Writing Effective Autograded Exercises using Bloom's Taxonomy

Dr. Lina Battestilli, North Carolina State University

Lina Battestilli is Teaching Associate Professor of Computer Science at NC State University. She received her Ph.D. in Computer Science from NCSU in August 2005, her masters in Computer Networking in August 2002 also at NCSU and her BS in Electrical Engineering and Minor in Applied Mathematics from Kettering University in 1999.

Prior to joining North Carolina State University in 2012, Dr. Battestilli was a network research engineer at the Next Generation Computing Systems at IBM Research. She worked on the PowerEN Technology, a blur between general purpose and networking processors and hardware accelerators. She identified and studied workloads at the edge of the network that required high-throughput and fast deep-packet processing.

Since 2012, her research has been focused on Computer Science Education, especially in the area of peer collaboration, scaling techniques for large courses, auto-graders and learning analytics. She is also working on software that can be used for teaching and learning. She is investigating techniques and best practices on broadening participation in Computer Science. Women and minorities need to be more involved in tech innovation as companies and teams perform better when there is diversity.

She is also interested in Cloud Networking, Internet Of Things, Software Defined Networks and the design and performance evaluation of networking architectures and protocols, which are areas she worked in while in industry.

Ms. Sarah Korkes, North Carolina State University

Sarah Korkes is a recent graduate of North Carolina State University. She received her B.S. in Computer Science from NCSU in May 2020, and she also minored in Spanish. She is interested in improving Computer Science Education, and has been working in CS Education research since 2018.

Writing Effective Autograded Exercises using Bloom's Taxonomy

Abstract

Computer Science (CS) enrollment continues to grow every year and many CS instructors have turned to auto-graded exercises to ease grading load while still allowing students to practice concepts. As the use of autograders becomes more common, it is important that the exercise sets are being written to maximize student benefit. In this paper, we use Bloom's Taxonomy (BT) to create auto-graded exercise sets that scale up from lower to higher levels of complexity. We conducted a field experiment in an introductory programming course (264 students) and focused on evaluating learning efficiency, code quality, and student perception of their learning experience. We found that it takes students more submission attempts in the auto-grader when they are given BT Apply/Analyze-type questions that contain some starter code. Students complete the auto-graded assignments with fewer number of submissions when there is no-starter code and they have to write their solution from scratch, i.e. BT Create-type of questions. However, when writing code from scratch, the students' code quality can suffer because the students are not required to actually understand the concept being tested and might be able to find a workaround to pass the tests of the auto-grader.

Introduction

The number of undergraduates seeking Computer Science (CS) degrees has nearly doubled in recent years[1]. To manage large course enrollments, many CS instructors use automated grading tools. An Automated Grading Tool (AGT) is instructional software that provides students with instant feedback to submitted programming assignments. AGTs are beneficial to instructors because they can significantly reduce grading time[2]. Since AGTs have become so integral for introductory programming classes, it is important to find ways to maximize student learning and perception of their learning experience when using AGTs.

We use Bloom's Taxonomy (BT) to assign and arrange auto-graded exercises by their complexity. Bloom's Taxonomy is a widely used educational hierarchical model [3,4] that has influenced the organization of class activities in many educational fields including computer science [5,6].

In this study, we examined the impact of increasing auto-graded exercise complexity in relation to Bloom's Taxonomy (BT) from lower to higher orders within auto-graded exercise sets. We analyzed students' learning efficiency, code quality and perception of their learning experience. We performed a field experiment in a large introductory CS1 course that utilizes the web-based

AGT by MathWorks called MATLAB Grader[7]. We hypothesized that creating exercise set based on BT would allow the students to be more efficient and satisfied with their learning. To investigate that hypothesis, we focused on the following research questions:

- **RQ1:** Do exercise sets that increase in BT complexity help students complete them more efficiently?
- **RQ2:** How does the code quality of solutions compare when students are given exercise sets that increase in BT difficulty vs. exercise sets that are all at the highest level of BT?
- **RQ3:** How do students perceive their learning experience when exercise set difficulty increases based on BT?

Background

As **Auto-Grading Tools** (AGTs) have become an essential component in computer science education, they have also been subjected to numerous studies [8,9,10]. The bulk of these studies cover changes in student performance after the integration of an AGT as well as student perception of AGTs, and they cover a variety of tools.

There are numerous AGTs available for beginner programmers and instructors[11,12,13]. The AGT we utilize in this study is MATLAB Grader, an AGT that is designed for MATLAB, a programming language used mainly by engineers. MATLAB Grader is web-based and allows for instructors to write their own exercises and tests.

AGTs have multiple applications in computer science classrooms, and typically are used to allow the students to get extra problem solving practice. Common ways to use AGTs are for: an in-class active learning supplement[8,9], as a laboratory grading platform, and as assigned homework[8].

AGTs have been shown to benefit **student performance** in several regards. Courses that have implemented AGTs have experienced reduced dropout rates[10]. In the case of two Argentinian Universities, an early drop-out rate decreased from 28% to 14% and 58% to 35% respectively [8]. The improvements in student retention and passing rates were attributed to allowing students to learn at their own pace. Classrooms that have implemented AGTs in their coursework have also experienced an overall increase student grades [10,14]. In the case of Oregon University, the first semester that integrated CodeLab [9] saw the average class GPA rise from 2.0 to 2.2 on a 4-point grading scale.

With regard to **code quality**, novice programmers are prone to mistakes and more willing to leave them in their code [15]. Some studies have tried to implement changes with regards to static analysis tools [16] or level of feedback given [17]. Code quality is important because it is an important proxy for determining actual understanding and identifying stumbling blocks that students are still having, even if they are able to pass the auto-grader's tests.

In a large meta study [10] that analyzed AGTs in computer science education, they had deemed that student perception was ambiguous. However, in this meta study, many of the negative-opinion papers and some of the positive-opinion papers are over ten years old, and they therefore may contain outdated students' perceptions. Students who rated the AGT positively often praised the

opportunity to receive feedback before the final submission of the assignment [8,6], which encouraged them to make multiple attempts to strive for a higher grade. This increased exposure to the material may have positively impacted their overall understanding of the material as well. On the other hand, students who had rated AGTs negatively almost unanimously disliked the level of precision required from their programs [10]. AGTs penalize students who may have been close to the correct solution, leading the students to still receive a low grade [18]. Some students also showed skepticism of the grading without human intervention.

Bloom's Taxonomy is one of the most commonly used models to describe a learner's level of understanding of a topic based on cognitive domains. It has been effective in assisting instructors in various fields in structuring coursework, homework assignments, and assessments [19,5,6,20]. Bloom's Taxonomy (BT) has also been used in CS education to provide a way for instructors to accurately compare test question complexities across several topics [5] and to teach computational thinking[6].

We chose to use the newer Bloom's Taxonomy model, which has six levels. from lowest to highest: Remember, Understand, Apply, Analyze, Evaluate, and Create [4]. Table 1 shows the levels of Bloom's Taxonomy as they are mapped to programming problems, as we've defined them using ideas from [5,21,20,22,23].

BT Level	Action Skills	Problem Types	Suitability to AGT
Remember	Direct recall of Syntax	Multiple Choice	NO: Best tested via conceptual questions
Understand	Trace & Understand Provided Code	Fill-in-the-blank with scafolded code	NO: Best tested via conceptual questions
Analyze	Determine errors in code	Determine errors in code & Debug	YES: can be tested with short coding exercises
Evaluate	Write/Execute black box and white box unit tests	Determine if code meets requirements via tests	NO: best tested with qualitative questions, longer assignments, static doe analysis tools
Create	Creating one's own code solutions	Write code from scratch to meet requirements	YES: can be tested with short coding exercises

Table 1: Bloom's Taxonomy Mapping to CS1 Programming Questions

Auto-Graded exercises can be best written to fall into the *Apply*, *Analyze* and *Create*-types of BT questions. *Remember* and *Understand* type questions are more appropriate for other types of assessments such as quizzes/tests. Working on *Evaluate*-type problems is typically done in courses beyond CS1 where the students learn how to evaluate and test code based on requirements.

In this study, we combined these areas of research by extending the learning concepts proposed by the Bloom's Taxonomy to the domain of AGTs. To our knowledge, this is the first study that scales exercise set difficulty from the lower levels to the higher levels of Bloom's Taxonomy within AGTs. We study the effects on student learning efficiency, code quality of solutions, and student perceptions of their learning experience.

Methods

We ran a field experiment in a CS1 introductory course for non-majors at a large public university in the Spring of 2019 with 264 enrolled students. This course is primarily taken by undergraduate engineering students, and it is required for their degrees. The course covers typical CS1 topics such as variables, plotting data, conditionals, loops, functions, string manipulation, arrays, file I/O, etc. and the programming language is MATLAB. Students' learning is assessed via graded exams, projects, in-class problem-solving participation via clickers, homework assignments and exercise sets completed in Lab.

This study focuses on the Lab, where students complete a series of auto-graded exercises (i.e. "exercise set") on topics introduced within the last week in the course. Students were stratified into nine Lab sections, each composed of 16 to 35 students. Each Lab Section was assigned to either the *control* or the *treatment* group based on number of students and the time of day of the Lab. Four Lab sections were assigned to the control group, totaling 131 students, and five lab sections were assigned to the treatment group, totaling 133 students. From the control group 109 students consented to this research and in the treatment group 112 students consented.

Each Lab section met every week for two hours and 45 minutes and was led by two undergraduate teaching assistants (TAs). In the first half of Lab, the TAs review the weekly topics and provide worked-out programming examples. Next, the students work on completing independent auto-graded exercise sets in MATLAB Grader. We ran this field experiment in the week which covered iterations (for loops, while loops, and nested loops).

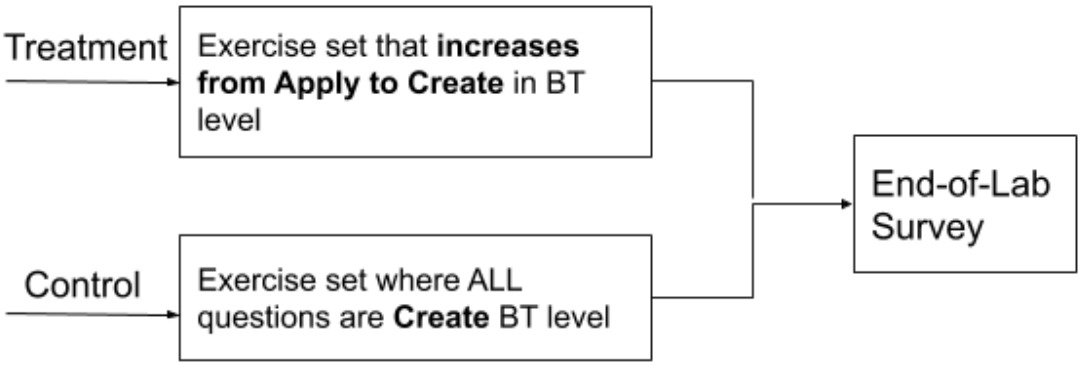


Figure 1: Timeline of the activities during the Lab

Figure 1 shows how Lab differed between the treatment and control groups. The control group worked on Create-type BT questions only, while the treatment group had questions that increased on the BT scale. At the end of Lab, both groups took an End-of-Lab survey that was used to gauge the students' perceptions of their learning.

Developing the Exercise Sets

Auto-Graded Platform: We created the auto-graded exercises using MATLAB Grader, a web-based auto-graded tool[7]. When using MATLAB Grader, the students write code and submit it to be evaluated for correctness via assessments, i.e. tests. Each exercise has a description and

possibly some starter code and/or starter variables. The student writes code to solve the exercise at hand and submits it for testing. In this study, the students were given an unlimited number of submissions so that they could continue to test their code without penalty.

From the MATLAB Grader tool, we had access to the following important data about the students performance: 1) Code the student wrote, 2) Time between the first and last submission for each student for each exercise, and 3) Number of submissions each student took on each exercise.

Bloom's Taxonomy Mapping: To create the questions for the exercise sets, we focused on three levels of Bloom's Taxonomy: Apply, Analyze and Create. We used [5,20] to help us map CS exercises to Bloom's Taxonomy. Isomorphic questions were used to compare the performance of the treatment and control groups. First, we wrote the exercise set for the treatment group and then reworked each non-Create type (Apply or Analyze) exercise into a Create-type exercise for the control group.

Exercise	**BT Level for Treatment**	**BT Level for Control**
Q1: FOR Loops 1	Apply	Create
Q2: FOR Loops 2	Analyze	Create
Q3: FOR Loops 2	Create	Create
Q4: WHILE Loops 1	Apply	Create
Q5: WHILE Loops 2	Analyze	Create
Q6: WHILE Loops 3	Create	Create
Q7: Nested Loops 1	Apply	Create
Q8: Nested Loops 2	Create	Create

Table 2: Progression on Bloom's Taxonomy in Exercise Sets

Table 2 shows the order of the exercises given in Lab, including what level they are on Bloom's Taxonomy. Both the treatment and control groups were assigned 8 questions. Per topic, the treatment group got questions from the lower to higher levels of BT, i.e. Apply to Analyze to Create. The questions for the control group were all written as Create-type questions. Questions 3, 6 and 8 were the same for both groups.

Examples of these questions can be seen in Table 3. The top left side of this table shows Q1 for the treatment group which is an Apply-type Question. This means that the students were given some starter code and asked to fill in the blanks. The top right row of the table shows the same question re-written as a Create-type, this is Q1 for the control group. A Create-type question requires the students to write most of the code from scratch but may have some variables predefined. The second row shows Q5 Analyze-type questions for the treatment group and the corresponding Q5 Create-type question for the control group.

End-of-Lab Survey

After completing the auto-graded exercises, the students completed a survey, which was written using validated questions from[24,25]. We asked the students self-efficacy questions and questions about their perception of the auto-graded exercises.

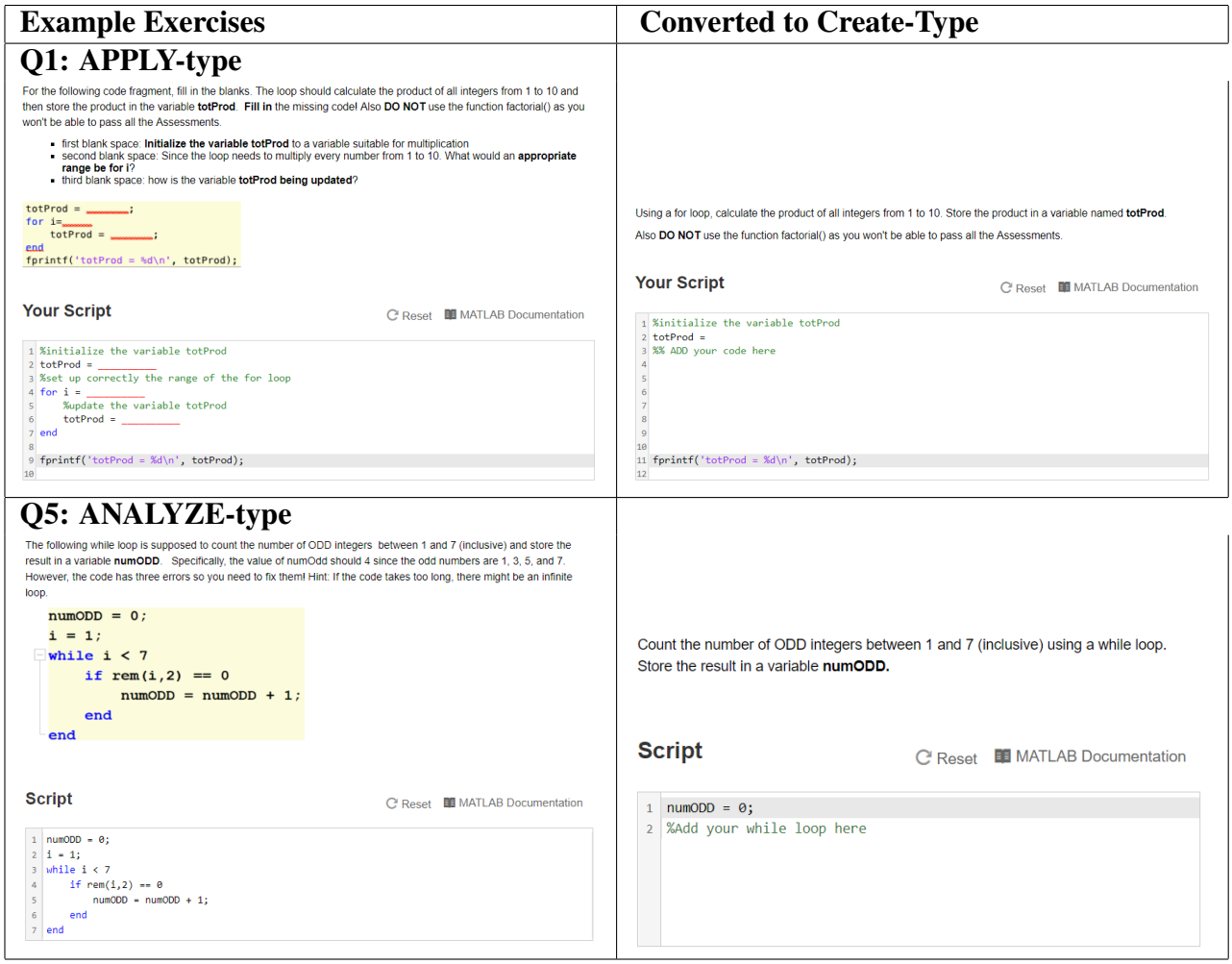


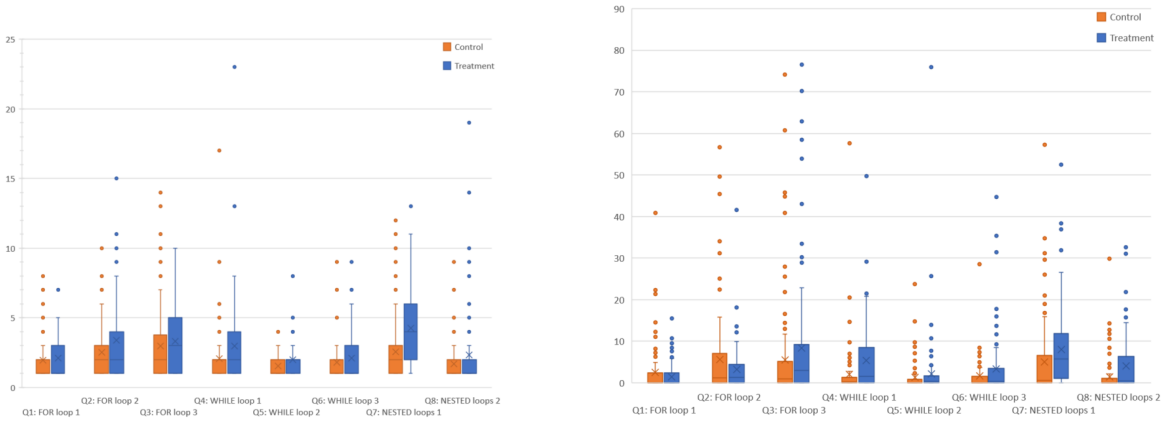
Table 3: Types of Auto-Graded Exercises

Results

Learning Efficiency (RQ1)

Figure 2a shows differences in the number of attempts on each question between the two groups through box-and-whisker plots with some outliers shown. The two groups followed similar trends in number of attempts, but the treatment group had slightly more submissions on most of the questions. A t-test was used which showed significant difference on Q2(p=0.006) , Q4 (p = 0.011), Q5(p=0.003) and Q7(p=1.61x10^{-6}) and Q8 (p=0.035). Recall from Table 2 that Q3, Q6 and Q8 were the same for both groups. The treatment group had more submissions on the Apply and Analyze-type questions but also struggled with Q8 which had to do with nested loops. This is opposite to our original hypothesis that the treatment group will complete the questions more quickly and efficiently.

Figure 2b shows the difference between students' first and last submission on each question, since time spent on each question could not be directly measured. Except for Q2, the treatment group took slightly longer than the control group. T-test results show that the difference between



(a) Number of Submissions

(b) Time (in minutes) between a student's First and Last Attempt

Figure 2: Submission Details per Question

the groups was statistically significant for Q2(p=0.019), Q4(p=0.003), Q7(p=0.004) and Q8(p=0.04). So the treatment group had more submissions on the Apply and Analyze-type questions and also took longer to complete.

Code Quality (RQ2)

We also inspected the quality of the students solutions. The solutions for Analyze and Apply type exercises results in very similar solutions because of the starter code. As expected, the Create-type solutions varied more often. To examine more closely the solutions for Q5 (shown in Table 3), we present the MATLAB Grader Solution Map in Figures 3 and 4.

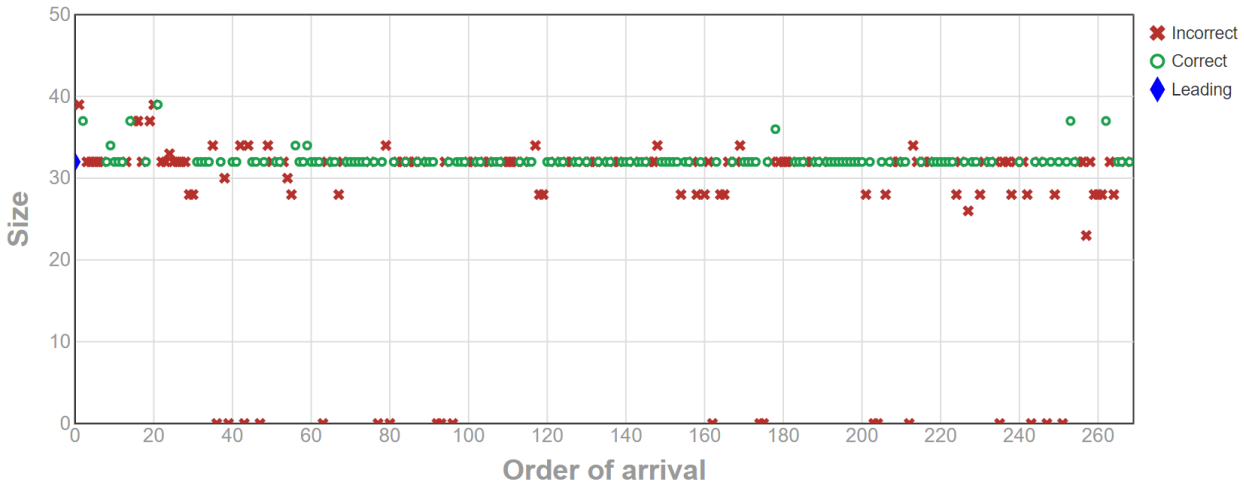


Figure 3: Student solutions from the Treatment Group for Q5: Analyze-type

As shown by the line at size 32 in Figure 3, the solutions from the treatment group were quite consistent. This is mostly due to the fact that the same bug-filled code was provided for the

students to debug. There is much more variance in the size of the student solution when they have to write the code from scratch.

For the control group, we found quite a bit more variation of the submitted code as shown in Figure 4. Common ”correct” student solutions are shown in Table 4. Some student solutions were clever and elegant but the majority of deviation resulted from hard-coded or solutions with unnecessary lines of code.

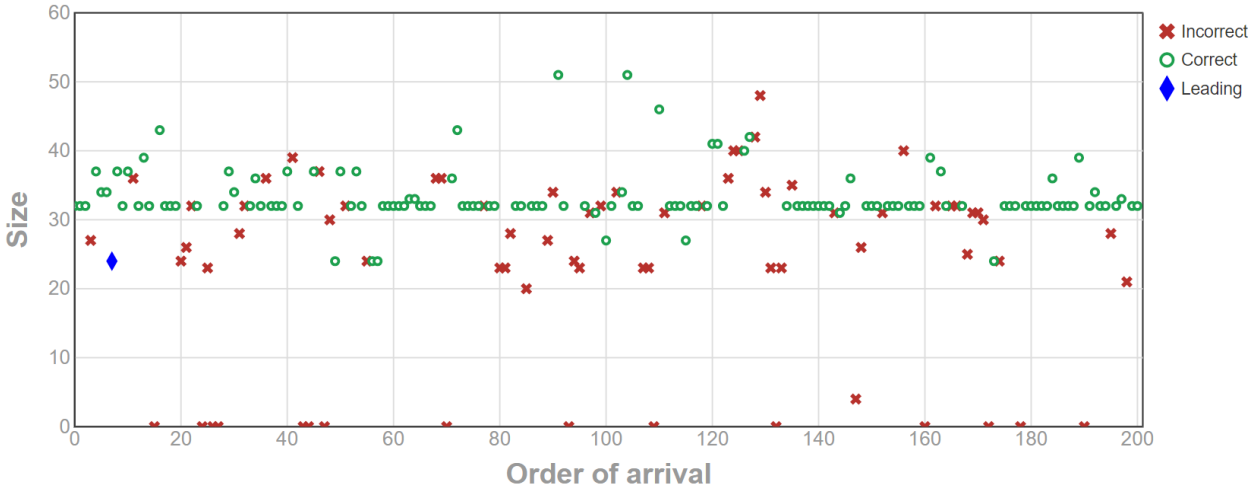


Figure 4: Student solutions from the Control Group for Q5: Create-Type

Student Perception (RQ3)

Perception results from the survey are shown in Figure 5. Both groups largely follow the same trend, with the mean response for the control group being 3.367 and the mean response for the treatment group being 3.357, so students in both groups, on average, agree with the statement that they are confident in their abilities as a MATLAB programmer.

Figure 6 shows a disparity between the two groups. Students in the control group were more likely to agree or strongly agree that they are capable of becoming a proficient programmer. This could be because it took them less submissions and less time to complete the exercises.

Discussion

We found that students complete auto-graded exercises faster if they write the code from scratch. Students take longer and have more submission attempts when they work on questions that require them to fill-in the blank or to debug code. This can be explained by the fact that novice programmers often struggle with reading and understanding code[26]. Further work needs to be done with regard to how this may affect actual understanding, especially since the code quality results showed that understanding varies wildly and does not necessarily correspond to the number of attempts taken to solve an exercise. In addition, the data for number of minutes between the first and the last attempt did not reflect a consistent trend of one group taking more time than the other. It is possible that these results are not the best metric of learning efficiency,

Category	Solution	Explanation

Expected	`numODD = 0;`
	`%Add your while loop here`
	`n = 1`
	`while n <= 7`
	`  if rem(n,2) ~=0`
	`  numODD = numODD + 1;`
	`  end`
	`  n = n + 1;`
	`end`
		This solution represents what we expected as a solution - a while loop that iterates through a range of numbers, checking if the current number is odd (and incrementing the numOdd counter if so). Then, the loop counter is incremented, and the loop runs again.

Clever	`numODD = 0;`
	`%Add your while loop here`
	`int = 1;`
	`while int <= 7`
	`  numODD = numODD + rem(int,2);`
	`  int = int+1;`
	`end`
		This solution still uses a check for odd/even, in a way that is more efficient. The sum is only incremented if the remainder is 1 (and thus, is odd).

Hard-coded	`numODD = 0;`
	`%Add your while loop here`
	`n = 1;`
	`while n <= 7`
	`  numODD = numODD + 1;`
	`  n = n + 2;`
	`end`
		We classified this solution as ”hard-coded” because it uses the fact that odd numbers are two apart instead of checking that a number is odd before another tally is added to the counter.

Extraneous code	`numODD = 0;`
	`%Add your while loop here`
	`n= 0`
	`while n<=7`
	`  n=n+1`
	`  if rem(n,2)~=0`
	`  numODD = numODD +1;`
	`  else`
	`  numODD= numODD;`
	`  end`
	`end`
		We classified this solution as ”extraneous” because it uses an extra else clause that is unnecessary.

Table 4: Sample student solutions on the Create-level version of Q5: While Loops 2 from the Control Group

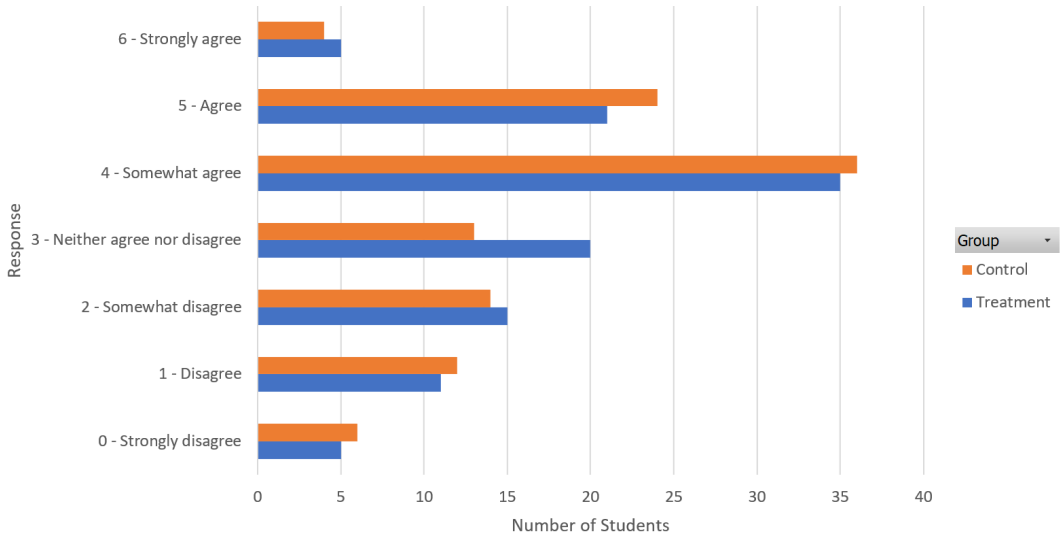


Figure 5: "I feel confident in my abilities as a MATLAB programmer". Control group: $mean = 3.367$, $s.d. = 1.579$; treatment group: $mean = 3.357$, $s.d. = 1.512$. Probability from a two-tailed t-test: $p = 0.962$

however, because a student who gets stuck on an exercise may skip to another one, thus inflating the time it took between the first and last attempt on the exercise they were stuck on.

When examining code quality, the variation in solutions in the control group revealed a troubling challenge. Although one might view the variation and freedom to find one's own solution as a good thing, the fact was that many of the solutions contained extraneous variables or clauses or were hard-coded. Having extraneous clauses or exploiting an assignment's purpose are not practices that instructors want to enforce, as they may lead to the students developing bad coding practices. In-depth style checks may be performed at the project level, for any instructor who desires to enforce style earlier and get their students in the habit of better practices, Create-level exercises may not be the best mechanism for instilling such practices, as they give too much freedom to novice programmers.

Regarding perception results, our intervention did not show big differences between the two groups in their confidence to become a programmer. However, the students in the treatment group felt less proficient. That might be because they had to work with the Analyze and Apply type of questions that took longer time to complete.

Threats to Validity

A limitation of this study is that during Lab, the students are allowed to talk to one another. Lab is usually run this way so we did not want to deviate much from the regular weekly experience in Lab. So, any patterns of doing well or not doing well on a question could be exacerbated by the fact that students may be working together on a problem and may submit the same solution, right or wrong. This could also affect number of attempts in that one student could submit a solution worked on by two students and wait to see if it is correct before the other student who contributed submits theirs. This would decrease, overall, the number of incorrect submissions.

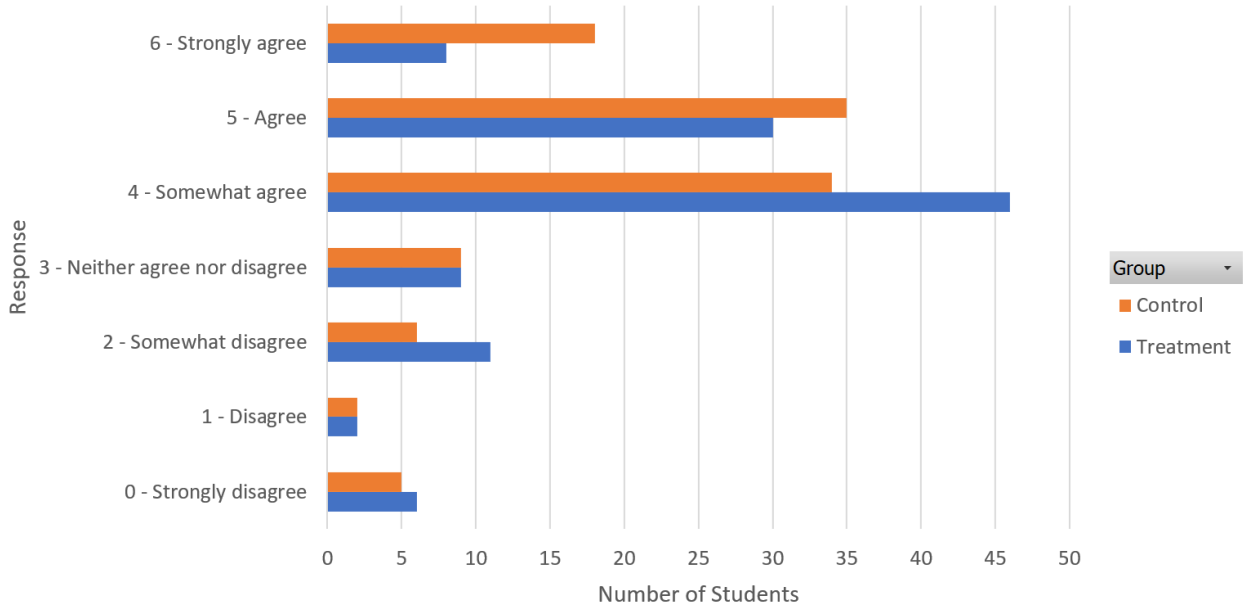


Figure 6: "I am capable of becoming a proficient programmer." Control group: $mean = 4.22$, $s.d. = 1.468$; treatment group: $mean = 3.866$, $s.d. = 1.43$. Probability from a two-tailed t-test: $p = 0.071$

Conclusion

In this study, we compared students on learning efficiency (number of submission on each exercise, time between first and last submission), code quality, and perception of their learning experience in an auto-grader. The treatment group of students was given an exercise set that increased in Bloom's Taxonomy level as students progressed through the set, and the control group of students were given an exercise set of all Create-level questions. We found that students who were given the exercises that scale up in Bloom's Taxonomy level took more attempts to solve their exercises than those given all Create-level questions. We conjecture that is because having to mold one's solution to someone else's requires first that you understand their solution, and that may take more than one attempt to achieve. On Create-level questions, we found that students often used the freedom they had to create a solution that was poorly styled or had other bad coding practices.

Further studies should examine these changes on student performance as well, and a more in-depth analysis with an automated tool needs to be conducted on how student code quality is impacted. Also, future studies could look at developing methods to better enforce code quality and good style practices in short exercises. In addition, future studies should confirm the Bloom's Taxonomy level of CS exercises before their use, and perhaps they should even aim to work with other instructors to create a bank of CS exercises and come to a consensus on how to map CS topics to BT.

References

[1] S. Zweben and B. Bizot. The taulbee survey. *Computing Research Association*, 2018. URL https://cra.org/resources/taulbee-survey/.

[2] Vincenzo Del Fatto, Gabriella Dodero, Rosella Gennari, Benjamin Gruber, Sven Helmer, and Guerriero Raimato. Automating assessment of exercises as means to decrease mooc teachers' efforts. In Óscar Mealha, Monica Divitini, and Matthias Rehm, editors, *Citizen, Territory and Technologies: Smart Learning Contexts and Practices*, pages 201–208, Cham, 2018. Springer International Publishing. ISBN 978-3-319-61322-2.

[3] B.S. Bloom, M.D. Engelhart, E.J. Furst, W.H. Hill, and D.R. Krathwohl. *Taxonomy of Educational Objectives Handbook 1: Cognitive Domain*. Pearson, 1956.

[4] Lorin Anderson, David R. Krathwohl, Peter W. Airasian, Kathleen A. Cruikshank, Richard E. Mayer, Paul R. Pintrich, James Raths, and Merlin C. Wittrock. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Pearson, 2000.

[5] M. Dorodchi, N. Dehbozorgi, and T. K. Frevert. "i wish i could rank my exam's challenge level!": An algorithm of bloom's taxonomy in teaching cs1. In *2017 IEEE Frontiers in Education Conference (FIE)*, pages 1–5, Oct 2017. doi: 10.1109/FIE.2017.8190523.

[6] Cynthia C. Selby. Relationships: Computational thinking, pedagogy of programming, and bloom's taxonomy. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, WiPSCE '15, pages 80–87, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3753-3. doi: 10.1145/2818314.2818315. URL http://doi.acm.org/10.1145/2818314.2818315.

[7] Matlab grader documentation. www.mathworks.com/help/matlabgrader/index.html.

[8] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J. Gomez. The effect of a web-based coding tool with automatic feedback on students' performance and perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 2–7, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5103-4. doi: 10.1145/3159450.3159579. URL http://doi.acm.org/10.1145/3159450.3159579.

[9] Valerie Barr and Deborah Trytten. Using turing's craft codelab to support cs1 students as they learn to program. *ACM Inroads*, 7(2):67–75, May 2016. ISSN 2153-2184. doi: 10.1145/2903724. URL http://doi.acm.org/10.1145/2903724.

[10] Raymond Scott Pettit, John D. Homer, Kayla Michelle McMurry, Nevan Simone, and Susan A. Mengel. Are automated assessment tools helpful in programming courses? In *2015 ASEE Annual Conference & Exposition*, number 10.18260/p.23569, Seattle, Washington, June 2015. ASEE Conferences. https://peer.asee.org/23569.

[11] N. Parlante. Codingbat. http://codingbat.com/java.

[12] G. Bowden and K. Maguire. Practice-it. https://practiceit.cs.washington.edu/.

[13] A. Singh, S. Karayev, I. Awwal, and P. Abbeel. Gradescope. https://gradescope.com/. Accessed August 2, 2018.

[14] Curtis Cohenour P.E. and Audra Lynn Hilterbran. Automated grading of access® databases using the matlab® database toolbox. In *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, June 2017. ASEE Conferences. https://peer.asee.org/27647.

[15] Hieke Keuning, Bastiaan Heeren, and Johan Jeuring. Code quality issues in student programs. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, pages 110–115, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4704-4. doi: 10.1145/3059009.3059061. URL http://doi.acm.org/10.1145/3059009.3059061.

[16] S. V. Yulianto and I. Liem. Automatic grader for programming assignment using source code analyzer. In *2014 International Conference on Data and Software Engineering (ICODSE)*, pages 1–4, Nov 2014. doi: 10.1109/ICODSE.2014.7062687.

[17] E. Araujo, D. Serey, and J. Figueiredo. Qualitative aspects of students' programs: Can we make them measurable? In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, Oct 2016. doi: 10.1109/FIE.2016.7757725.

[18] Vreda Pieterse and Janet Liebenberg. Automatic vs manual assessment of programming tasks. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, Koli Calling '17, pages 193–194, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5301-4. doi: 10.1145/3141880.3141912. URL http://doi.acm.org/10.1145/3141880.3141912.

[19] E. de Bruyn, E. Mostert, and A. van Schoor. Computer-based testing - the ideal tool to assess on the different levels of bloom's taxonomy. In *2011 14th International Conference on Interactive Collaborative Learning*, pages 444–449, Sep. 2011. doi: 10.1109/ICL.2011.6059623.

[20] Errol Thompson, Andrew Luxton-Reilly, Jacqueline L. Whalley, Minjie Hu, and Phil Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education - Volume 78*, ACE '08, pages 155–161, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc. ISBN 978-1-920682-59-0. URL http://dl.acm.org/citation.cfm?id=1379249.1379265.

[21] John T. Bell and H. Scott Fogler. The investigation and application of virtual reality as an educational tool. In *American Society for Engineering Education Annual Conference*, Jun 1995.

[22] D.I. Chatzopoulou and A.A. Economides. Adaptive assessment of student's knowledge in programming courses. *Journal of Computer Assisted Learning*, 26(4):258–269, 2010. doi: 10.1111/j.1365-2729.2010.00363.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2729.2010.00363.x.

[23] R. Heer. A model of learning objectives. 2012.

[24] Computing attitudes survey. http://stelar.edc.org/sites/stelar.edc.org/files/cas-v4.pdf.

[25] E.N. Wiebe, L. Williams, K. Yang, and C. Miller. Computer science attitude survey. Technical report, North Carolina State University Technical Report, 2003.

[26] Benjamin Xie, Greg L. Nelson, and Andrew J. Ko. An explicit strategy to scaffold novice program tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 344–349, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351034. doi: 10.1145/3159450.3159527. URL https://doi.org/10.1145/3159450.3159527.