Learner Autonomy as a Means to Improve Pass Rates Among First-year Computing Students

DANIEL T. FOKUM, DANIEL N. COORE, AND CURTIS BUSBY-EARLE

Abstract

The worldwide pass rate for a first year tertiary-level Computing course (CS1) has been found to be about 67 per cent. At The University of the West Indies, Mona, the Semester I pass rate for the very first tertiary level computing course has ranged between 37.6 per cent and 66.7 per cent between the 2011/2012 and 2013/2014 academic years. Previous work from Ireland has shown that self-regulated learning, a concept related to learning autonomy, is important in learning how to program. In a bid to improve pass rates, one of the authors has used an online programming lab to facilitate students practising programming on their own. This paper documents the results of that experiment, as well as other techniques that have been used worldwide to improve performance in CS1.

Introduction

THE WORLDWIDE PASS RATE FOR A FIRST-YEAR TERTIARY level computing course (CS1) has been found to be about 67 per cent (Bennedsen and Caspersen 2007; Watson and Li 2014). At The University of the West Indies (UWI), Mona, the semester I pass rate for the very first tertiary level computing course has ranged between 37.6 per cent and 66.7 per cent between the 2011/2012 and 2013/2014 academic years. In a bid to improve pass rates, one of the authors has used an online programming lab, called MyProgrammingLab, developed by a textbook publisher to facilitate students practising programming on their own. This intervention was done in conjunction with traditional face-to-face methods of teaching programming, including the use of tutorials and laboratory exercises that reinforce the concepts taught in lectures.

Computer programming, like music, is something that is learned through practice. Papert (1980) proposes that computer programming is best learned through a constructionist approach. This means that after a teacher has conveyed a concept via lecture, the student must build a mental model of that concept, experiment with it and make observations, and adapt the model until he/she is able to solve novel problems with it before he/she becomes competent at it. This is an example of self-regulated learning. So a student who becomes a competent programmer must engage in self-regulated learning. In our context, the tutorials and laboratory exercises assigned in the very first tertiary level computing course are typically short bounded problems that get students to learn to think algorithmically through practice. Thus, the tutorials and labs are structured mechanisms for students to engage in self-regulated learning. We do not expect that merely working on the tutorials and lab exercises is sufficient to turn the students into competent programmers. Thus, we encourage students to engage in programming exercises outside of those that we assign to them.

Ideally, when autonomous learning is implemented, the learner takes charge of his/her own learning (Thanasoulas 2000). Thanasoulas, citing (Benson and Voller 2014), described five different ways that the term "autonomous" has come to be used in the context of learning, the first of which was: "for situations in which learners study entirely on their own". Murray (2014) states that self-regulated learning and learning autonomy, although different concepts, share several characteristics in common to the extent that they are frequently used interchangeably.

Although the students in this study were not working entirely on their own, we regard their using software tools that help them to learn a programming language on their own as a degree of autonomous learning. Recognizing that our students need more autonomous learning in order to become competent programmers, one of the authors used an educational intervention wherein the students tackled short programming exercises that allowed them to experiment with concepts being conveyed in lectures. The programming exercises are tied to a particular topic and they are graded immediately. In answering the exercises, students get practice in learning the syntax and semantics of a programming language, which should in turn lead to better performance in a programming course (Pearson 2016). The objective of this paper is to measure the effect of this intervention at The UWI, Mona. Specifically to determine whether the intervention had any statistically significant impact on the performance of the students in the course.

Literature Review

Wilson and Shrock (2001) examined 12 factors that contribute to success in an introductory computer science course at a Midwestern university. Their study found that comfort level[1] and mathematical background had a positive influence on success, whereas attribution to luck had a negative influence on success (Wilson and Shrock 2001). Their study also found that a formal class in computing had a positive influence on success, whereas playing computer games had a negative influence on class grades (Wilson and Shrock 2001).

McGill (2012) examined the role that personal robots have on the motivation of students who are learning to program. McGill (2012) found that programming with robots captured the attention of students better than programming alone. However, McGill's study did not show any positive effect on motivation from using robots in an introductory programming course.

Hare (2013) documented classroom interventions to reduce failures in an introductory Computer Science course at a public urban university. The interventions included hiring peer student mentors to tutor students individually and on a group basis. The peer mentors also followed up with students who were absent from class. The intervention did not have its desired goal of reducing the failures, however, there were several lessons drawn regarding the factors leading to poor classroom performance. Some of those factors included (1) many of the students who ended up failing or withdrawing from the course also worked full-time; (2) students most in need of assistance were also those who were least likely to accept any; (3) requiring students to start early on an assignment, e.g., submitting a written algorithm days before the project was due, seemed to help students deal with procrastination (Hare 2013). As a result of the intervention, Hare's university also introduced MyProgrammingLab

(Pearson 2015), the same online programming lab that was used at UWI, Mona for the CS1 course.

Wilcox (2015) discusses the role of automation in an undergraduate computer science programme. As computer science enrolments in the USA have climbed, many programmes are using automation for grading of student work. Wilcox (2015) aims to determine (1) how automated tools positively or negatively affect learning; and (2) if the benefits of automation outweigh its costs. The aspects of automation considered by Wilcox are: automated grading, peer instruction, and online tutorials such as MyProgrammingLab. Wilcox found that by introducing automated grading, students are able to get immediate feedback on their submissions and so students are more likely to continue working on an assignment until the entire software test suite is passed. By Spring 2014, Wilcox was seeing an average of 4.11 submissions per student per programming assignment, which was up from 1.11 in Fall 2013. No statistically significant improvement was seen due to peer instruction. However, evaluation of the online tutorials revealed that students who used MyProgrammingLab generally got higher grades than those who were in a control group that did not use the tool. Furthermore, this difference was statistically significant. At Wilcox's institution the introductory programming course is delivered using two lectures, a peer instruction session, and two 50-minute labs per week for 15 weeks. In the peer instruction session students are assigned to groups of 3–4 for the duration of the semester, and in each session they work on an automated quiz. The online tutorial, MyProgrammingLab, is in addition to all of the other course activities. At UWI, Mona the introductory programming course is delivered with 3 lecture hours, 2 hours of tutorials, and 2 lab hours per week for six weeks. This introductory course is immediately followed by another with the same structure for the next six weeks of the semester.

Bergin, Reilly, and Traynor (2005) examined the role of self-regulated learning on performance in an introductory tertiary-level programming course. Self-regulated learners are persons who are metacognitively, motivationally, and behaviourally active in their own learning. By using a questionnaire designed to measure strategies for learning, Bergin et al. (2005) report that self-regulated learning is important in learning how to programme. This is consistent with the view that programming is best learned with a constructionist approach.

Failure rates in introductory programming courses were examined by Bennedsen and Caspersen (2007). A worldwide survey found that about 67 per cent of students pass introductory programming courses, with pass rates of 30%–50% not uncommon (Bennedsen and Caspersen 2007). A follow-up study found a nearly identical worldwide pass rate of 67.7 per cent (Watson and Li 2014).

This section has presented literature that shows factors associated with success in Computer Science and discussed various interventions to improve performance in Computing. In two cases the literature has indicated that students who use an online programming lab have better performance. In the next section we present our methodology.

Methodology

The data in this paper comes from two main sources. From MyProgrammingLab we downloaded a list of all students who took COMP1126 and COMP1127 in semester I 2014/2015. This list contained student IDs and whether each programming assignment was completed correctly on time, completed correctly late, incorrect, or missing. Thus, for each student we could report on how many exercises were completed, incorrect, or missing. From UWI, Mona's grade upload system we retrieved student IDs and their final grades for both COMP1126 and COMP1127. A Microsoft Access query was written to combine the two data files, and output files were written for both COMP1126 and COMP1127. The output files were then imported into R (R Core Team 2014), a statistical environment, and additional analysis was done.

Measurement of impact

To measure the effect of the intervention, we used Pearson's $\chi^2$2-test. To do this we computed the expected proportions of the grades in COMP1126 and COMP1127 across five categories: **A, B, C, D, Non Pass**, based on the distributions obtained between 2011 and 2013, inclusive. We then examined the actual distribution across the same five categories, for the same subjects in 2014 when the autonomous learning intervention was done, and compared them against the expected proportions, using Pearson's χ^2-test (with four degrees of freedom).

Measurement of association

We also examined whether there were any notable associations between the discipline with which students engaged in the autonomous tool and their overall performance. We used the outcomes of their auto-generated assessments as a representation of that discipline. To test the strength of association, we carried out Pearson's χ^2 test of independence. Pearson's χ^2 test of independence is quite similar to Pearson's goodness of fit test, except that the degrees of freedom is now $(R-1)\times(C-1)$, where R and C are the number of rows and columns, respectively, in the contingency table.

Results

Effect of online programming lab on performance

The list below summarizes the null hypotheses that we considered for our study:

1. H_{01}: The observed distribution of A, B, C, D, and non-passing grades after the intervention follows the same distribution of grades before the intervention
2. H_{02}: Letter grade and the percentage of completed exercises are independent
3. H_{03}: Letter grade and the percentage of incorrect online programming lab exercises are independent
4. H_{04}: Letter grade and the percentage of missing (incomplete) online programming lab exercises are independent.

Table 1 summarizes the enrolment and the percentage of A, B, C, D, and non-passing grades from 2011/2012 Semester I to 2014/2015 Semester I for COMP1126. We observed that the number of students attempting the course increased steadily from 2012/2013, while the grade distribution varied from year to year.

Table 1: Performance in COMP1126 Semester I, 2011–2014

Semester	# Attempted	% A	% B	% C	% D	% Non pass
2011/12 Sem. I	259	33.6	15.8	6.6	0.4	43.6
2012/13 Sem. I	261	42.1	18.8	5.7	0.0	33.0
2013/14 Sem. I	277	16.2	13.4	8.3	0.0	62.1
2014/15 Sem. I	336	24.7	29.5	4.2	0.0	41.7

Our next goal was to determine if the intervention, i.e., use of the online programming lab, made a difference with the grade distribution. To do this we computed the average percentage of As, Bs, Cs, Ds, and non-passing grades for all semester I offerings of COMP1126 from 2011/2012 through 2013/2014. While COMP1126 is offered in both semesters I and II, we only chose semester I offerings because the performance in the second semester is frequently quite different from that observed in semester I. Our goal was to carry out a chi-squared goodness of fit test to determine if there was a statistically significant difference in the distribution of grades after the intervention. Table 2 summarizes the input for our analysis. From table 2 we conclude that there was no statistically significant difference in the grade distribution after the intervention, since $0.143 \leq 9.488 = \chi^2_{0.05}(4)$. Thus, we conclude that H_{01} is true.

Table 3 summarizes the enrolment and the percentage of As, Bs, Cs, Ds, and non-passing grades from 2011/2012 semester I to 2014/2015 semester I for COMP1127. As was the case for COMP1126, the number of students attempt-

Table 2: COMP1126 Grade Distribution in 2014/2015 Semester I, Compared to Distribution 2011/2012–2013/2014

	% A	% B	% C	% D	% Non pass	χ^2
2014/15 Sem. I Observed	24.7	29.5	4.2	0.0	41.7	0.143
Expected	30.4	15.9	6.9	0.0	46.7	

Table 3: Performance in COMP1127 Semester I, 2011–2014

Semester	# Attempted	% A	% B	% C	% D	% Non pass
2011/12 Sem. I	254	40.2	15.4	3.5	0.8	38.2
2012/13 Sem. I	260	54.2	10.8	3.1	0.8	31.2
2013/14 Sem. I	277	21.3	16.2	11.6	1.4	49.5
2014/15 Sem. I	337	26.1	35.6	8.9	0.0	29.4

ing COMP1127 increased steadily from 2012/2013, while the grade distribution varied from year to year. The grade distribution for COMP1127 was marginally better than that for COMP1126, with more students passing COMP1126, with better grades. This is due to the students gaining more competence with programming as the semester progressed.

Next, we determined if the intervention made a difference with the grade distribution. We computed the average percentage of As, Bs, Cs, Ds, and non-passing grades for all semester I offerings of COMP1127 from 2011/2012 through 2013/2014. We only chose semester I offerings because the performance in semester II is quite different from that observed in semester I. A chi-squared goodness of fit test was carried out to determine if there was a statistically significant difference in the distribution of grades after the intervention. Table 4 summarizes the input for our analysis. From table 4 we concluded that there was no statistically significant difference in the grade distribution after the intervention, since $0.415 \leq 9.488 = \chi^2_{0.05}(4)$. Thus, we conclude that H_{01} is true.

Table 4: COMP1127 Grade Distribution in 2014/2015 Semester I, Compared to Distribution 2011/2012–2013/2014

	% A	% B	% C	% D	% Non pass	χ^2
2014/15 Sem. I Observed	26.1	35.6	8.9	0.0	29.4	0.415
Expected	38.2	14.2	6.2	1.0	40.5	

Table 5: Observed COMP1126 Grade Distribution for Given Percentage of Correct Online Lab Exercises

	Per Cent Online Labs Correct				
Course grade	[0, 40%)	[40%, 60%)	[60%, 80%)	[80%, 100%]	*p*-value
A	8.0%	1.3%	8.0%	82.7%	0.000
B	11.9%	7.1%	21.4%	59.5%	
C	40.0%	40.0%	0.0%	20.0%	
F	23.9%	9.1%	10.2%	56.8%	

Next we evaluated whether there was statistical association between the final letter grades in COMP1126 and COMP1127 and (1) the percentage of correctly answered online programming lab exercises; (2) the percentage of incorrectly answered online programming lab exercises; and (3) the percentage of missing, i.e., incomplete, online programming lab exercises. To test the statistical association we carried out a chi-squared test of independence. Table 5 shows the observed COMP1126 grade distribution in semester I, 2014/2015 for different percentages of correctly answered online programming lab exercises. It is quite clear that students who got higher grades in COMP1126 tended to answer more online programming lab exercises correctly. The chi-squared test of independence returned a *p*-value of 0.000, which shows that H_{02} should be rejected in favour of the alternative hypothesis, that there is dependence between the letter grade and the percentage of correctly answered online programming lab exercises. Similarly, table 6 shows that there is dependence between the COMP1127 grade and the percentage of correctly answered lab exercises.

Table 6: Observed COMP1127 Grade Distribution for Given Percentage of Correct Online Lab Exercises

	Per Cent Online Labs Correct				
Course grade	[0, 40%)	[40%, 60%)	[60%, 80%)	[80%, 100%]	*p*-value
A	2.4%	1.2%	4.8%	91.6%	0.000
B	2.2%	12.1%	12.1%	73.6%	
C	16.7%	11.1%	27.8%	44.4%	
F	28.8%	15.4%	7.7%	48.1%	

Tables 7 and 8 are contingency tables showing the observed COMP1126 and COMP1127 grade distributions, respectively, in semester I 2014/2015 for different percentages of incorrectly answered online programming lab exercises. In both cases, the *p*-value is less than 0.05, meaning that we reject H_{03} in favour of the alternative hypothesis that there is statistical association between the grade and the percentage of incorrectly answered lab exercises.

Table 7: Observed COMP1126 Grade Distribution for Given Percentage of Incorrect Online Lab Exercises

	Per Cent Online Labs Incorrect				
Course Grade	[0, 2.5%)	[2.5%, 5.0%)	[5.0%, 10%)	[10%, 100%]	*p*-value
A	73.3%	20.0%	4.0%	2.7%	0.035
B	61.9%	13.1%	21.4%	3.6%	
C	60.0%	0.0%	40.0%	0.0%	
F	71.6%	17.0%	8.0%	3.4%	

Table 8: Observed COMP1127 Grade Distribution for Given Percentage of Incorrect Online Lab Exercises

	Per Cent Online Labs Incorrect				
Course Grade	[0, 2.5%)	[2.5%, 5.0%)	[5.0%, 10%)	[10%, 100%]	*p*-value
A	74.7%	18.1%	4.8%	2.4%	0.001
B	62.6%	14.3%	22.0%	1.1%	
C	44.4%	27.8%	11.1%	16.7%	
F	75.0%	13.5%	7.7%	3.8%	

Finally, tables 9 and 10 are contingency tables showing the observed COMP1126 and COMP1127 grade distributions, respectively, in semester I 2014/2015 for different percentages of missing online programming lab

Table 9: Observed COMP1126 Grade Distribution for Given Percentage of Missing Online Lab Exercises

	Per Cent Online Labs Missing				
Course Grade	[0, 10%)	[10%, 20%)	[20%, 30%)	[30%, 100%]	*p*-value
A	73.3%	14.7%	2.7%	9.3%	0.000
B	50.0%	23.8%	7.1%	19.0%	
C	20.0%	0.0%	0.0%	80.0%	
F	43.2%	17.0%	5.7%	34.1%	

Table 10: Observed COMP1127 Grade Distribution for Given Percentage of Missing Online Lab Exercises

	Per Cent Online Labs Missing				
Course Grade	[0, 10%)	[10%, 20%)	[20%, 30%)	[30%, 100%]	*p*-value
A	77.1%	15.7%	1.2%	6.0%	0.000
B	51.6%	23.1%	5.5%	19.8%	
C	27.8%	16.7%	22.2%	33.3%	
F	34.6%	13.5%	5.8%	46.2%	

exercises. It is clear that students who have higher grades have a smaller percentage of missing online programming lab exercises. In both cases, the *p*-value is less than 0.05 meaning that we reject H_{04} in favour of the alternative hypothesis that there is statistical association between the grade and the percentage of missing lab exercises.

Table 11 shows the mean marks obtained on each question of the COMP1126 examination from 2011/2012 semester I through 2014/2015 semester I. There has always been a maximum of 60 available marks on the COMP1126 examination. From 2011/2012 through 2013/2014, question 1 (Q1) of the examination, which had 20 multiple-choice sub-questions had a

Table 11: Observed Mean Marks in COMP1126 from 2011/12 Semester I through 2014/15 Semester 1

Semester	Q1	Q2	Q3	Exam Total	Coursework	Total
2011/12 Sem. I	13.2	7.7	7.2	27.4	31.5	58.2
2012/13 Sem. I	11.8	7.4	4.3	27.9	32.3	59.3
2013/14 Sem. I	11.6	5.0	5.0	21.6	26.4	48.1
2014/15 Sem. I	16.9	4.9	6.2	32.5	28.6	61.1

maximum of 20 points. In 2014/2015 Q1 had a maximum of 30 points. Analysis to test the equality of means for the Q1 scores shows that F=1.211<161.4=$F_{0.05}$(1,1), p=0.386. Thus, we fail to reject Ho, and conclude that the data do not provide convincing evidence that one pair of mean Q1 scores are different from each other.

Table 12 shows the mean marks scored on each question of the COMP1127 examination between 2011/2012 semester I and 2014/2015 semester I. The COMP1127 examination is structured very similarly to the COMP1126 with three equally weighted questions, except for 2014/2015 semester I, when the weights of the questions became 30, 15, and 15, respectively. Analysis to test the equality of means for the Q1 scores shows that F=17.637<161.4=$F_{0.05}$(1,1)and p=0.149. Thus, we fail to reject Ho, and conclude that the data do not provide convincing evidence that one pair of mean Q1 scores are different from each other.

Table 12: Observed Mean Marks in COMP1127 from 2011/12 Semester I through 2014/15 Semester 1

Semester	Q1	Q2	Q3	Exam Total	Coursework	Total
2011/12 Sem. I	13.5	9.5	8.2	29.6	31.2	60.8
2012/13 Sem. I	13.0	9.4	9.0	30.9	32.0	62.8
2013/14 Sem. I	11.6	3.8	8.9	24.4	23.0	47.4
2014/15 Sem. I	18.3	7.8	6.5	36.4	24.5	60.9

Student performance in first-year courses

Prior to the 2014/15 academic year, UWI required that students achieve a grade of D (40%) or better in order to progress in a programme. Figures 1 and 2 show the percentage of students achieving a pass in the first year courses between 2004/2005 semester I and 2013/2014 semester I. The average pass rate over this window is approximately 62.5 per cent. The data show a small overall decrease in the pass rate for the first computing course (either CS11A, COMP1125 or COMP1126 then COMP1127) in the Department of Computing (DoC). With CS11A and CS11B, students were not required to pass CS11A before attempting CS11B. However, the pass rate for CS11B was generally marginally higher, or about the same, as that for CS11A. This was most likely due to the students gaining greater proficiency with programming as time progressed. Since the semester I enrolment for CS11A was approximately the same as that in semester II for CS11B it is unlikely that many students were dropping out of the introductory course sequence on the basis of their CS11A grades.

Beginning with 2008/2009 semester I, students were expected to pass COMP1125 before progressing to COMP1160. What this meant is that the pass rate for COMP1160 became much higher than that for COMP1125, except for semester I of 2010/2011. This point is obviously an outlier and can be ignored, with one additional observation: the normal sequence called for COMP1160 to be attempted in semester II. As a result, students who attempted COMP1160 in semester I tended to be those who had failed either COMP1125 or COMP1160 at least once. Therefore, they were more likely to be weaker than the rest of the students taking computing courses.

From 2011/2012 semester I, the DoC has seen a minor drop in the pass rates for COMP1126 and COMP1127 as well as for COMP1161. As was observed with the 2008/2009–2010/2011 curriculum the first courses in the sequence have a lower pass rate than COMP1161, the next course in the sequence. COMP1210 has seen pass rates of under 59 per cent in every semester since its introduction. This course, which has only CSEC Mathematics as a prerequisite, was supposed to strengthen students' mathematical ability and prepare them for further study in computing. It is possible that this low pass rate is representative of poor mathematical ability amongst incoming students. The high pass rate for COMP1220 as shown in figures 1 and 2 is not inconsistent with this observation because COMP1220 does not involve much programming or application of mathematics.

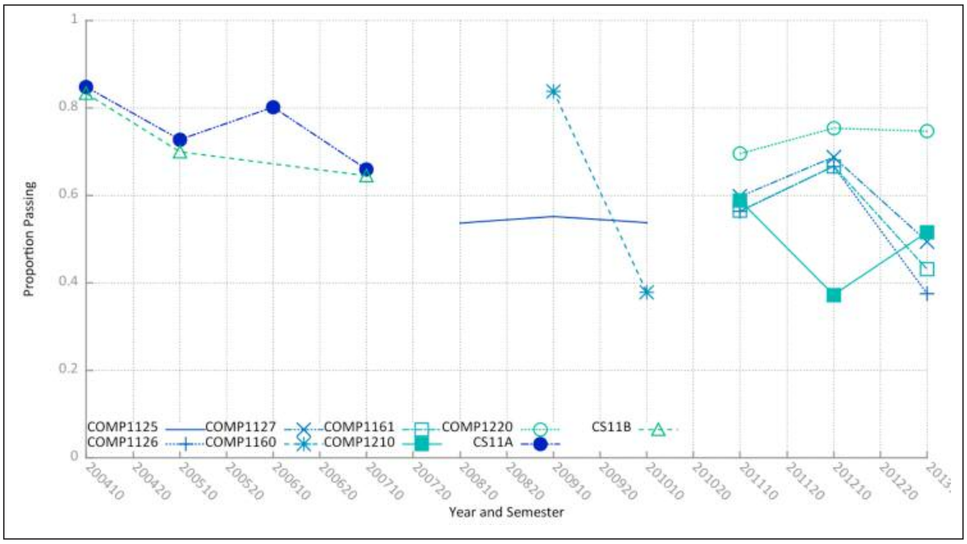

Figure 1: Performance in 1st Year Computing Courses from 2004/2005 to 2013/2014 (Semester I only)

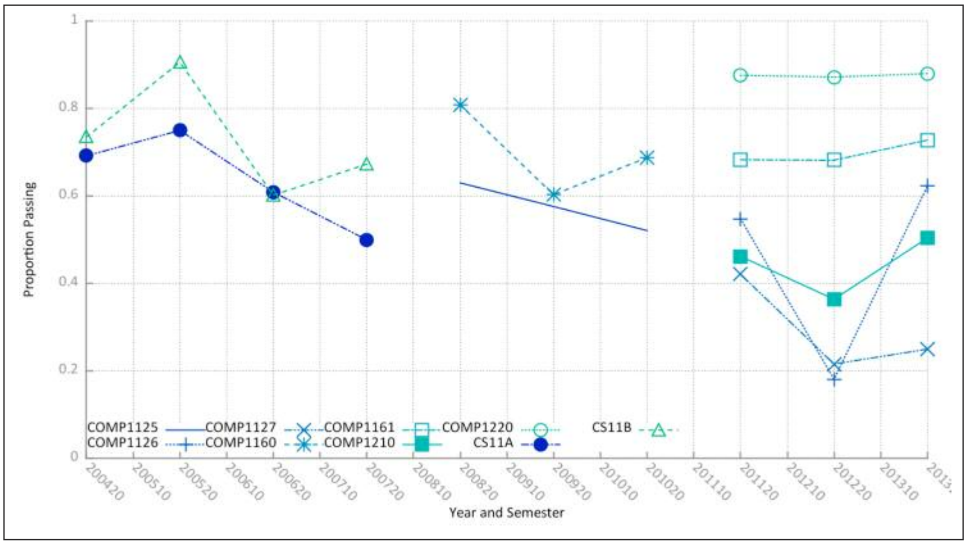

Figure 2: Performance in 1st Year Computing Courses from 2004/2005 to 2013/2014 (Semester II only)

Beginning in 2014/2015 semester I, UWI introduced a new grading scheme that saw the minimum pass mark move from 40% to 50%. This change was motivated by a need to move assessment from marking based on raw scores to one based on defined competencies. Assessment in the DoC has long been based on defined competencies, as a result, students' marks reflect their ability to analyze, design, and implement solutions to computational problems.

Figures 3 and 4 show the percentage of students in each course who achieved at least 50%. On average, about 55 per cent of students achieved a pass mark of 50% or greater. Figures 3 and 4 show that between 2004/2005 semester I and 2007/2008 semester II, there was an overall decline in the percentage of students who achieved grades of over 50%. The data also show that except for one semester, students performed much better in COMP1160 than in COMP1125. The one outlier can be explained as discussed above.

Since 2011/2012 semester I there has been a lot of variation in the percentage of students achieving marks of 50% or more. A few points that can be extracted from the data include:

1. Most COMP1220 students score 50% or more each semester
2. Students who take COMP1126 and COMP1127 in semester I perform much better than those who do so in semester II

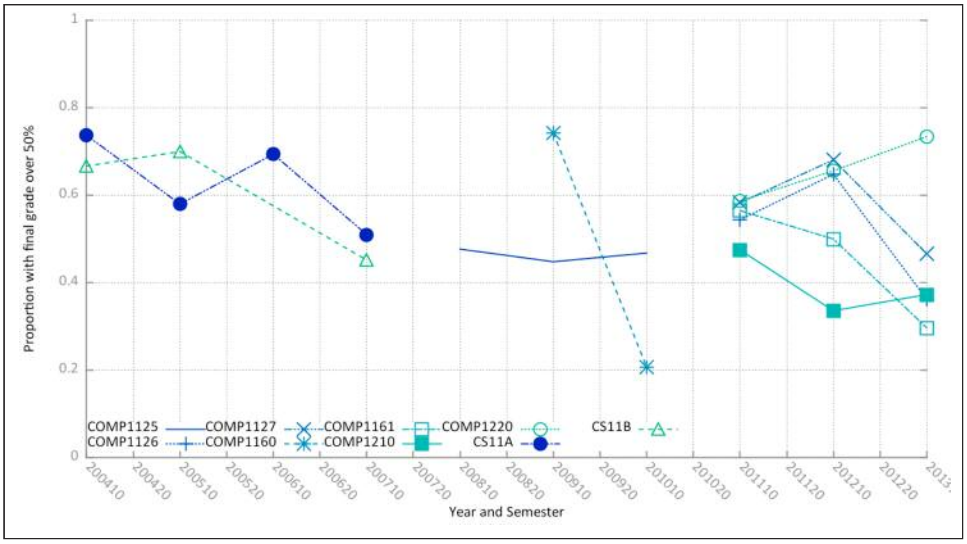

Figure 3: Proportion of Students Achieving 50% or more in 1st Year Computing Courses from 2004/2005 to 2013/2014 (Semester I only)

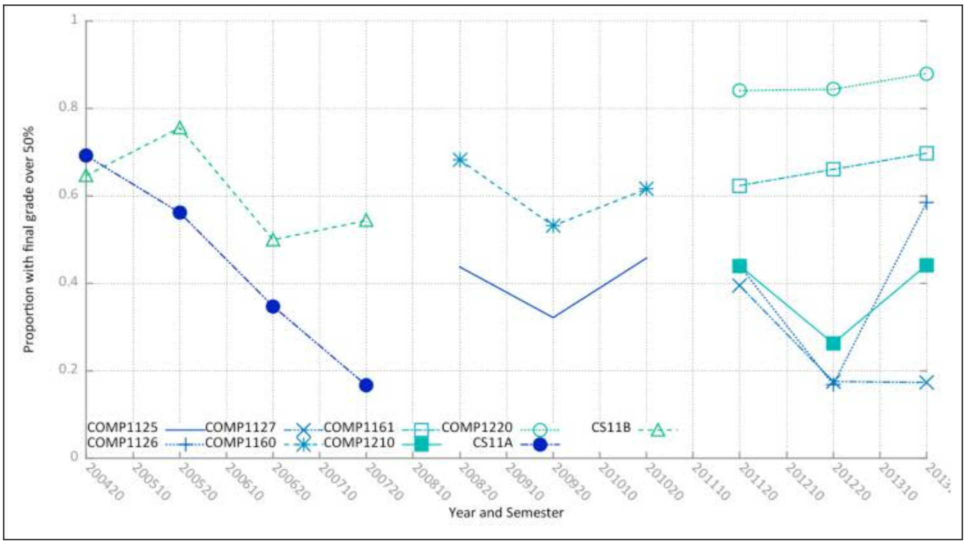

Figure 4: Proportion of Students Achieving 50% or more in 1st Year Computing Courses from 2004/2005 to 2013/2014 (Semester II only)

3. Students who take COMP1161 in semester II perform much better than those who take COMP1161 in semester I
4. The percentage of students achieving 50% or greater in COMP1210 has seen an overall decline since 2011/2012 semester.

Finally, figures 3 and 4 show that beginning in 2011/2012 semester I, the average rate of students achieving marks of 50% or more for the first-year courses offered by the DoC has ranged between 37 per cent for COMP1210 and 73 per cent for COMP1220. This indicates that, barring a change in student effort or a simplification of the assessment, more students will fail courses once UWI's new GPA scheme is introduced.

Discussion

Although previous studies (Wilcox 2015) had shown that MyProgrammingLab was expected to have a measurable impact on student performance, we did not detect that in our own experiments. This is not necessarily contradictory. First of all, the degree to which the intervention was measured to have a difference on student performance may not have been sufficient to be reflected as a change in overall grade distribution. The range of proportions of students attaining each grade level varied significantly over the four years that were used to set the expected proportions, and that would have raised the threshold of impact of the intervention to be detectable by the Pearson χ^2-test.

Feedback from the students who used the on-line programming environment, MyProgrammingLab, indicated that it was helpful in improving their understanding of the Python programming language through the opportunity it provided in allowing them to practise at their convenience, with immediate feedback. From interaction with the students, during lectures, a fundamental problem still exists: the ability to formulate algorithms to solve the small, well-defined problems that are presented to them. This is difficult to address for two apparent reasons: firstly, as the COMP1126 course is six weeks in duration, the time is inadequate to guide them in developing that skill. Secondly, over the (nearly) ten years that one of the authors has been involved with this course in its various iterations, the ability to think in a manner that supports the development of such algorithms is not something that comes naturally to many and in fact, is rather difficult to develop.

Conclusion

Although the MyProgrammingLab intervention probably has the capacity to improve the assessed outcomes of student's learning, we did not detect any statistically significant difference in the outcomes for either COMP1126 or COMP1127 (semester I). To the extent that autonomous learning has been successfully applied to learning natural languages, we believe that in the form of its application described here, it probably had a similar impact on students learning the syntax of the programming language in question. However, the assessment of these first year courses goes beyond simply being able to read code and write code that expresses a given mathematical relationship. It also requires students to be able to first formulate the necessary steps to solving a problem, then to express those steps in the programming language in a manner that combines productively to solve the overall problem. Reading and writing such code is not something that follows naturally from knowing the syntax of a programming language. It is our opinion that the root of the problem in performance may very well lie at the (general) inability of students to formulate those steps on their own, and not as much in expressing them in the programming language. Perhaps that is where the next autonomous learning intervention ought to be aimed, provided we can find the appropriate tools to do so.

Notes

1. Comfort level was a continuous variable computed from factors such as asking and answering questions in class/lab/office hours, anxiety level while working on computer assignments, perceived difficulty of the course, perceived understanding of course concepts compared to peers, and perceived difficulty of completing programming assignments (Wilson and Shrock 2001).

References

Bennedsen, J., and M. E. Caspersen. 2007. Failure rates in introductory programming. *SIGCSE Bulletin* 39 (2): 32–6. DOI: http://dx.doi.org/10.1145/1272848.1272879

Benson, P., and P. Voller. 2014. Introduction. In *Autonomy and independence in language learning*, ed. P. Benson and P. Fuller, 1–12. London and New York: Routledge.

Bergin, S., R. Reilly, and D.Traynor. 2005. Examining the role of self-regulated learning on introductory programming performance. *Proceedings of the First International Workshop on Computing Education Research (ICER '05)*, 81–6. DOI: http://dx.doi.org/10.1145/1089786.1089794

Hare, B. K. 2013. Classroom interventions to reduce failure and withdrawal in CS1: A field report. *Journal of Computing Sciences in Colleges* 28 (5): 228–35. DOI: http://dl.acm.org/citation.cfm?id=2458569.2458618

Holec, H. 1979. *Autonomy and foreign language learning*. ERIC. Strasbourg: Council for Cultural Cooperation.

McGill, M. M. 2012. Learning to program with personal robots: Influences on student motivation. *Transactions on Computing Education* 12 (1): 4:1–4:32. DOI: http://dx.doi.org/10.1145/2133797.2133801

Murray, G. 2014. The social dimensions of learner autonomy and self-regulated learning. *Studies in Self-Access Learning Journal* 5 (4): 320–41.

Papert, S. 1980. *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Pearson. 2015. MyProgrammingLab. Retrieved from http://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/

Pearson. 2016. MyProgrammingLab Tour Video Transcript. Retrieved from http://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/tour-video-transcript/index.html

R Core Team. 2014. R: *A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from http://www.R-project.org

Thanasoulas, D. 2000. What is learner autonomy and how can it be fostered? *The Internet TESL Journal* 6 (11), 37–48. Retrieved from http://iteslj.org/Articles/Thanasoulas-Autonomy.html

Watson, C., and F. W. B. Li. 2014. Failure rates in introductory programming revisited. *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*, 39–44. New York: ACM. DOI: http://dx.doi.org/10.1145/2591708.2591749

Wilcox, C. 2015. The role of automation in undergraduate computer science education. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)* (pp. 90–95). New York: ACM. DOI: http://dx.doi.org/10.1145/2676723.2677226

Wilson, B. C., and S. Shrock. 2001. Contributing to success in an introductory computer science course: A study of twelve factors. *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education (SIGCSE '01)*, 184–88. Charlotte, NC: ACM. DOI: http://dx.doi.org/10.1145/364447.364581