norman [22] studied switching a course's language from C++ to Python and replacing weekly assignments with labs and online problem sets. They reported an increase in exam, lab, and overall course scores. Layman [18] examined assignments from 21 CS 1 courses and found only 34% had a practical component. Guzdial [14] developed a course to teach programming using Python to manipulate sound, images, and movies. Game design and development [5][20][30] have been used to improve the student experience.

Edgcomb [10] analyzed results after introducing an interactive textbook to an introductory STEM class and found improvements in multiple aspects of the class.

Other studies include allowing students to author questions that would appear on their exam [8], creating online tutors for students to utilize [17], and assigning students the responsibility to review and grade their peers' assignments [2].

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 instructor's time. Universities and companies have built automated homework grading systems and are studying how to effectively use them [1][9][11][33]. In addition, smaller coding problems are being introduced into classrooms. Systems such as CloudCoder [6], CodingBat [7], Pearson's MyProgrammingLab [23], Problets [27], Turingscraft's CodeLab [32], and zyBooks' Challenge Activities [34] have been frequently used for homework, warm up, or extra practice. In contrast, we are examining redesigning the weekly programming assignments themselves. We continue to use publisher-provided small coding problems for homework as well (our students do several hundreds of those small coding problems during the term).

3 SETUP

3.1 The Program Auto-Grader

The program auto-grader that we used is published by Zyante. Zyante's auto-grader is a fully web-based system that makes creating and grading assignments very simple. An instructor creates a new assignment by clicking a "New assignment" button, which 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, how many submissions are allowed (we selected unlimited), and whether submissions are metered (we did not meter. The instructor can provide a code template for students (we usually provided a basic template having includes and the main() function, but little else).

Next, the instructor creates test cases. "Input/output" test cases involve the instructor just typing input values and expected output values. For example, if a program should square its input, the instructor might create a test case with input 3 and output 9, and another with input -5 and output 25. The instructor can create any number of test cases, and assign any point value to each test case. The instructor can also configure the test case to ignore output whitespace, to 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 the instructor can directly call a student's function/method and check the returned result.

3.2 The Course

The experiment was conducted in our CS 1 course at a U.S. public research university. The CS department is typically ranked in the top 60 by U.S. News and World Report. The course serves about 350 students per quarter, with 3 quarters per year, plus summer. The course in this paper was in Spring 2017, which had just over 300 students, all being non-computing majors: bioengineering, civil engineering, electrical engineering, physics, biology, chemistry, math, and more. The course is typically taught with 4 sections of 80-100 students each. 4 instructors rotate to teach the course, all with over 5 years of experience, and all with strongly-positive student evaluations. Usually two instructors teach the course in a given quarter, each with their own sections. Each section consists of three hours of instructor-led "lecture" per week, where our lectures since the late 1990's have consisted of short talks (often with the instructor coding small examples live) followed by small-group coding activities, repeated several times per session. All sections also have two scheduled-lab hours per week, led by a TA. Instructors / TAs hold weekly office hours. An online discussion board is actively used for questions / answers.

3.3 The Experimental Group

One of the four class sections was the experimental group, with the other three being the control group. The experimental group contained about 80 students and the control group contained about 240 students. Most features were kept the same for all four sections, except as described below related to the weekly program assignments. All sections took the same midterm and final exams.

3.4 Course Tasks and Grades

In all sections, students were assigned three tasks each week. (1) Reading tasks consisted of completing small activities found in our online textbook, consisting mostly of answering questions (multiple choice, true/false, short-answer). 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) Program assignments required students to apply that week's topics by writing one or more full programs.

In the experimental group, program assignments accounted for 15% of the course grade (vs. 25% for the control group). In both groups, students earned 7.5% for the reading tasks, 7.5% for the homework tasks, and 5% for in-class participation. The midterm was 30% and the final 35%, vs. 20% and 35% for the control group. Both groups took the same exams. Each exam was half multiple-choice and half coding problems.

The experimental group was given 7 small programming assignments per week, vs. the usual 1 large programming assignment per week plus warm ups in the control group. Each many-small program was worth 10 points. Students could earn 0-10 points, depending on how many test cases their program passed. In the experimental group, students 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 feedback on scores. Neither group had limits on the number or rate of submissions.

With each many-small program being lower stakes in the experimental group vs. the control group, the experimental group was told that they could collaborate, vs. 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). They were told the only thing considered academic dishonesty would be having someone else simply write their program. They were told that half the midterm and final exams consisted of short coding problems of similar difficulty as the assignments, and were given sample midterm and final exams illustrating that fact.

4 MANY-SMALL PROGRAMMING ASSIGNMENTS

The course covered input/output, variables/assignments, branches, loops, functions, and vectors, in C++, in 9-weeks (the 10th week covered various topics, not involving program assignments). We summarize the weekly program assignments below.

CH1 Introduction: Hello World! / No parking sign / Welcome message / Mad lib / House real estate summary / Right-facing arrow / Caffeine levels. Generally, students output formatted text, obeying all whitespace, sometimes involving simple variable values being output. Instructor's solution sizes were about 8-10 lines of code (the first week starts very simple).

CH2 Variables/Assignments: Divide by x / Driving costs / Simple statistics / Using math functions / Musical note frequencies / Expression for calories burned during workout / Phone number breakdown. Generally, students must compute values using simple variables and assignment operators. Instructor's solutions: 10-15 lines.

CH3 Branches: Vending machine / Largest number / Remove gray from RGB / Leap year / Interstate highway numbers / Seasons / Exact change. Generally, students must write a variety of if-else structures to generate correct output based on different input values. Instructor's solutions: 20-25 lines.

CH4 Strings/Loops 1: Convert to binary / Name format / Varied amount of input data / Count characters / Repeating mad lib / Password modifier / Palindrome. Generally, students used loops to solve problems, output formatted text, and make changes to variables. Instructor solutions: 15-20 lines.

CH5 Loops 2: Remove spaces / Count input length without spaces, periods, or commas / Output range with increment of 10 / Print name in reverse / Checker for integer string / Countdown until matching digits / Brute force equation solver. Generally, students used loops solve problems, output formatted text, and make changes to variables. Instructor solutions: 20-25 lines.

CH6 Functions 1: Miles to track laps / Max magnitude / Driving cost / Step counter / Flip a coin / Acronyms / A jiffy. Generally, students created functions using pass-by-value parameters to solve problems and output formatted text. Instructor solutions: functions 5-10 lines, main 5-10 lines.

CH7 Functions 2: Count characters - functions / Remove spaces - functions / Leap year - functions / Max and min numbers - functions / Convert to binary - functions / Swapping variables - functions / Exact change - functions. Generally, students created functions, sometimes using pass-by-reference parameters or functions calling functions, to solve problems and output formatted text. Instructor solutions: functions 10-15 lines, main 5-10 lines.

CH8 Vectors 1: Output numbers in reverse / Middle item / Output value below an amount / Adjust list by normalizing / Word frequencies / Replacement words / Contains the character. Generally, students used vectors to more elegantly solve previous assignments, or to solve new problems involving lists of items. Instructor solutions: 35-45 lines.

CH9 Vectors 2: Elements in a range / Two smallest numbers / Even-odd values in a vector / Contact list / Sort a vector. (Only 5 assignments this week, with only 35 points needed for full credit). Generally, students used vectors to solve more challenging problems involving lists of items. Instructor solutions: 40-50 lines.

Each week typically involved 2 easy assignments, 3-4 medium, and 1-2 hard.

5 STUDENT SURVEYS

Our main goal was to improve the students' experience, not only to hopefully reduce attrition, but also to attract students to computing majors. We created what we internally call a "stress survey", asking about a student's experience. The questions are shown in Table 1. To reduce bias towards answering positively, we phrased some questions such that higher is favorable, and others such that lower is favorable. The survey was given in the 8th week of a 10-week quarter.

For nearly all questions, the answers were more favorable for the experimental group versus the control group. Questions where the difference was approaching statistical significance ($p < 0.05$) are denoted by * and questions where the difference was significant under the Bonferroni correction ($p < 0.0028$) are denoted by **.

We did not expect the experimental group would be spending any more time per week on programs, because although more programs existed, they were smaller. We expect all CS 1 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 1, the experimental group students also 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).

Table 1: Results of the "stress survey" for Spring 2017. p-values denoted with * are nearing significance ($p < 0.05$) and p-values denoted with ** are significant under the Bonferroni correction ($p < 0.0028$). Most favored the experimental group. Note that for some questions, lower numbers are better, for others higher.

Question	**Control group average**	**Experimental group average**	**p-value**
I enjoy the class.	4.53	4.87	0.046*
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
This class is an appropriate amount of work per week for the number of units.	3.73	4.09	0.073
I felt anxious before the midterm exam.	4.25	4.18	0.396
I was prepared for the midterm exam.	3.63	4.18	0.004*
I feel anxious about the final exam.	4.89	4.37	0.020*
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 were stressful.	4.31	3.93	0.058
The weekly programming assignments were frustrating.	4.34	3.99	0.078
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

In addition, we conducted a z-score analysis. Per question, we z-scored all responses. Then, we concatenated the z-scores from the control group into one list, and separately, the experimental group into another list. Then, 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.

Further, we converted the average of each list into a percentage difference via: (1) Calculate the absolute difference between the average of each list, (2) Convert the difference into a percentage using the online tool: https://measuringu.com/pcalcz/ (two-sided),

Table 2: Results of a z-score analysis on the "stress survey" for Spring 2017. The experimental group preferred the course 9% more than the control group.

Control group z-score	Experimental group z-score	p-value	z-score %
-0.0707	0.1563	1.68E-69	9%

(3) Divide 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 than the control group. Table 2 shows our results.

Lastly, we also sought to compare the students' experiences between the current experimental group and the previous time that same instructor taught the course, to determine whether the instructor alone might account for the differences. The instructor provided his/her teaching evaluations for his/her previous offering (55 students completed the evaluation 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 an unusually high 4.72 (department average 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 went from 4.3 (dept. avg. was 4.2) to 4.67 (dept. avg. was 4.25). In fact, the course evaluations were in the 85'th 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 many-small labs had a strong positive impact on the students' experiences.

6 PERFORMANCE

We sought to improve the student experience without worsening student performance. We thus compared the experimental and control groups' performance on the midterm and final exams. The exams were half multiple-choice questions, and half short coding questions (both in terms of points, and approximate time). Table 3 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 activities. The control group performed better than the experimental group on homework activities.

Table 3: 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**

Our department runs one section fully online each quarter. The online section is run identically to the physical sections, with the lectures and lab sessions carried out via synchronized online meetings, with required real-time attendance (like the physical sections). The experimental group happened to be the online section. 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 obtained and analyzed

Table 4: Physical vs. online averages for Spring 2016, Fall 2016, and Winter 2017, for about 1,000 physical and 300 online students, showing little difference in exam scores.

	Physical	Online
Final	79.3%	78.4%
Midterm	79.7%	78.3%
Reading activities	91.8%	93.1%
Homework activities	93.7%	91.7%
Weekly programming activities	87.9%	83.1%

the scores for the past three offerings of the CS 1 course. Table 4 shows that the online section traditionally does not outperform the physical sections, instead doing slightly worse on the exams.

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), and that instructor's online section never previously outperformed the other sections, instead usually doing slightly worse, consistent with Table 4.

7 DISCUSSION

The experimental group had three related changes: many-small programs vs. one-larger-program per week, allowed collaboration on those programs, and those programs being worth only 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 % and increasing exam %, but such a change was highly unpopular, with students in those cases complaining they spent 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 many-small programs are less intimidating, meaning students are less likely to seek inappropriate help, and more likely to attempt the programs themselves. The many-small programs are more focused, so students can more clearly see themselves improving their skills (e.g., each loop gets easier to write) and can envision those skills helping on the exams. Those factors enabled us to allow collaboration, since students won't immediately cheat such a system, instead first making attempts because the programs are approachable and their usefulness more clear, and then later getting help if needed. (One might note that the experimental group's answer to survey question "I frequently collaborated…" is not higher than the control group). Likewise, those factors allowed us to reduce the program %, 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 many-small programs approachable and useful.

One might ask, given the rampant cheating common in CS 1, that isn't allowing collaboration going in the wrong direction? We disagree. We believe most students want to learn and will do so given what they understand to be a fair and valuable learning experience. With such an experience, collaboration can help the learning (see related work section). We note that the teaching assistant, when told of the policy that collaboration was OK, was supportive and shared this story from when he/she took CS 1 at our school years earlier: "When I was a freshman, I worked with classmates in a way I assumed was OK, but we got called in for suspected cheating. I was found not guilty, but I was so shook up by the experience—it was terrifying—that I didn't work with any classmates again for two years." The positive benefits of student collaboration are well known, and we wish to encourage such collaboration in CS 1.

Ultimately, we believe this is a case of technology enabling a new approach and perspective. The traditional one-program-a-week model may never have been best for students, but were what instructors could manage. In particular, when programs were graded by hand (and when creating auto-graded assignments was time-consuming), instructors couldn't 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 didn't know how their scores for a week or more. When one looks at other skills, like piano or basketball, one notes that instructors do not set up one recital or one game per week. Instead, they have students spend extensive time doing numerous focused drills, like playing scales or focused pieces, or doing lay ups and shooting free throws, with immediate feedback from the instructor as well.

We also note that, for probably the first time in decades of teaching the course, not a single student requested an extension. The many-small programs has a deadline, but there was no single high-stakes big deadline.

One might ask, if CS 1 only has many-small programs, when will students learn to write larger programs? Our thoughts:

- Majors will learn to write larger programs in CS 2.
- Non-majors, if they need to program in their careers, are more likely to have to write programs similar to the many-small programs, 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 probably will take a CS 2 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 larger assignment mid-quarter, and one larger assignment end-of-quarter, with the other 8 weeks using the many-small programs approach.

Finally, we note that the instructor who taught the experimental section stated that he/she "is never going back". The instructor has always had positive experiences teaching, and (like the other CS 1 instructors) has above-average student evaluations and positive student comments (and has received several teaching awards as well). Still, the instructor said this was the best CS 1 teaching experience he/she has had in 20 years, and believes it was also the best experience for the TA as well.

8 CONCLUSION

New technology, namely a program auto-grader with rapid/easy assignment creation, enables replacing the traditional CS 1 one-program-per-week approach by a "many-small" programs approach. A many-small approach involves numerous smaller focused programs due each week, accompanied by a scoring threshold approach (in our case, 50 of 70 points for the week yields full credit). The approach is less intimidating for students, allowing them to build confidence and skill on the easier small programs, and then to work on the harder ones, skipping around if necessary, earning partial points on one and then moving to another, and then returning to uncompleted programs. The many-small approach enabled us to allow collaboration, which was an opportunity students seemed to not abuse. It 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 really don't know who is doing the programming). The approach led to significantly greater satisfaction and less stress among the students. The approach also yielded improved performance on the exams. And the approach led to improved experiences for the instructor and TA, including not having to spend any time on academic dishonesty and having more positive interactions in the lectures and labs. As a result of the experiment, our department will be switching all sections to primarily use the many-small approach, and we encourage other departments to consider the approach.

REFERENCES

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

[2] L. de Alfaro, M. Shavlovsky. 2014. CrowdGrader: a tool for crowdsourcing the evaluation of homework assignments. SIGCSE ACM technical symposium on Computer science education, 415-420.

[3] J. Bishop, M.A. Verleger. 2013. The Flipped Classroom: A Survey of the Research. ASEE Annual Conference and Exposition.

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

[5] D.C. Cliburn, S.M. Miller. 2008. Games, Stories, or Something More Traditional: The Types of Assignments College Students Prefer. SIGCSE '08 Proceedings of the 39th SIGCSE technical symposium on CS education.

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

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

[8] P. Denny. 2015. Generating Practice Questions as a Preparation Strategy for Introductory Programming Exams. SIGCSE ACM Technical Symposium on Computer Science Education, 278-283.

[9] P. Denny, A. Luxton-Reilly, E. Tempero, J. Hendrickx. 2011. CodeWrite: supporting student-driven practice of java. ACM Technical Symposium on Computer Science Education, 471-476.

[10] A. Edgcomb, F. Vahid, R. Lysecky, A. Knoesen, R. Amirtharajah, M.L. Dorf. 2015. Student Performance Improvement using Interactive Textbooks: A Three-University Cross-Semester Analysis. ASEE Annual Conference and Exposition.

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

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

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

[14] M. Guzdial. 2003. A Media Computation Course for Non-Majors. ITiCSE annual conference on Innovation and technology in computer science education, 104-108.

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

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

[17] A.N. Kumar. 2004. Using Online Tutors for Learning – What do Students Think? Frontiers in Education.

[18] L. Layman, L. Williams, K. Slaten. 2007. Note to self: make assignments more meaningful. SIGCSE technical symposium on CSter science education, 459-463.

[19] C.B. Lee, S. Garcia, L. Porter. 2013. 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 Volume 13 Issue 3, Article No. 12.

[20] L. Soh. 2004. Using Game Days to Teach a Multiagent System Class. SIGCSE '04 Proceedings of the 35th SIGCSE technical symposium on Computer science education. 219-223.

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

[22] V. Norman, J. Adams. 2015. Improving Non-CS Major Performance in CS1. SIGCSE Technical Symposium on Computer Science Education, 558-562.

[23] Pearson: MyProgrammingLab. https://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/. August 2017.

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

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

[26] Porter, L. and Simon, B. Retaining nearly one-third more majors with a trio of instructional best practices in CS1. SIGCSE ACM technical symposium on Computer science education, pgs 165-170, 2013.

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

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

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

[30] S. Leutenegger, J. Edgington. 2007. A Games First Approach to Teaching Introductory Programming. SIGCSE '07 Proceedings of the 38th SIGCSE technical symposium on Computer science education.

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

[32] Turingscraft: CodeLab. https://www.turingscraft.com/. August 2017.

[33] C. Wilcox. 2016. Testing Strategies for the Automated Grading of Student Programs. ACM Technical Symposium on CS Education, 437-442.

[34] zyBooks. http://www.zybooks.com/. August 2017