Weekly Programs in a CS1 Class: Experiences with Auto-graded Many-small Programs (MSP)

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, University of California, Riverside

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.

Dr. Alex Daniel Edgcomb, Zybooks

Alex Edgcomb finished his PhD in computer science at UC Riverside in 2014. Alex works with zyBooks.com, a startup that develops interactive, web-native textbooks in STEM. Alex has also continued working as a research specialist at UC Riverside, studying the efficacy of web-native content for STEM education.

Weekly Programs in a CS 1 Class: Experiences with Auto-Graded Many-Small Programs (MSP)

Abstract

We describe an experiment in changing a CS 1 introductory programming course from the traditional one large programming assignment per week to seven small assignments per week: "many-small programs" (MSPs). The change was enabled by a program auto-grader that allowed easy creation of each new assignment in only 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 (no extra credit). With that setup, we allowed collaboration. The change was made for one of three class sections (about 80 students per section) in Spring 2017 at our research university whose CS 1 course serves about 350 students/quarter (over 1,000 students/year, majors and non-majors), with a diverse student population. Our goal was to improve the student's overall experience in the course. Via student surveys, we found less stress, more confidence, and higher satisfaction. Students using MSPs were less anxious about the class (3.15 vs. 3.72; on 6-point scale; p-value = 0.02) and found the weekly programming assignments more enjoyable (4.13 vs. 3.37; on 6-point scale; p-value = 0.001). Students using MSPs scored a very substantial 20 percentage points better on the coding half of the midterm, for an overall midterm improvement of 10 percentage points (p-value < 0.001). Students using MSPs scored 8 percentage points better on the coding portion of the final, for an overall final improvement of 5 percentage points (p-value < 0.01). The instructor and teaching assistant reported their own high satisfaction. Since collaboration was allowed, for the first time in decades, the instructors spent no time dealing with academic dishonesty cases. Unlike most past terms, no student asked for an extension. As a result, the department has since changed all sections to use MSPs, with continued success.

Introduction

College-level introductory programming courses (known as CS 1) have many well-known issues: high student stress, dissatisfaction, academic dishonesty, low grades, and high drop rates. 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 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 large weekly assignments.

Students may, however, get help during instructor/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/or stressful. We also check for and detect overly-similar submissions using MOSS. Investigating and pursuing academic dishonesty (typically 10-20 per quarter) 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 handling 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 academic dishonesty.

In 2016, a publisher 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.

Many-small programs (MSPs): The ease of creating new program assignments coupled with students getting immediate, fine-grained score feedback enabled us to consider a new option. This paper describes an experiment in which, for one of three class sections, we replaced the one-program-per-week model by seven smaller programs—what we call "many-small" programs (MSPs)—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 MSP assignment approach. We provide results of student surveys showing significantly happier students. We provide exam results showing improved performance. Based on the results, our department has changed all sections to use the MSP assignment approach.

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, fewer 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][30] 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][31] have been used to improve the student experience.

Other studies include allowing students to author questions that would appear on their exam [8], creating online tutors for students to utilize [17], introducing interactive textbooks in classrooms [10], 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 instructors' 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] design small coding problems used as homework, warm-up, or extra practice in the classroom. 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).

Program auto-grader

We used a program auto-grader published by zyBooks [34]. zyBooks' auto-grader is a fully web-based system that makes creating and grading assignments simple. An instructor creates a new assignment 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 submissions allowed (we selected unlimited), and whether submissions 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.

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 usually serves about 350 students per quarter (three quarters per year, plus summer) with four sections containing 80-100 students per section. In 2013, our university made one section completely online. The online section is run identically to the physical 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 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 CS 1 course had about 250 students split into three sections. Most students were non-computing majors.

The experimental group

The experimental group was the online section and contained about 80 students. The control group was the other two sections and contained about 160 students. Most features were kept the same for all three sections except the weekly program assignments, the midterm percentage, and allowing collaboration as described below. All sections took the same midterm and final exams.

Course tasks and grades

In all 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) Program 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 program assignments (control 25% vs. experimental 15%) and the midterm exam (control 20% vs. experimental 30%).

The experimental group was given seven MSP assignments per week vs. the usual one large programming assignment per week (plus warm-ups) in the control group. Students could earn 0-10 points, per MSP, 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.

With each MSP 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). 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 assignments, and were given sample midterm and final exams illustrating that fact.

Many-small program 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 program assignments). Solution sizes ranged from 10-50 lines of code. Each week consisted of two easy, 3-4 medium, and 1-2 hard MSPs.

Student surveys

Our main goal was to improve student experience; hoping to reduce attrition and also attract students to computing majors. We created a "stress survey" to ask students about their experience. The questions are shown in Table 1. To reduce bias towards answering positively, we phrased some questions such that higher is favorable (listed above the bolded line) and others such that lower is favorable (listed below the bolded line). 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 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 average	Experimental 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

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 the online tool: https://measuringu.com/pcalcz/ (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 programs per week since 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 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 MSPs had a strong positive impact on the students' experiences.

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 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 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 activities	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 CS 1 course. Table 3 shows that the online section traditionally does not outperform the physical sections, and in fact, typically performs slightly worse on the exams.

Table 3: 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%

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 3. This data increases our confidence that the improvements seen in the online section of Spring 2017 is indeed due to the introduction of MSPs.

Discussion

The experimental group had three related changes: MSPs vs. one large program per week, allowed 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 MSPs are less intimidating, meaning students are less likely to seek inappropriate help and more likely to attempt the programs themselves. The MSPs 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 more clear. (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 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 many-small programs approachable and useful. Also, no student asked for an extension.

One might ask, given the rampant cheating common in CS 1, 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 (see related work) 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 was what instructors could manage. In particular, 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 CS 1 only has MSPs, 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 MSPs, 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 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 large assignment mid-quarter and one large assignment end-of-quarter, with the other eight weeks using the MSP approach.

The MSP students did just as well as the non-MSP students for the next term in CS 2, which uses the traditional one large program per week and does not allow collaboration.

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 CS 1 instructors) has above-average student evaluations and positive student comments. Still, the instructor said this was the best CS 1 teaching experience they had in 20 years.

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. 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 programs and then work on the harder ones; skipping around until they earn enough points. The MSP approach enabled us to allow collaboration. MSPs 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 (vs. programming assignments where even with a no collaboration policy, instructors cannot be certain 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. 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.

Acknowledgements

This work was supported by the U.S. Dept. of Education (GAANN fellowship) and by Google.

References

[1] T. Ahoniemi, E. Lahtinen, and T. Reinikainen, "Improving pedagogical feedback and objective grading," *SIGCSE Technical Symposium on Computer Science Edu.* pp. 72-76, 2008.
[2] L. de Alfaro and M. Shavlovsky, "CrowdGrader: a tool for crowdsourcing the evaluation of homework assignments," *SIGCSE technical symposium on Comp. sci. edu.*, pp. 415-420, 2014.
[3] J. Bishop and M.A. Verleger, "The Flipped Classroom: A Survey of the Research," *ASEE Annual Conference and Exposition*, 2013.
[4] D. Blaheta, "Reinventing homework as cooperative, formative assessment," *ACM Technical Symposium on Computer Science Education*, pp. 301-306, 2014.
[5] D.C. Cliburn and S.M. Miller, "Games, Stories, or Something More Traditional: The Types of Assignments College Students Prefer," *SIGCSE technical symposium on Comp. sci. edu.*, 2008.
[6] CloudCoder. https://www.cloud-coder.com/. August 2017.
[7] CodingBat. http://codingbat.com/about.html. August 2017.
[8] P. Denny, "Generating Practice Questions as a Preparation Strategy for Introductory Programming Exams," *SIGCSE Technical Symposium on Computer Science Education*, pp. 278-283, 2015.
[9] P. Denny, 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.
[10] A. Edgcomb, F. Vahid, R. Lysecky, A. Knoesen, R. Amirtharajah, M.L. Dorf, "Student Performance Improvement using Interactive Textbooks: A Three-University Cross-Semester Analysis," *ASEE Annual Conference and Exposition*, 2015.
[11] N. Falkner, 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.
[12] S. Findlay-Thompson and P. Mombourquette, "Evaluation of a Flipped Classroom in an Undergraduate Business Course," *Business Education & Accreditation*, v. 6 (1) p. 63-71, 2014.
[13] M.B. Gilboy, S. Heinerichs, G. Pazzaglia, "Student Engagement Using the Flipped Classroom," *Journal of Nutrition Education and Behavior*, 47(1), 109–114, 2014.
[14] M. Guzdial, "A Media Computation Course for Non-Majors," *ITiCSE annual conference on Innovation and technology in computer science education*, pp. 104-108, 2003.
[15] D. Hendrix, L. Myneni, H. Narayanan, M. Ross, "Implementing studio-based learning in CS2," *ACM technical symposium on Computer science education*, 2010.
[16] C. Hundhausen, 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.
[17] A.N. Kumar, "Using Online Tutors for Learning – What do Students Think?," *Frontiers in Education*, 2004.
[18] L. Layman, L. Williams, K. Slaten, "Note to self: make assignments more meaningful," *SIGCSE technical symposium on Computer science education*, pp. 459-463, 2007.
[19] C.B. Lee, 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.
[20] S. Leutenegger and J. Edgington, "A Games First Approach to Teaching Introductory Programming," *SIGCSE technical symposium on Computer science education*, 2007.
[21] H.N. Mok, "Teaching Tip: The Flipped Classroom," *Journal of Information Systems Education*, 25(1), 7-11, 2014.
[22] V. Norman and J. Adams, "Improving Non-CS Major Performance in CS1," *SIGCSE Technical Symposium on Computer Science Education*, pp. 558-562, 2015.
[23] Pearson: MyProgrammingLab. https://www.pearsonmylabandmastering.com/myprogramminglab/. August 2017.
[24] L. Porter, 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.
[25] L. Porter, 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.
[26] L. Porter and B. Simon, "Retaining nearly one-third more majors with a trio of instructional best practices in CS1," *SIGCSE technical symposium on Comp. science edu.*, pp. 165-170, 2013.
[27] Problets. http://problets.org/. August 2017.
[28] F.J. Rodríguez, 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.
[29] A. Roehl, "The Flipped Classroom: An Opportunity To Engage Millennial Students Through Active Learning Strategies," *Journal of Family and Consumer Sciences*, 2013.
[30] B. Simon, 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.
[31] L. Soh, "Using Game Days to Teach a Multiagent System Class," *SIGCSE technical symposium on Computer science education*, pp. 219-223, 2004.
[32] Turingscraft: CodeLab. https://www.turingscraft.com/. August 2017.
[33] C. Wilcox, "Testing Strategies for the Automated Grading of Student Programs," *ACM Technical Symposium on Computing Science Education*, pp. 437-442, 2016.
[34] zyBooks. http://www.zybooks.com/. August 2017