2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #40146

First Try, No (Autograder) Warm Up: Motivating Quality Coding Submissions

Liia Butler, University of Illinois, Urbana-Champaign
Dr. Geoffrey L. Herman, University of Illinois, Urbana-Champaign

Dr. Geoffrey L. Herman is the Severns Teaching Associate Professor with the Department of Computer Science at the University of Illinois at Urbana-Champaign.

©American Society for Engineering Education, 2023

First Try, No (Autograder) Warm Up: Motivating Quality Coding Submissions

Abstract

Instructors face the challenge of encouraging well-tested, quality code submissions from students while battling the double-edged sword of the autograder. While autograders can provide feedback to students quickly, students can become reliant on the autograder as the primary means for determining correctness of their code. In a similar spirit, instructors also frequently promote submitting early and not waiting until the last second. To encourage students to submit fewer erroneous submissions and completing programming assignments earlier, we examine a policy change in lab submissions from time-restricted submissions to point-restricted submissions, implemented in consecutive semesters of a large Computer Architecture course. We survey students on their initial perception of the two policies, then survey students on their perception at the end of the semester. We also analyze the lab data, comparing success metrics and timelines of submissions between the two semesters. Several labs experienced a statistically significant increase in correct, first submissions under the point-restricted policy. We use these results to lead discussion about our experience using a point-restricted policy for larger programming assignments.

1 Introduction

The cost of poor software quality is calculated to be in the hundreds of billions of dollars[1]. The quality of software relies on the quality of skills programmers possess. In other words, to succeed as programmers, students need to develop high-quality code[2]. To produce this code, students must learn how to rigorously test, bug track, and debug. Unfortunately, the increasing use of autograders, an automated grading program often used to quickly and automatically assess student-submitted code, can short-circuit students' learning of these skills. Students may be more grade-oriented than learning-oriented[3,4] and students may also prioritize obtaining the grade they desire over an instructor pedagogy that may be perceived as a threat to the students' grades.[5] Therefore, students may be incentivized to use the autograder as their test suite rather than learning how to develop their own tests and other methods that increase confidence in their code quality.

As there are no autograders in industry, instructors face the challenge of maintaining the benefits of using the autograder, such as alleviating some grading burden and providing feedback quickly to students[6], while still promoting healthy coding habits that can be transferred to industry. Instructors may also wish promote healthy work habits, such as starting early.

One option instructors may turn to is crafting a policy regarding programming assignments to try to address all of these concerns.

In this paper, we present a research study comparing two homework submission policies, a time-restricted policy and a point-restricted policy, and their relative effects on students' ability to write bug-free code and their decisions to work on coding problems well before the final deadline. We sought to answer the following questions by implementing these policies in a Computer Architecture course.

- RQ1: How do students perceive a time-restricted lab submission policy versus a point-restricted lab submission policy?
- RQ2: How do these policies affect when students work on assignments and on students' submission of bug-free code?

2 Background and Related Work

Many in computing education are calling for more instruction on testing[7]. There have been a number of approaches taken to address this need in the CS curriculum. Approaches taken to address this need include better tool support for teaching testing[8], web-based tutorials[9] and games[10], and a vision for a test-driven development (TDD)-centered CS curriculum[11].

Introducing testing concepts early in a student's programming career (i.e., in CS1) may provide several benefits to a student's coding ability.[12,13] A key component to increasing testing is designing programming tasks that help students realize the importance of testing, as Isomöttönen and Lappalainen discuss when incorporating TDD-like testing and game contextualization game to their CS1 course.[14] Another natural point to examine the teaching of testing is in software engineering courses, frequently seen as an upper-level course, incorporating testing concepts pertaining to software.[15,16] However, the body of work addressing more testing thins as we take a look at mid-level Computer Science courses. A student may be introduced to programming concepts in their CS1 course, then go more in-depth in their software engineering course, but the importance of testing may be lost in the meantime and it may be difficult to translate their testing knowledge to other courses, such as Computer Architecture, in our case.

Autograders have had a substantial positive impact on getting feedback quickly to students and saving instructors countless hours of grading and supplying feedback, alleviating some of that burden.[6] However, autograders still remain a significant issue to address when it comes to encouraging positive software development habits and testing behavior. As several have reported[17,18,19,20], students may be negatively influenced by the autograder, such as relying on the autograder to know if their program is correct or exploiting the autograder to gain information.

One approach Karavirta, Korhonen, & Malmi took to address potential autograder exploitation issues was looking at clustering groups of students based on their resubmissions to an automatic assessment system, identifying which group could most use the guidance of having restricted number of submissions, as to discourage an "aimless trial-and-error problem solving method."[21] Another approach is incorporating sutdent-written tests into the autograding and feedback process to tackle some of these autograder concerns.[17,18,19] Baniassad, Zamprogno, Hall, and Holmes

adopt a similar point-restricted policy approach in a software engineering course of a "regression penalty" every time a student's grade dropped submitting against an autograder, also switching from a policy comparable to the time-restricted policy where students could continuously submit, but only check their grade every twelve hours.[20]

3 Data and Policy Context

3.1 Course Context and Lab Assignments

Computer Architecture is a required course for Computer Science majors at a Midwestern University and is typically taken by second-year undergraduate students. 300-400 hundred students enroll each semester. Prerequisites to the course are two introductory programming courses and a data structures course that may be taken concurrently. The course meets twice a week, Mondays and Wednesdays, for lecture and has a lab component.

The course has 13 long programming assignments called labs that are collectively worth 20% of a students' final grade. The rest of a student's final grade consists of in-class assignments that are worth 15%, quizzes that are worth 55% (11 quizzes, 5% per quiz), and the final exam, worth 10%. Lab assignments are intended to challenge students the most on content knowledge compared to other course components, but they are lower stakes as seen by the grade percentage. Students write code in Verilog, MIPS assembly, and C. These labs are mini-capstone exercises, where students complete a multi-component or multi-function coding task in stages. For most labs, students may work alone or in groups up to three students (three labs are to be completed individually). The lab assignment is made available at the start of the week and is split into two parts. Part one (usually one module or function) is worth 20% of the lab grade and due Friday at 8pm. Part two (which usually uses the module or function built in Part one) is worth 80% of the lab grade and due Monday at 8pm, but also has a late deadline up to 48 hours late with a 10% penalty every 12 hours past the initial deadline. Most lab parts are made up of only one question, but a couple lab parts consist of two questions with separate submissions. Part one is meant to serve as a comprehension check and part two is intended to be the hardest assessment for a particular content module.

Assignments are completed through an online learning platform, PrairieLearn[22]. An incomplete set of test cases is provided, and students are instructed to extend those test cases.

3.2 Time-Restricted Policy

Spring 2022 lab assignments allowed for one submission for full credit every twelve hours until the due date. The instructor used a time-limited policy because of the limited features on the online learning platform to disincentivize students from relying on the autograder. The policy had a positive byproduct of promoting good work habits of starting earlier on the assignment.

3.3 Point-Restricted Policy

With the introduction of a new feature to the online learning platform, Fall 2022 lab assignments limited the number of submissions available for a question. More specifically, the policy allows

for two submissions for full credit, then decreases the credit value for subsequent submissions until no submissions remain. The use of two submissions for full credit rather than one was provided as a mechanism to guard against complaints about "accidental submissions." In addition, students can receive extra credit for submitting a correct solution by the early deadline (24 hours before the due date). Part one has a 5% bonus and part two has a 3% bonus. The instructor adopted this policy to continue the goal of encouraging students to test their code and not rely on the autograder, but with hopes of maintaining the byproduct of students starting early on the assignment.

4 Methods

We are interested in student opinion and perception of the two policies to address RQ1, as well as metrics that may indicate students taking a desired approach to labs with respect to our goals RQ2.

4.1 Student Survey

We surveyed students from the Point-Restricted-Policy semester regarding this policy change at both the beginning and end of the semester to help answer our research question, RQ1, of how students perceive a time-restricted lab submission policy versus a point-restricted lab submission policy. We did not survey students from the Time-Restricted-Policy semester as we did not know at the time that we would have this chance to change policies. However, most students from the Point-Restricted-Policy semester are familiar with a time-restricted policy, as our university's data structures course is a co-requisite for our Computer Architecture course and uses a time-restricted submission policy for their long programming assignment equivalent. For clarity, we will refer to the data structures course as the Time-Restricted-Policy course.

4.1.1 Pre-Semester Survey

We surveyed students at the beginning of the Point-Restricted-Policy semester. We asked for their preference between the point-restricted policy and the time-restricted policy, explaining both policies and explicitly stating our goals of trying to encourage students to thoroughly test and make fully correct submissions, as well as start the assignment early. We also asked students to explain their policy choice. We use these responses to provide depth to our answer for RQ1, better understanding the thought process behind students' initial perception of the policies and how the responses align with (or divert from) our goals that we explicitly described to students.

4.1.2 End of Semester Survey

We then surveyed students from the Point-Restricted-Policy semester at the end of the semester.

We had the students compare the Poinr-Restricted-Policy course against Time-Restricted-Policy course. We specifically asked in what ways did a student approach completing the programming assignments differently between the two courses. Students were instructed to skip the question if they did not take the course recently with the time-restricted policy. We asked this question to

answer RQ1 from the perspective of students who have now experienced both policies and are able to reflect on both. These responses also help inform our answer to RQ2, supplementing our observations through the lab success measures as we describe in the next subsection with students' self-described approaches. We categorize responses on if their approach changed and was effected by the different policies or if it did not change. (we exclude responses that were unclear or if a student explained a change in approach based on factors outside of the policy, such as mentioning a difference in course difficulty). We analyze the responses from those who did express a change in approach and uncover several themes.

4.2 Lab Success Metrics

We look at a number of metrics that could indicate lab success for each lab between the two semesters. We drop labs 1, 6, 7, 9, and 13 from our analysis as they were modified between semesters and do not offer direct comparison. We are interested in the percentage of correct first submissions, which could indicate intent of only submitting fully correct code.

4.3 Student Submission Timelines

We also examine the percent of student first submissions received with respect to the lab deadline. We display aggregates of all first submissions from part one of the labs under consideration and all first submissions from part two of the labs under consideration for ease of viewing as well as to make observations across all labs. We look for patterns of when students first start submitting and when students submit correct first submissions. Points of interest include spikes in submissions and correct submissions, as well as where and which lines between the two semesters diverge.

5 Results

5.1 RQ1: Student Perception

Students from the point-restricted semester were surveyed at the beginning of the semester regarding the two policies. Of the 248 responses, 166 students preferred the time-restricted policy to the point-restricted policy (82 students).

Students who preferred the time-restricted policy commonly brought up that they would feel more incentivized to start early under the policy. However, this was frequently paired with less desired reasoning as to why they would be incentivized to starting early. Unsurprisingly, students who mentioned they would start early brought up wanting to maximize the number of full-credit attempts they are allowed on a lab. Students also commonly mentioned using feedback from the autograder as part of their testing when discussing maximizing attempts, moving even further away from our goal of students only submitting fully correct code.

Students who preferred the point-restricted policy most commonly brought up that the policy would be less disruptive to their workflow. Students who preferred the point-restricted policy also touched on being more incentivized to be more thorough and thoughtful in ensuring their code is

correct and improve their testing and debugging skills. Students also touched on the policy helping to ease off the autograder as a resource.

While students preferred the time-restricted policy, the explanations of the students who chose point-restricted policy more frequently aligned with our goal of encouraging students to thoroughly test their code and rely less on the autograder that their submission is fully correct.

5.1.1 Time-Restricted Policy Course Comparison

Students compared their approach to assignments from the Time-Restricted-Policy course against our Point-Restricted-Policy course. Of the 124 responses to this question, 52 reported no difference in approach and 56 touched on differences that were relevant to the policy difference (responses that did not fall into one of these two categories were unclear or the approach change was not related to the policy or its goals, such as one class's assignments being harder than the other).

One theme from the responses was students starting earlier in the Time-Restricted-Policy course and waiting longer to start in our Point-Restricted-Policy course. Many of the students' self-described actions lined up with the pre-course survey thoughts as to why they would start earlier, stating that they would take more submissions and use the autograder feedback as bigger resource in their solving approach. Along a similar vein, another theme was feeling less stress or pressure to get the submission right. However, for those whose workflows were adversely affected by the wait time, this policy was sometimes perceived as more stressful, with students finding the wait annoying, frustrating, or a waste.

Another theme was students being more thoughtful when submitting for the Point-Restricted Policy. Students described reviewing their code more, being more careful about submitting, and testing more. These responses were often met with students also expressing more stress or that the increased caution wasted a lot of time. Some students also reacted positively, feeling they were able to iterate faster and work on changes faster.

5.2 RQ2: Student Behavior

5.2.1 Lab Measures

We look at the number of first submissions student groups that were correct. We use Mann-Whitney U test to compare the labs with α value of 0.05. Table 2 reveals that Lab 2 Part 2, Lab 3 Part 2, Lab 4 Part 2, Lab 5 Part 2, Lab 8 Part 1, Lab 8 Part 2, and Lab 10 Part 2 all had significant differences in number of correct first submissions.

5.2.2 Submission Timelines

Examining the timelines in Figure 1, we see across all first submissions from part 1 of labs and part 2 of labs that more submissions from the time-restricted semester begin rolling in earlier, but of those submissions, the correct submissions are comparable between the two semesters.

	Time-Restricted % Correct Group (total)	Point-Restricted % Correct Group (total)	p-value	Absolute Effect Size
Lab 2 Part 1†	86.01% (289)	88.11% (252)	0.439	2.01%
Lab 3 Part 1	52.50% (84)	57.78% (78)	0.365	5.28%
Lab 4 Part 1	48.17% (79)	49.26% (67)	0.851	1.09%
Lab 5 Part 1	15.57% (26)	19.26% (26)	0.400	3.69%
Lab 8 Part 1	42.07% (69)	58.16% (82)	0.005*	16.09%
Lab 10 Part 1	87.20% (143)	83.33% (120)	0.340	3.87%
Lab 11 Part 1	91.93% (148)	90.07% (127)	0.575	1.86%
Lab 12 Part 1	40.38% (63)	46.32% (63)	0.308	5.94%

Table 1: Percentage of Student First Submissions that were marked correct. Significant differences indicated by an *. Lab Parts that have two questions combined in reporting numbers indicated by a †. For this table and the following, we exclude labs 1, 6, 7, 9, and 13 as they were modified between semesters and do not offer direct comparison.

Looking at all part 1 first submissions, we see submissions and correct submissions begin to diverge between policies as we approach 24 hours before the deadline, i.e., the point-restricted extra credit deadline. However, across all part 1 labs, as we approach having all our submissions in, the ratio of correct submissions among all submissions also converges at a comparable percentage.

Moving our attention to part 2 of all labs, we observe that for all first submissions that the percentage of submissions follow a very comparable trajectory. However, we see the percentage of submissions that are correct begin to diverge heading towards the 24 hour early deadline and stay separated at the deadline and through to the 48 hour late deadline, in favor of the point-restricted semester.

We have the same observations about the timelines for all submissions for part ones and part twos of labs as all first submissions we include in Figure 1.

6 Discussion

6.1 Implications for Instruction

As we have observed, our Point-Restricted policy has had a modest improvement on students submitting fully-correct coding submissions early for lower-stakes assignments. A Point-Restricted policy may be an appropriate nudge for instructors to help encourage students to thoroughly test their code and ensure their submission is bug-free. However, students may push back and voice concerns, as we observed in our pre-semester survey and end of semester survey. Instructors looking to adopt a similar policy may want to be mindful how to address such concerns, such as knowing how one may want to explain the benefits and importance of the policy, as well as consider mitigation strategies when appropriate. These considerations may help allow the policy to greater benefit students while easing some student reservations.

	Time-Restricted % Correct (total)	Point-Restricted % Correct (total)	p-value	Absolute Effect Size
Lab 2 Part 2	52.60% (91)	76.06% (108)	<0.001*	23.46%
Lab 3 Part 2	32.03% (49)	50.76% (67)	0.001*	18.73%
Lab 4 Part 2	51.88% (83)	74.43% (99)	<0.001*	22.55%
Lab 5 Part 2	09.43% (15)	18.18% (24)	0.030*	8.75%
Lab 8 Part 2†	46.50% (146)	60.14% (166)	<0.001*	13.64%
Lab 10 Part 2	44.79% (73)	68.38% (93)	<0.001*	23.59%
Lab 11 Part 2	41.98% (68)	49.29% (69)	0.204	7.31%
Lab 12 Part 2	14.56% (23)	19.86% (28)	0.225	5.30%

Table 2: Percentage of Student First Submissions that were marked correct. Significant differences indicated by an *. Lab Parts that have two questions combined in reporting numbers indicated by a †.

6.2 Limitations

6.2.1 Survey Limitations

We address limitations with our surveys. Firstly, a major limitation was a lack of survey responses from the time-restricted semester, since we did not know this new feature would be implemented. Secondly, much of our survey data comes from open-ended responses, and therefore may not represent a whole, complete opinion with respect to all aspects we are considering in this study. Lastly, while we excluded responses that directly addressed aspects of the Time-Restricted-Policy course and the Point-Restricted-Policy course that were not regarding the difference in policy, these course differences still could have impacted responses.

6.2.2 Generalizability

We now touch on some of the concerns that affect the generalizability of our findings. As we previously discussed, our data is from a large Computer Architecture course primarily taken by second year undergraduates. Additionally, this course is conducted at a large research university in the midwestern United States. While our specific implementation of a point-restricted policy may be adapted to fit other contexts, the effects of the policy change may not transfer to other contexts.

7 Conclusion and Future Work

We examine two different policies, a time-restricted policy and a point-restricted policy, to see which policy aligned more with the goal of students only submitting well-tested, quality code submissions. Under the point-restricted policy, we experienced a modest increase in correct first submissions. For future work, we shift our focus to tackle the testing aspect of submitting only quality code. Integrating more explicit testing components to labs will answer the question of how well students are testing these submissions, which is an important component to ensure

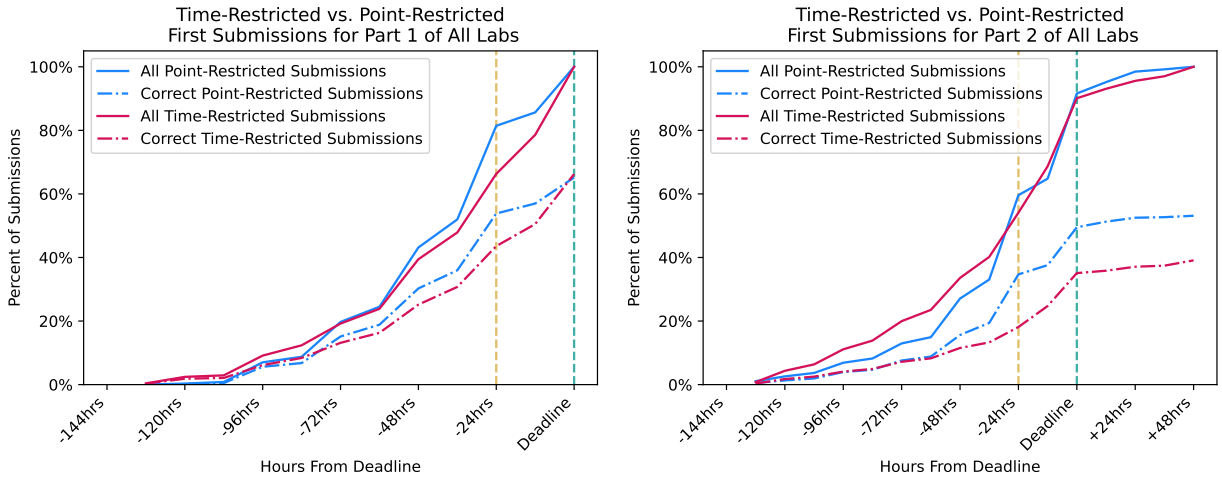

Figure 1: Comparison of the Percentages of All First Submissions Received with Respect to their Deadlines for All Lab Part 1s and Lab Part 2s.

correct first submissions to the autograder.

References

[1] Herb Krasner. The cost of poor software quality in the us: A 2020 report. Proc. Consortium Inf. Softw. QualityTM (CISQTM), 2021.

[2] Lillian Cassel, Alan Clements, Gordon Davies, Mark Guzdial, Renée McCauley, Andrew McGettrick, Bob Sloan, Larry Snyder, Paul Tymann, and Bruce W. Weide. Computer science curriculum 2008: An interim revision of cs 2001. Technical report, New York, NY, USA, 2008.

[3] James A Eison. A new instrument for assessing students' orientations towards grades and learning. Psychological Reports, 48(3):919–924, 1981.

[4] James A Eison. Educational and personal dimensions of learning-and grade-oriented students. Psychological Reports, 51(3):867–870, 1982.

[5] Dayna DeFeo, Trang Tran, and Sarah Gerken. Mediating students' fixation with grades in an inquiry-based undergraduate biology course. Science Education, 30, 02 2021. URL https://doi.org/10.1007/s11191-020-00161-3.

[6] Pete Nordquist. Providing accurate and timely feedback by automatically grading student programming labs. J. Comput. Sci. Coll., 23(2):16–23, dec 2007. ISSN 1937-4771.

[7] Terry Shepard, Margaret Lamb, and Diane Kelly. More testing should be taught. Commun. ACM, 44(6): 103–108, jun 2001. ISSN 0001-0782. URL https://doi.org/10.1145/376134.376180.

[8] Andrew Patterson, Michael Kölling, and John Rosenberg. Introducing unit testing with bluej. In Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE '03, page 11–15, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581136722. URL https://doi.org/10.1145/961511.961518.

[9] Sebastian Elbaum, Suzette Person, Jon Dokulil, and Matt Jorde. Bug hunt: Making early software testing lessons engaging and affordable. In 29th International Conference on Software Engineering (ICSE'07), pages 688–697, 2007. URL https://doi.org/10.1109/ICSE.2007.23.

[10] Pedro Henrique Dias Valle, Armando Maciel Toda, Ellen Francine Barbosa, and José Carlos Maldonado. Educational games: A contribution to software testing education. In 2017 IEEE Frontiers in Education Conference (FIE), pages 1–8, 2017. URL https://doi.org/10.1109/FIE.2017.8190470.

[11] Stephen H. Edwards. Rethinking computer science education from a test-first perspective. In Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA '03, page 148–155, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137516. URL https://doi.org/10.1145/949344.949390.

[12] Kevin Buffardi and Stephen H Edwards. Impacts of teaching test-driven development to novice programmers. International Journal of Information and Computer Science, 1(6):9, 2012.

[13] Juan Jenny Li and Patricia Morreale. Enhancing cs1 curriculum with testing concepts: A case study. J. Comput. Sci. Coll., 31(3):36–43, jan 2016. ISSN 1937-4771.

[14] Ville Isomöttönen and Vesa Lappalainen. Csi with games and an emphasis on tdd and unit testing: Piling a trend upon a trend. ACM Inroads, 3(3):62–68, sep 2012. ISSN 2153-2184. URL https://doi.org/10.1145/2339055.2339073.

[15] Peter J. Clarke, Debra Davis, Tariq M. King, Jairo Pava, and Edward L. Jones. Integrating testing into software engineering courses supported by a collaborative learning environment. ACM Trans. Comput. Educ., 14(3), oct 2014. URL https://doi.org/10.1145/2648787.

[16] Nicole Clark. Peer testing in software engineering projects. In Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, ACE '04, page 41–48, AUS, 2004. Australian Computer Society, Inc.

[17] Stephen H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, SIGCSE '04, page 26–30, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581137982. URL https://doi.org/10.1145/971300.971312.

[18] Kevin Buffardi and Stephen H. Edwards. Reconsidering automated feedback: A test-driven approach. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, SIGCSE '15, page 416–420, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450329668. URL https://doi.org/10.1145/2676723.2677313.

[19] Grant Braught and James Midkiff. Tool design and student testing behavior in an introductory java course. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education, SIGCSE '16, page 449–454, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450336857. URL https://doi.org/10.1145/2839509.2844641.

[20] Elisa Baniassad, Lucas Zamprogno, Braxton Hall, and Reid Holmes. Stop the (autograder) insanity: Regression penalties to deter autograder overreliance. In Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, SIGCSE '21, page 1062–1068, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380621. URL https://doi.org/10.1145/3408877.3432430.

[21] Ville Karavirta, Ari Korhonen, and Lauri Malmi. On the use of resubmissions in automatic assessment systems. Computer Science Education, 16(3):229–240, 2006. URL https://doi.org/10.1080/08993400600912426.

[22] Matthew West, Geoffrey L. Herman, and Craig Zilles. PrairieLearn: Mastery-based Online Problem Solving with Adaptive Scoring and Recommendations Driven by Machine Learning. In 2015 ASEE Annual Conference & Exposition, pages 26.1238.1–26.1238.14, 2015. URL https://peer.asee.org/24575.