Frequent, Timed Coding Tests for Training and Assessment of Full-Stack Web Development Skills: An Experience Report

Kathryn Bridson
University of Memphis
Memphis, Tennessee, USA
kbridson@memphis.edu

Scott D. Fleming
University of Memphis
Memphis, Tennessee, USA
Scott.Fleming@memphis.edu

ABSTRACT

This experience report describes the use of frequent, timed coding tests in a project-intensive software engineering course in which students first learn full-stack web development using Ruby on Rails and then apply their skills in a team project. The goal of the *skills tests* was twofold: (1) to help motivate students to engage in distributed practice and, thus, gain adequate coding skills to be an effective team member during the team project and (2) to accurately assess whether students had acquired the requisite skills and, thereby, catch deficiencies early, while there was still time to address them. Regarding the first goal, although several students indicated that the tests motivated them to engage in substantial practice coding, it was ultimately inconclusive as to the extent of the tests' impact on students' distributed practice behavior and on their preparation for the project. Regarding the second goal, the skills testing approach was indeed considerably more effective than graded homework assignments for assessing coding skill and detecting struggling students early. Lessons learned from our experiences included that students had significant concerns about the strict time limit on the tests, that the tests caused a spike in mid-semester withdrawals from the course that disproportionately impacted students from underrepresented groups, and that detecting struggling students was one thing, but effectively helping them catch up was a whole other challenge.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

software engineering education, skills testing, mastery learning, assessment, full-stack web development

ACM Reference Format:
Kathryn Bridson and Scott D. Fleming. 2021. Frequent, Timed Coding Tests for Training and Assessment of Full-Stack Web Development Skills: An Experience Report. In *The 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432549

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432549

1 INTRODUCTION

This experience report describes the use of a *skills testing* approach to enhance student mastery of software development skills and to provide more accurate and timely assessment of such skills in an undergraduate software engineering course. Following the basic approach used by other software engineering courses in the literature (e.g., [9, 16]), our course teaches software engineering principles in the context of "full-stack" web development—that is, the development of both client and server software built upon a platform that includes, among other things, a web server framework and a database management system (DBMS). In particular, the course uses Ruby on Rails (*Rails* for short) as the web server framework and PostgreSQL as the DBMS. During the first half of the semester, students receive practical training on how to develop web applications using Rails and PostgreSQL. The second half of the semester is project focused, with the students working in collaborative teams to build Rails-based web applications.

In prior iterations of the course, a combination of worked examples and practice homework assignments was used to train students; however, the homework assignments were consistently found to be inadequate both in helping students gain the necessary expertise and in assessing their mastery of the requisite skills. Teams frequently reported unproductive team members who were unable to complete the tasks assigned to them and who, all too often, contributed code that was itself broken or caused other team members' code to break. In one-on-one interactions with these struggling students, instructors often found that they were unable to perform basic tasks covered in the homework assignments. However, inexplicably, these students often had completed the homework assignments successfully, earning high marks. Thus, the instructors concluded that the homework assignments were both failing to help these students gain the necessary skills for the project and failing to accurately assess their skills. Others in the literature have reported similar problems with traditional course designs being unsuccessful in helping students master software development skills [8, 16].

To address these problems, we introduced a skills testing approach with two main goals:

(1) **Training:** Improve student preparation for the project by helping them learn to perform common development tasks quickly and correctly.
(2) **Assessment:** More accurately assess whether students have achieved the desired level of proficiency and reveal struggling students earlier in the course.

The skills tests were administered each week in class (i.e., in a controlled testing environment) during the first half of the course. Each test required students to complete a practical coding task. Full

access to the web was allowed, but only existing documentation and forum posts could be viewed (e.g., no texting or posting). Students were given 25 minutes to complete the task, and the time limit was strictly enforced. The tests were graded as pass/fail, and for every two tests, an additional second-chance test was offered.

2 RELATED WORK

Our skills testing approach has a number of things in common with prior applications of *mastery learning* for teaching coding [10]. Mastery learning aims to help students achieve a high level of understanding of a domain by having them progress through a series of learning steps, not advancing to the next step until the current step has been sufficiently mastered, and moving through the steps at their own pace [1]. *Mastery tests* are used to assess whether a learner is ready to advance to the next step. Studies have shown the considerable educational benefits of mastery learning, such as learning gains in lower ability students [18]. Similar to our skills testing approach, mastery learning approaches have been applied to help prepare students to work on advanced software projects [15, 23]. Like our skills tests, the mastery tests used in these approaches have employed strict time limits [23] and have used pass/fail grading [8, 19, 23]. However, due to the time constraints of our course, our approach diverged from mastery learning in two key ways: students could advance without passing a test, and students had to follow the course schedule (i.e., not go at their own pace). To compensate for these differences, we followed prior (non-CS) approaches [12, 13, 20, 25] and incorporated *second-chance tests*, which allow students to receive feedback on tests and then be retested on the same material with some form of grade replacement.

Athletic Software Engineering [14, 16, 17] is another educational approach similar to our skills testing. Athletic Software Engineering also aims to prepare students for full-stack web development projects (albeit in Meteor rather than in Rails). They aim to encourage students to engage in distributed practice by challenging them with *Workouts of the Day* (WODs), which are similar to our skills tests. Like our skills tests, WODs require students to perform common development tasks, are strictly timed, and use a pass/fail grading system. The key differences between their approach and ours are that they have additional procedures to encourage students to practice (e.g., in-class group practice sessions) prior to graded testing, and they do not offer second-chance tests.

3 SKILLS TESTING DESIGN AND RATIONALE

As mentioned above, our skills testing approach aimed to address training and assessment goals. Regarding our training goal, we hypothesised that timed tests would more effectively communicate our expectation that students would be able to perform common tasks *quickly* as well as correctly. Furthermore, we hypothesized that the time limit would motivate students to prepare for tests by engaging in repeated practice of tasks, thus, causing them to benefit from practice effects. Regarding our assessment goal, we hypothesized that a timed coding test would more accurately assess how well prepared students were to be productive team members during the project than would our prior coding homework assignments. The rationale for this hypothesis was twofold: the time limit would reveal the inability to perform basic tasks quickly, and a controlled testing environment would prevent behaviors that invalidate the assessment, like overreliance on help or copying.

Table 1: Skills testing schedule and learning goals assessed.

Week	Test	Main Learning Goals
1	1	• Setup a Rails web development environment • Initialize and run an existing Rails web app
2	2	• Add web pages to a Rails web app • Add web forms to a Rails web app
	1-SC	*Second chance for test 1 goals*
3	3	• Create a model class to store data in a DB • Insert seed data into a DB • Render backend data on a web page
4	4	• Create model validations to catch invalid data • Create automated tests of the model
	2/3-SC	*Second chance for test 2 and 3 goals*
5	5	• Create UI features for CRUDing DB records
6	6	• Create a one-to-many model association • Create model tests that include associations • Create UI features for CRUDing model objects that manipulate association links between objects
	4/5-SC	*Second chance for test 4 and 5 goals*
7	7	• Secure an application with authentication • Restrict resource access based on authorization
8	*6/7-SC*	*Second chance for test 6 and 7 goals*

Another key aspect to our skills testing approach was the use of a pass/fail grading system. One reason for this decision was that it is inherently difficult to assess partial credit commensurate with a student's level of mastery based only on a code submission. For example, it is hard to tell from partial or broken code how much the student actually understood, and many students have the bad habit of copying and pasting existing code (e.g., found on the web) without understanding it, which can result in broken but sort-of-correct-looking code. Another rationale for pass/fail grading was that timely grading of the tests was critical, and making a pass/fail determination takes less time than assessing partial credit.

3.1 Overview of the Rails Skills Tests

We created a sequence of seven weekly skills tests to cover a core set of common coding tasks in Rails-based web development. Table 1 provides a description of each test. The tests covered a range of learning goals, such as simply running an existing app, making an app retrieve and display data from a database, and securing an app with authentication. The rationale for having seven weekly tests was that spreading them out over many weeks would encourage students to engage in *distributed practice*—that is, the breaking up of practice over an extended period of time. Distributed practice has been shown to improve student learning and test performance versus cramming-style massed practice [3, 6, 24].

In addition to the seven weekly tests, we also included four second-chance tests, one for roughly every two weekly tests. Each second-chance test was new, but it would cover the same learning goals as the associated previous tests, often using an isomorphic problem. One reason for these second chances was that second-chance testing has been found to improve student outcomes by allowing students to study and learn from feedback on a prior test [20, 21]. Another reason is that second-chance testing has been found to decrease test anxiety [4, 5]. We anticipated that anxiety might be high for the skills tests because of the risk that a careless

mistake might cause a student to run out of time and fail. Even for highly skilled students, such mistakes can happen occasionally. Thus, the second chances help to ensure that such students can earn full marks despite the occasional slip up.

3.2 Coding Tasks

Each skills test provided the code base for a web app and a set of task instructions. For example, Skills Test 4 included a partially complete web app with a `Movie` model class, test fixtures for the class (i.e., sample data for use in testing), and a unit test that used the fixtures. The student was tasked with adding two attribute validations to the model class (e.g., to ensure that the `title` attribute cannot be `nil`) and with adding two new unit tests that test the validations.

In designing each skills test, several key principles were followed. First, to ensure that a test was not overlong, it must be doable for a well-prepared student in less than half (or at most two-thirds) of the time limit (25 minutes). As a confirmation that this principle was followed, the first author, who did not write the tests, was able to solve each of them in 10 minutes, except for one that took 15 minutes. Second, the starting app must be quickly understandable to a well-prepared student. Thus, the starting apps were generally kept small and used logic that was as straightforward as possible. Third, the test must contain nothing tricky. Thus, the tests generally included only material covered in the accompanying worked example documentation.

Because the aim of the skills tests was to prepare students for the team project, the development environment they used on the skills tests was the same as the one they would use for the project. They used all the same tools and technologies (e.g., code editor, web browser, operating system, web-server framework, and DBMS). Additionally, the skills tests were open book, open note, and mostly open internet to reflect the access to these resources that students working on the project would typically have. The only exception was that students were not allowed to communicate directly with any other individual (except the course instructor) during a test.

3.3 Test Administration

All tests were conducted in a controlled classroom environment. At the start of a test session, the test code and instructions were distributed using Git and GitHub. Students submitted their work via an online dropbox that would automatically lock once the test time expired. Thus, to successfully pass the test, a student had to complete and upload their submission before the dropbox locked.

To to deter cheating, each student was required to record a screen-capture video of their test performance. As a rule, the instructor generally viewed the videos only if there was some specific reason to do so, such as to investigate suspected cheating or to understand a problem that a student encountered during a test.

4 COURSE EXPERIENCES

We applied our skills testing approach during Spring 2020 in the undergraduate software engineering course at the University of Memphis. The second author served as instructor for the course, and the first author assisted as a consultant with the course and skills tests. 38 junior- and senior-level undergraduate students enrolled in the course. Seven (18%) of the students were female. Eight (21%) of the students were Black or African American. One of the students was Hispanic or Latino, and one was American Indian.

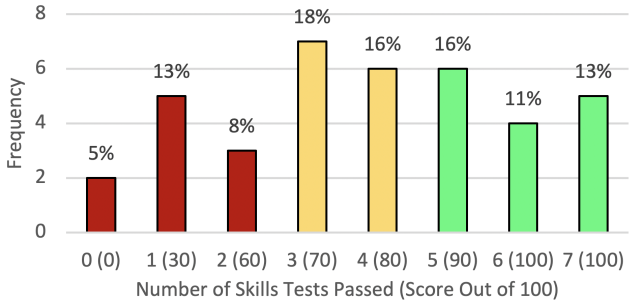

Figure 1: Student performance on the skills tests. The X-axis is the number of tests passed (with the points out of 100 awarded for passing that many tests), and the Y-axis is the number of students who passed that many tests. Green bars denote scores that correspond to an A grade; yellow bars denote scores that correspond to a B or a C grade; red bars denote scores that correspond to a D or an F grade.

Seven skills tests, plus three second-chance tests, were administered during the first seven weeks of the course. There was supposed to be one additional second-chance test; however, the COVID-19 pandemic unexpectedly caused in-person classes to be suspended following the seventh week of the course. Thus, the last second-chance test could not be administered in class and was dropped. To account for the loss of this second-chance test, the grading scale for the skills tests was softened (e.g., only 6 passed tests were required to earn full marks as per the X-axis in Figure 1).

4.1 Skills Test Performance

As Figure 1 shows, 40% of the students passed 5 or more skills tests, thus, earning the equivalent of an A grade on the tests. Around one-third (34%) of the students were borderline, passing only 3 or 4 out of the 7 tests, earning the equivalent of a B or a C grade. Over a quarter (26%) of the students effectively failed the skills test portion of the course, passing 2 or fewer of the 7 tests. The skills tests accounted for 35% of a student's final grade in the course.

4.2 Mid-Semester Withdrawals

Five of the 38 students withdrew from the course midway through the semester, after the last skills test was administered. Two of the five passed 0 tests, and the other three passed only 1 test. Thus, no students who passed 0 tests participated in the team project, and less than half of those who passed only 1 test participated.

In addition to the five students who officially withdrew from the course, three students who remained enrolled in the course did not participate at all in the project. These students were more diverse in their skills test performance than the ones who dropped, with one passing 5 tests, one passing 4 tests, and one passing 2 tests. The reasons for their total lack of participation were not entirely known; however, the student who passed 5 of the 7 tests did inform the instructor that they would be unable to participate in the project due to personal issues caused by COVID-19. Because these students made no attempt to contribute to the project, we will treat them

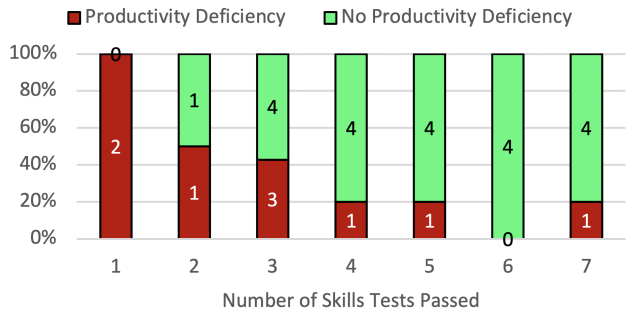

Figure 2: The relationship between the number of skills tests passed (X-axis) and the proportion of students who did and did not exhibit a productivity deficiency during the team software project (Y-axis). Red and green bars denote the proportion of students who did and did not exhibit a productivity deficiency, respectively. Each bar is annotated with the number of students.

as if they had withdrawn and omit their data from the analysis of project productivity below.

4.3 Project Productivity Deficiencies

Skills test performance was a good predictor of whether a student would have a productivity deficiency during the project—that is, as the number of tests passed increased, the number of students with productivity deficiencies tended to decrease. The project involved two 2-week development iterations, and at the outset of each iteration, each student planned the development tasks they would complete by the end of the iteration. A student received a productivity-deficiency deduction for an iteration if they completed substantially fewer tasks than they planned, and the work they completed was considerably less than what would be reasonably expected for a 2-week assignment. As Figure 2 shows, all of the students who passed only 1 skills test exhibited a productivity deficiency during the project, and roughly half of the students with 2 or 3 tests passed exhibited a productivity deficiency. In contrast, far fewer of the students with 4 or more passed tests exhibited productivity deficiencies (only 3 out of 19). Indeed, there was a strongly negative correlation between the number of students with a productivity deficiency and the number of skills tests passed (Pearson: $r(5) = -0.86$, $p = 0.014$).

4.4 Student Opinions

To better understand students' impressions of the skills tests, we asked them to complete an opinion questionnaire after the final exam. A small amount of extra credit on the exam was given as compensation for completing the questionnaire. The instructor assured the students that their responses would not be reviewed until after final grades were assigned. In the end, 29 (94%) of the 31 students who took the final completed the questionnaire.

The questions were a mix of Likert-style quantitative questions and open-ended questions. In this section, we report only the quantitative questions. The open-ended questions generally asked participants to elaborate on their responses to the quantitative questions. Where relevant, we will mention the things they said in Section 5.

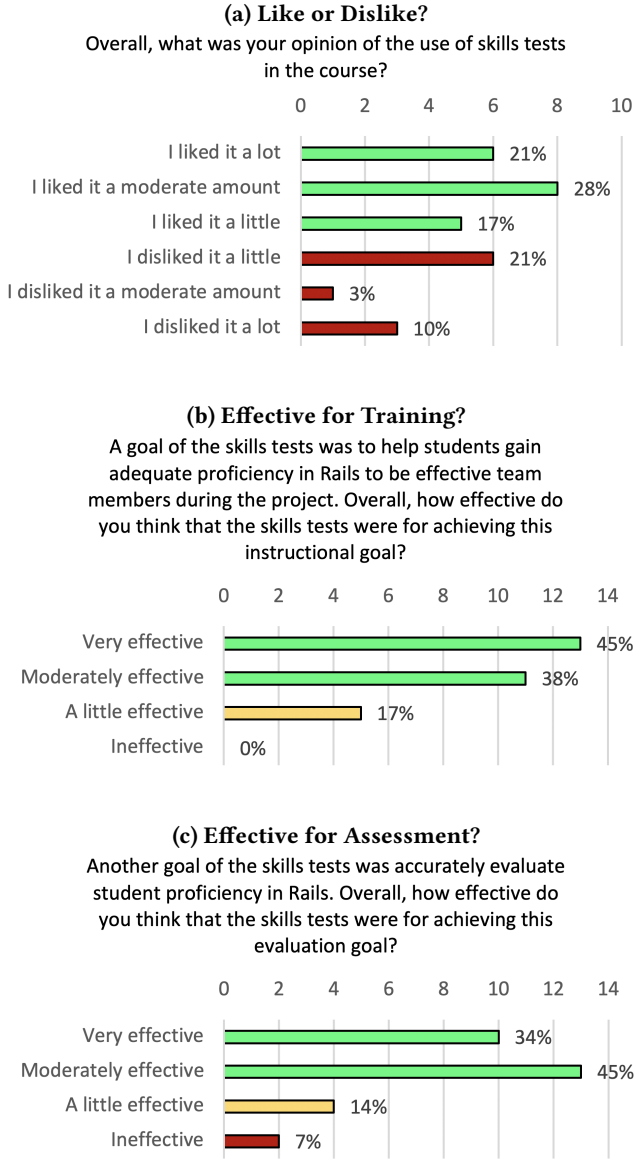

Figure 3: Student responses to the opinion questions regarding their overall impressions of the skills tests and their effectiveness.

4.4.1 Overall Opinions of Effectiveness. The first part of the opinion questionnaire sought to get students' overall impression of the skills tests and their effectiveness. Although students may not be able to objectively assess the effectiveness of the skills tests, we wanted to know whether they *believed* that the tests were effective, because their beliefs about the tests could impact class morale and their motivation to invest effort into the tests. Figure 3 lists these questions and provides a summary of the students' responses.

Overall, a strong majority of the students viewed the skills tests favorably. As Figure 3a shows, roughly two-thirds (66%) of students liked the skills tests to some degree, with roughly half (49%) of

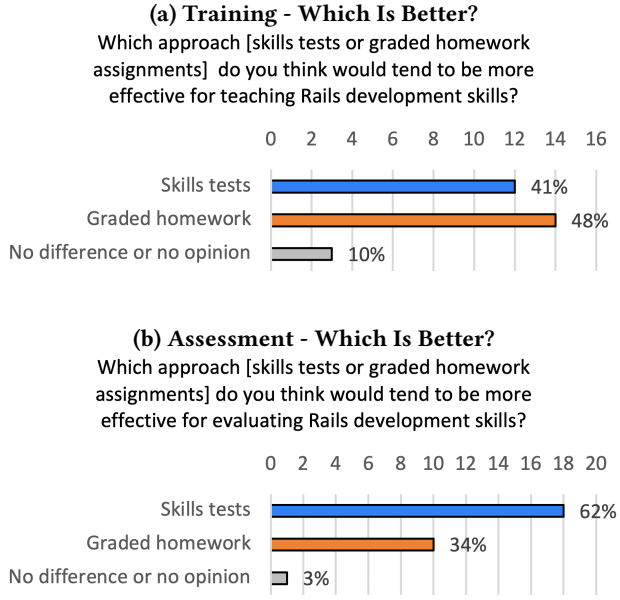

Figure 4: Student responses to the opinion questions that asked them to compare the skills tests to graded homework assignments.

them liked the skills tests a moderate amount or a lot. However, that means that roughly one-third (34%) of the students held a negative opinion of the skills tests, and 5 (13%) of the students held a moderately or strongly negative opinion of the tests.

Regarding the effectiveness of the skills tests for helping students gain proficiency in Rails-based web development, all the students found the tests to be at least somewhat effective. As Figure 3b shows, no students rated the skills tests as ineffective for training, and a very strong majority (83%) found them to be moderately or very effective.

Regarding the effectiveness of the skills tests for evaluating development skills, the students were again generally positive. As Figure 3c shows, 93% of students thought that the skills tests were effective to some degree for assessment, and 79% were more positive, rating the tests moderately to very effective. However, 2 (7%) of the students thought the tests were ineffective for assessment.

4.4.2 Skills Tests versus Graded Homework Assignments. The second part of the opinion questionnaire asked students to compare the skills tests with graded homework assignments, the main competing approach for training and assessing Rails development skills. Similar to the opinion questions about the effectiveness of the skills tests, the rationale for these questions was to see what students *believe* about the relative effectiveness of skills tests versus homework assignments, because their beliefs might suggest which approach would engender higher morale and/or greater motivation. Figure 4 lists these questions and provides a summary of the students' responses.

Student opinions were almost evenly divided on whether they thought that skills tests versus graded homework assignments would be more effective for *learning* Rails-based web development skills. As Figure 4a shows, fourteen (48%) of the students thought that graded homework assignments would be more effective for learning, whereas twelve (41%) of the students thought that skills tests would be more effective.

In contrast, the students were more strongly in favor of skills tests versus graded homework assignments where *assessment* was concerned. As Figure 4b shows, 62% of the students thought that skills tests would be more effective for assessment, whereas only about one-third (34%) of the students thought that graded homework assignments would be more effective.

5 KEY TAKEAWAYS AND LESSONS LEARNED

5.1 Improved Assessment? Yes!

We will discuss our goal of improving assessment first, because it was the one that had a more definitive (and positive!) outcome. Numerous indicators pointed to the skills tests as being substantially more effective for assessment than graded homework assignments. Skills test scores were strongly predictive of which students would exhibit productivity deficiencies during the project (recall Figure 2). Qualitatively, the instructor generally felt confident that skills test performance was a good reflection of a student's skill level. Furthermore, the instructor also felt that he had a much better handle throughout the course on who was succeeding and who was struggling. As early as the third skills test, the instructor was able to identify students in trouble and offer help. This enhanced awareness was a major improvement over prior semesters in which such issues were discovered only very late in the course. Even the students tended to respect the skills tests as an effective assessment, with a strong majority indicating so in the opinion questionnaire (recall Figure 3c). Moreover, a strong majority also thought that the skills tests would be more effective for assessment than graded homework (recall Figure 4b).

5.2 Improved Training? Inconclusive

Unfortunately, it was less clear whether the skills tests induced the desired practice effects and, thus, improved student preparation for the team project. The instructor's perspective on this goal was inhibited by the unexpected move to an on-line format during the projects. As a consequence, the instructor did not have many opportunities to observe students working on the project and to notice how well they were performing. The students' opinion questionnaire feedback was generally very positive regarding the tests effectiveness in preparing them (recall Figure 3b); however, the students were split almost evenly on whether skills tests or homework assignments would have better prepared them (recall Figure 4a). In their responses to the open-ended opinion questions, five students (out of 29) specifically mentioned that the skills tests encouraged them to practice or study—for example, one said the following:

> For instance, for the practice skills test, I just kept doing them and doing them over and over again. Not so much to learn. Really, I just wanted to see how much time I could shave off my previous effort. But in the process, I was learning.

Thus, the skills tests clearly had the desired effect for some students; however, further study would be needed to understand the extent to which they are effective across all students.

5.3 Lessons Learned

5.3.1 Time Limit Concerns. By far, the aspect of the skills tests that the students voiced the most concern about was the time limit. Eighteen (62%) of the 29 students who completed the opinion questionnaire made a negative comment about the time limit in their responses to the open-ended questions. A key concern was the fear of encountering bugs during the test and running out of time while debugging, as expressed in this comment:

> The only thing that bothered me about the skills tests were the time restrictions… the time limit was very tight and did not really allow for any type of technical or system errors that may occur. For example during one skills test I left a closed parenthesis in a block which caused my code to fail. However since I knew the time frame was very short and strict this caused me to not really be able to go back and identify the problem with a clear head as I was frantic over the fact that the code was not working and I was running short on time.

It remains an open question whether failures due to careless mistakes unduly depressed test scores or whether a student's inability to resolve such errors within the time limit was a valid indicator that they had not achieved the desired level of proficiency. If such failure scenarios are not actually that common or problematic, then the question remains as to what can be done to assuage students' anxieties about them.

5.3.2 Increased Mid-Semester Withdrawals. The introduction of skills tests created a noticeable spike in the number of students who withdrew from the course at mid-semester. In the past four iterations of the course prior to introducing skills tests, a total of 2 students withdrew at mid-semester, and the average mid-semester drop rate per semester was 1%. In contrast, 5 (13%) of the students in the skills test class withdrew—roughly a tenfold increase. These high withdrawal rates are consistent with the high dropout and failure rates reported for mastery learning based approaches to teach coding skills (i.e., 21% [23] and 27% [15]).

5.3.3 Better Interventions for Struggling Students Needed. Although the skills tests were effective at revealing struggling students early, providing effective support to help those students remained problematic. The main approach adopted in the course was to invite struggling students to one-on-one tutoring sessions with the course instructors. A few students took full advantage of this invitation, spending lots of time in office hours, and anecdotally, the tutoring seemed to help these students improve significantly; however, the vast majority of struggling students never responded to the invitation. The other main help resource for students during the skills testing portion of the course was Piazza, to which they could post questions anonymously; however, few struggling students posted questions. Interestingly, during the project, when the course was moved online, Discord was adopted as a communication mechanism, and there was a noticeable increase in the number students seeking help. The reason for this increase may have been that Discord offers key features that Piazza lacks, including voice channels, synchronous messaging, and screen sharing, and those features improved the quality of feedback and help that could be provided to students.

5.3.4 High Instructor Effort. The instructor effort required to apply our skills testing approach was generally high. Because the skills tests were open book and students were allowed internet access, old tests could not be reused verbatim. Thus, a new version of each test needed to be created every time it was administered, and this task generally took 1–2 hours per test (and more if the test was totally new and original). Grading the tests also took substantial time, because, at the least, each test submission would need to be executed and the code inspected, which generally took 10–15 minutes per test. However, all too often, submissions would require extra effort to understand (e.g., the screen capture video might need to be viewed), which would greatly increase the time needed for grading. With 1–2 tests being given each week, the total time spent on creating, administering, and grading them added up to be a considerable effort. Such high effort has also been reported as a concern with numerous other mastery learning based approaches for teaching coding skills [2, 8, 15, 19, 23].

5.3.5 Issues with Equity and Inclusion. Equity and inclusion for underrepresented groups in computer science education programs remains a high concern, and thus, it is important to consider such issues in the evaluation of educational approaches. Fourteen (37%) of the 38 students enrolled in the course belonged to an underrepresented group based on gender or race/ethnicity. Unfortunately, the median pass rate on the skills tests for these students was noticeably lower than the median pass rate for the class (3 versus 4 tests passed), and 4 (80%) of the 5 students who withdrew from the course at mid-semester belonged to underrepresented groups. Changes to the skills testing approach that could be explored in future work to help address these negative trends include adopting a more equitable grading policy (e.g., as discussed in [11]) and introducing a student mentoring program, like the one used at Swarthmore College that was found to improve performance and retention among students from underrepresented groups [22].

6 CONCLUSION

In this experience report, we described our use of a skills testing approach to teach full-stack web development in an undergraduate software engineering course. The key takeaways of our experiences were (1) that the skills testing approach was considerably more effective than graded homework assignments for assessing coding skill and detecting struggling students early and (2) that, although a few students mentioned that the tests motivated them to practice coding, it was ultimately inconclusive as to its overall impact on students' distributed practice behavior and on their preparation for the project. Lessons learned included that students had significant concerns about the tests' strict time limit, that the tests caused a spike in mid-semester withdrawals from the course that disproportionately impacted students from underrepresented groups, and that detecting struggling students was one thing, but effectively helping them catch up was a whole other challenge. Our experiences also motivate a number of future directions, including how to reduce student test anxiety while still motivating them to engage in the level of distributed practice necessary to master full-stack development skills, how to reduce instructor effort (e.g., with auto-grading [7]), and how to provide more equitable grading and learner support to help *every* student succeed in computer science.

ACKNOWLEDGMENTS

We give special thanks to Jeff Atkinson for his exemplary service as teaching assistant for the software engineering course.

This material is based upon work supported by the National Science Foundation under Grant No. 1822816 and Grant No. 1918751. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Benjamin S. Bloom. 1971. Mastery Learning. In *Mastery Learning: Theory and Practice*, James H. Block (Ed.). Holt, Rinehart and Winston, 47–63.

[2] Dino Capovilla, Marc Berges, Andreas Mühling, and Peter Hubwieser. 2015. Handling Heterogeneity in Programming Courses for Freshmen. In *Proceedings of the 2015 International Conference on Learning and Teaching in Computing and Engineering (LATICE '15)*. 197–203. https://doi.org/10.1109/LaTiCE.2015.18

[3] Shana K. Carpenter, Nicholas J. Cepeda, Doug Rohrer, Sean H. K. Kang, and Harold Pashler. 2012. Using Spacing to Enhance Diverse Forms of Learning: Review of Recent Research and Implications for Instruction. *Educational Psychology Review* 24, 3 (2012), 369–378. https://doi.org/10.1007/s10648-012-9205-z

[4] William B. Davidson, William J. House, and Thomas L. Boyd. 1984. A Test-Retest Policy for Introductory Psychology Courses. *Teaching of Psychology* 11, 3 (1984), 182–184. https://doi.org/10.1177/009862838401100320

[5] Amy Diegelman-Parente. 2011. The Use of Mastery Learning With Competency-Based Grading in an Organic Chemistry Course. *Journal of College Science Teaching* 40, 5 (2011), 50–58.

[6] John Dunlosky, Katherine A. Rawson, Elizabeth J. Marsh, Mitchell J. Nathan, and Daniel T. Willingham. 2013. Improving Students' Learning With Effective Learning Techniques: Promising Directions From Cognitive and Educational Psychology. *Psychological Science in the Public Interest* 14, 1 (2013), 4–58. https://doi.org/10.1177/1529100612453266

[7] Stephen H. Edwards and Manuel A. Perez-Quinones. 2008. Web-CAT: Automatically Grading Programming Assignments *(ITiCSE '08)*. 328. https://doi.org/10.1145/1384271.1384371

[8] Sophie Engle and Sami Rollins. 2013. Expert Code Review and Mastery Learning in a Software Development Course. *J. Comput. Sci. Coll.* 28, 4 (April 2013), 139–147.

[9] Armando Fox and David Patterson. 2012. Crossing the Software Education Chasm. *Commun. ACM* 55, 5 (May 2012), 44–49. https://doi.org/10.1145/2160718.2160732

[10] James Garner, Paul Denny, and Andrew Luxton-Reilly. 2019. Mastery Learning in Computer Science Education. In *Proceedings of the Twenty-First Australasian Computing Education Conference (ACE '19)*. 37–46. https://doi.org/10.1145/3286960.3286965

[11] Mark Guzdial. 2020. *CS Teachers, It's (Past) Time To Learn About Race | blog@CACM*. Retrieved August 28, 2020 from https://cacm.acm.org/blogs/blog-cacm/245408-cs-teachers-its-past-time-to-learn-about-race/

[12] G. Herman, K. Varghese, and C. Zilles. 2019. Second-chance Testing Course Policies and Student Behavior. In *IEEE Frontiers in Education Conference (FIE '19)*. 1–7.

[13] Geoffrey L. Herman, Zhouxiang Cai, Timothy Bretl, Craig Zilles, and Matthew West. 2020. Comparison of Grade Replacement and Weighted Averages for Second-Chance Exams. In *Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER '20)*. Association for Computing Machinery, New York, NY, USA, 56–66. https://doi.org/10.1145/3372782.3406260

[14] E. Hill, P. M. Johnson, and D. Port. 2016. Is an Athletic Approach the Future of Software Engineering Education? *IEEE Software* 33, 1 (2016), 97–100.

[15] Mehdi Jazayeri. 2015. Combining Mastery Learning with Project-Based Learning in a First Programming Course: An Experience Report. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15)*. 315–318.

[16] Philip Johnson. 2019. Design and Evaluation of an "Athletic" Approach to Software Engineering Education. *ACM Trans. Comput. Educ.* 19, 4, Article 41 (Aug. 2019). https://doi.org/10.1145/3344273

[17] P. Johnson, D. Port, and E. Hill. 2016. An Athletic Approach to Software Engineering Education. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. 8–17.

[18] Chen-Lin C. Kulik and James A. Kulik. 1987. Mastery Testing and Student Learning: A Meta-Analysis. *Journal of Educational Technology Systems* 15, 3 (1987), 325–345. https://doi.org/10.2190/FG7X-7Q9V-JX8M-RDJP

[19] Noel LeJeune. 2010. Contract Grading with Mastery Learning in CS 1. *J. Comput. Sci. Coll.* 26, 2 (Dec. 2010), 149–156.

[20] Jason W. Morphew, Mariana Silva, Geoffrey Herman, and Matthew West. 2020. Frequent mastery testing with second-chance exams leads to enhanced student learning in undergraduate engineering. *Applied Cognitive Psychology* 34, 1 (2020), 168–181. https://doi.org/10.1002/acp.3605

[21] Craig E. Nelson. 1996. Student Diversity Requires Different Approaches To College Teaching, Even in Math and Science. *American Behavioral Scientist* 40, 2 (1996), 165–175. https://doi.org/10.1177/0002764296040002007

[22] Tia Newhall, Lisa Meeden, Andrew Danner, Ameet Soni, Frances Ruiz, and Richard Wicentowski. 2014. A Support Program for Introductory CS Courses That Improves Student Performance and Retains Students from Underrepresented Groups. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. 433–438. https://doi.org/10.1145/2538862.2538923

[23] Steven C. Shaffer and Mary Beth Rosson. 2013. Increasing Student Success by Modifying Course Delivery Based on Student Submission Data. *ACM Inroads* 4, 4 (Dec. 2013), 81–86. https://doi.org/10.1145/2537753.2537778

[24] Nicholas C. Soderstrom and Robert A. Bjork. 2015. Learning Versus Performance: An Integrative Review. *Perspectives on Psychological Science* 10, 2 (2015), 176–199. https://doi.org/10.1177/1745691615569000

[25] Craig Zilles., Matthew West., Geoffrey Herman., and T. Bretl. 2019. Every University Should Have a Computer-Based Testing Facility. In *Proceedings of the 11th International Conference on Computer Supported Education - Volume 1: CSEDU,*. 414–420. https://doi.org/10.5220/0007753304140420