Toward Effectively Reinforcing Test-Driven Development

Kevin Buffardi
Virginia Tech
2202 Kraft Drive
Blacksburg, VA 24060
kbuffardi@vt.edu

Stephen H. Edwards
Virginia Tech
2202 Kraft Drive
Blacksburg, VA 24060
edwards@cs.vt.edu

Abstract

Learning contemporary software development practices and acquiring good programming habits are valuable skills in computer science education. However, there are challenges in encouraging students' adherence to the incremental testing pattern involved in Test-Driven Development. We propose a model for an adaptive feedback system to observe and encourage incremental testing. Using data analysis of 6,953 submissions of students' programming assignments, we offer insight into students' development habits and their interaction with our adaptive feedback system. Based on our findings, we evaluate a model for measuring incremental testing and suggest approaches for improving reinforcement.

Categories and Subject Descriptors K.3.2 [**Computers and Education**]: Computer and Information Science Education; D.2.5 [**Software Engineering**]: Testing and Debugging.

General Terms Measurement, Human Factors, Verification.

Keywords *Test-driven development, unit testing, reinforcement, adaptive feedback, automated grading, instructional technology.*

1. Introduction

Test-Driven Development (TDD) is a process that emphasizes an incremental "test a little, code a little" approach [2]. TDD's popularity in industry [10] and the Accreditation Board for Engineering and Technology's (ABET) requirement to prepare students with "An ability to use current techniques, skills, and tools necessary for computing practice" [1] provide compelling reasons for including TDD in computer science curriculum. However, while some universities' introduced TDD into computer science classes, reports have yielded mixed results.

In professional settings, TDD is lauded for improving confidence and quality of code [3][7]. By practicing TDD in school, students should benefit from improving their code and testing as well as gaining practical experience in an established software engineering method. In addition, the metacognition involved in writing tests before their corresponding solutions may encourage students to engage "reflection-in-action" in place of weaker "trial-and-error" strategies [9].

However, learning TDD also comes with its challenges. In particular, multiple studies have found that despite TDD's benefits, students and novice programmers often require extra motivation to adopt test-first strategies [5][11][13][15]. However, little research exists on approaches for encouraging students to adopt the software engineering method.

Methods in general can be difficult to teach and evaluate since traditional school assignments only involve grading a single representation of each student's completed assignment. Without insight into processes and behaviors exhibited over time, it is difficult to observe adherence to methods. However, with the introduction of automated grading systems such as Marmoset [14] and Web-CAT [8], students can submit their work multiple times while they improve their work. With multiple submissions of programming assignments, we can begin to observe students' behaviors and patterns of development.

Unfortunately, there are no practical means for evaluating test-first development on programming assignments since tests without their corresponding solutions would be unreasonable to evaluate. However, while we might not be able to observe test-first behavior, students can at least demonstrate incremental testing by submitting corresponding tests and solutions.

With each submission of their work, we gather a snapshot of students' development process. In addition, each time a student interacts with an automated grading system, we have the opportunity to evaluate her work and apply teaching interventions to improve her learning. With automated interventions during students' development process, we have a unique opportunity to measure and influence students' behavior mid-assignment. By encouraging students to test early and incrementally, we hope to help students produce higher quality code and testing while also developing practical TDD experience and appreciation.

2. Related Work

Melnik and Maurer acknowledged that adopting Agile methods may be more challenging in an academic setting than it is in industry. Consequentially, they surveyed students' opinions of different aspects of eXtreme Programming (XP), including Test-Driven Development (TDD). They found generally positive views of all aspects of eXtreme Programming from 240 respondents, representing a variety of demographics with differing degrees of experience and exposure to XP.

Those students who believed following XP improved the quality of their code attributed the improvement to both pair programming and TDD. In addition, they discovered a weak positive correlation between attitudes toward TDD and students' ages. However, they also found that some students struggled to think with a test-first approach. They reasoned this difficulty may be due to TDD "almost like working backwards" by drawing attention to documenting design early through writing unit tests [13].

Similarly, Janzen and Saiedian compared the opinions and acceptance of TDD between novice and mature developers in computing courses. They found that mature developers are more willing to accept TDD. Furthermore, students were significantly more likely to choose to follow TDD in the future after having tried it [11].

To aide teaching TDD, Spacco and Pugh leveraged Marmoset [14], a rich submission system that includes public (made availa-

ble to the students) and release tests (obscured from students). Students benefit from receiving prompt, online feedback including how their code performs against the tests. While the specifics of release tests are obscured from the students, they receive the name of the first two test methods that fail on a submission to offer some guidance in identifying the problem. Marmoset also provides measurements of test performance and coverage. Despite the emphasis and prompt feedback on testing, Spacco and Pugh recognize that many students still favor a "test-late mentality of writing their implementation and then testing it at the very end" and call for a need to design incentives to motivate students to test early [15].

Positive reinforcement can be a powerful tool in scaffolding new behaviors. Schedules of reinforcement have encouraged target behaviors in both games and learning environments [12]. Linehan suggests a model for leveraging Applied Behavior Analysis to motivate target behaviors through a process of measuring performance, analyzing performance, presenting feedback, and defining a rewards schedule that coordinates with the target behavior. Likewise, we have the unique opportunity to measure, analyze, and use positive reinforcement to influence students' development processes.

Meanwhile, we have been teaching TDD in CS1 and CS2 courses. In a preliminary study of a five-year data set of snapshots of students' work, we found positive correlations between indicators of incremental testing and consequential outcomes. Specifically, we identified two measurements of quantity of testing average test statements per solution statements (TSSS) and average test methods per solution method (TMSM) with small but statistically significant correlations with functional correctness and test coverage. Similarly, average test coverage across all snapshots for an assignment was positively correlated with final functional correctness [6]. However, despite its advantages, we also witnessed some students who resisted adhering to TDD.

In a separate study, we investigated students' attitudes toward the test-first and incremental unit testing aspects of TDD [5]. Similar to reports in related literature, we discovered that students generally appreciated the value of testing but were apprehensive to adopt test-first habits. While students valued an incremental unit-testing approach, most students did not follow strict test-first procedures. Likewise, we identified a close relationship between students' perception of how helpful these aspects of TDD are and how likely they are to adhere to them.

This relationship is likely reciprocal in that expecting TDD to help should make students more likely to adhere; likewise, adhering to TDD should advocate students' appreciation of its benefits. However, by their own reporting, most students did not persistently write tests in small increments nor did they test first. For these reasons, we recognized a need to better understand students' development processes and investigate approaches to encouraging adherence to TDD.

3. A Model for Adaptive Feedback

To observe students' development processes, it is necessary to gain insight into changes in their work over time. By using Web-CAT [8]—an automated grading system—to collect student submissions and provide rapid evaluation of their performance, students are encouraged to submit several versions of their work as they refine their assignments. Among other features, Web-CAT evaluates students' code on its correctness and coverage. Correctness is determined by the percent of instructor-provided tests (obscured from students) successfully passed by the student's solution. Coverage is determined by the amount of solution code evaluated by the students' own unit tests. Upon submitting their work, students promptly receive results of their correctness and coverage scores. Students may submit their work unlimited times, without penalty, until the assignment deadline.

With each time students submit and receive feedback, there are opportunities to assess their adherence to incremental testing methods and trigger interventions to encourage the desired behavior. Ideally, incremental testing would be demonstrated by maintaining high (at or near 100%) coverage while the correctness gradually increases. Consequently, we designed the system to reinforce this behavior and to correct students who deviate from it.

Our adaptive feedback system supplements Web-CAT's correctness and coverage assessment by monitoring progress from one submission to the next. Students receive positive reinforcement through images and brief messages acknowledging improvements in their solution and/or testing, as shown in Figure 1. When their submissions do not demonstrate improvement, the feedback encourages them to improve their testing by offering additional incentives. In particular, students receive hints about how to improve their solution as a reward for improving the thoroughness of their testing. Figure 1 illustrates reinforcing feedback with two hints displayed.

The system adaptively caters hints to help students correct problems with their solutions that are identified by failed instructor tests. Each instructor unit test includes a hint message to provide some guidance about why it failed. Closely related instructor tests may generate identical hint messages, but duplicates are combined and the collection of hints is sorted so precedence is given to hints with more occurrences.

Students earn their first hint from their first submission that demonstrates some progress on both the solution and testing with non-zero correctness and coverage scores. To earn additional hints on subsequent submissions, students have to meet a minimum threshold of coverage (85% or greater) to demonstrate they are testing their solution substantially. Given the minimum coverage is met, students can earn hints by either maintaining 100% coverage and making changes to their solution code, or by improving their coverage over the previous submissions. Following this

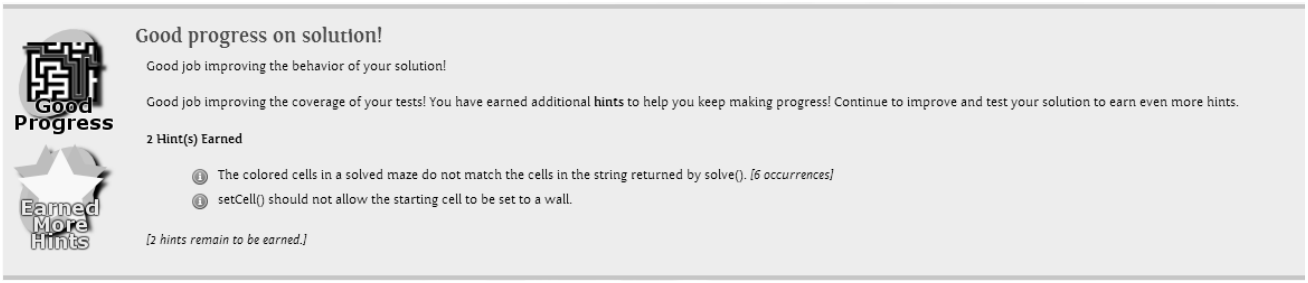

Figure 1. The Web-CAT plugin displays adaptive feedback with positive reinforcement and hints.

model, students are required to begin testing in early submissions and continue to meet progressively higher coverage requirements as they progress.

If a student earns hints on sequential submissions, she may receive the same hints in both submissions' feedback. On its face, this approach may not seem to reward the student for earning hints. However, it allows the student to track the flaws in her assignment. For example, if a previously seen hint is replaced by a new hint, one may falsely conclude that the bug that generated the hint has been resolved. Instead, earned hints are only dismissed once their corresponding instructor test passes.

If students deviate from the incremental testing process, they do not receive hints on their submissions until they demonstrate sufficient coverage again. The system also controls for potential attempts to manipulate progress measurements into providing unearned hints. For example, the system records the highest correctness score achieved so far to prevent students from deleting or sabotaging their solution in one (worse) submission only to give the illusion of improvement by reverting to the previous (better) solution in the next submission.

4. Method

4.1 Intervention Experiment

As described in detail in our previous publication [4], we evaluated the impact of the adaptive feedback system, by comparing Web-CAT assignment data from two semesters of a CS2 course, Software Design and Data Structures. The first semester, students submitted their assignments to Web-CAT with the adaptive feedback system disabled. Instead of earning hints by incrementally testing, students received up to three hints at a time when instructor tests failed. Instead of requiring certain behaviors to earn hints, these students received their hints for "free."

The second semester used the adaptive feedback system for programming assignments. At the conclusion of the semester, we compared measurements of incremental testing between the first (control) and second (experimental) semesters. Average test methods per solution method (TMSM) measured the relative quantity of test code while average coverage measured the quality or thoroughness of tests throughout the development process [6]. High values for either measurement indicates strong incremental testing patterns. Final correctness and coverage scores for each assignment were also analyzed to investigate potential effect of the adaptive feedback intervention on the assignments' outcomes.

Participation was voluntary and only data from students who consented to have their data analyzed were included. 78 of 129 (60%) enrolled students from the experimental semester and 87 of 130 (66%) from the control semester consented. Submissions with incomplete or errant information (from cancelled or non-compiling submissions) were excluded.

In addition to investigating the adaptive feedback system's impact on code and testing quality, we also surveyed students to gauge their attitudes and perceptions of Test-Driven Development. Along with the survey, questionnaires measured students' fear of negative evaluation and anxiety when performing programming assignments [4].

4.2 Model Behavior Analysis

As a system designed both to measure software metrics and to influence students' behavior, the adaptive feedback system requires both effective technical and social components. The following factors are necessary to affect change in programming assignment outcomes:

1. The system accurately measures and identifies development patterns that indicate incremental testing
2. The system matches observed behaviors with corresponding feedback
3. The adaptive feedback provides sufficiently valuable incentives to motivate student behavior
4. The behavior is executed effectively to impact measurable improvement in code quality

A failure to meet any one of these parts will likely disrupt any chance for measurable change. However, by investigating each objective independently, we can better evaluate the nuances of a system serving both engineering and social roles. To examine each objective, we performed post-hoc analysis on 6,953 submissions from the experimental semester with the aim to identify patterns that indicate: reliability in measuring incremental testing; frequency and scale of incentives corresponding to measured behavior; and changes in testing strategies and code quality.

The analysis included the development of 319 programming assignments, each averaging approximately 22 submissions (M=21.80, sd=16.55). For each submission, we recorded several measurements, including time of submission, amount of code, solution correctness, and testing coverage. We omitted assignments without multiple submissions for a student because a single submission cannot demonstrate a process of development. Since the number of submissions and amount of time spent on development varied for each assignment, we concentrated on notable submission milestones within each assignment:

- Initial: The first submission that earned an initial hint by achieving non-zero correctness and/or coverage scores
- Threshold: The earliest submission that surpasses the 85% coverage threshold to earn additional hints beyond the initial milestone
- Maximum: The earliest submission where the student first reaches the highest correctness score attained for that assignment

The initial milestone provides our earliest insight into each student's work. Meanwhile, the threshold milestone identifies where in the student's development process that he first has substantial testing. The maximum milestone helps distinguish late development patterns. Once a student reaches his maximum correctness, he may conclude his work on the assignment by making last changes to style and documentation, or he may demonstrate late testing practices. To account for varying amount of time spent on an assignment, the time recorded for each milestone reflects the time elapsed on to a normalized scale from first (0.0) to last (1.0) submission for that assignment. From this stage forward, we refer to an assignment's normalized time elapsed as **relative work-time**. Table 1 summarizes the mean, standard deviation, and Shapiro-Wilk test, where a value below 0.01 rejects the hypothesis of normal distribution. Since $W<0.0001$ for each metric, the Mann–Whitney–Wilcoxon test for repeated measures was used to compare means from non-parametric samples.

Table 1. Mean, standard deviation, and Shapiro-Wilk test for normality of milestone metrics.

	Mean	sd	W
No. of submissions	21.80	16.55	*< 0.0001**
No. of submissions earning hints	10.01	9.60	*< 0.0001**
Final Correctness	0.8251	0.2789	*< 0.0001**
Final Coverage	0.9383	0.1691	*< 0.0001**
Initial Time	0.1353	0.2722	*< 0.0001**
Initial Coverage	0.6375	0.3064	*< 0.0001**
Threshold Time	0.4519	0.3607	*< 0.0001**
Threshold Coverage	0.9198	0.0792	*< 0.0001**
Maximum Time	0.7878	0.3230	*< 0.0001**
Maximum Coverage	0.9386	0.1746	*< 0.0001**

To begin characterizing different testing strategies, we explored when in their development processes students first demonstrated substantial testing: the threshold milestone. Figure 2 illustrates the distribution of the relative time of the threshold milestone within each assignment. The distribution shows two major peaks: one where students tested substantially within the first 10% of their submission time and another where students did not test substantially until within the last 20% of their work. Consequently, we grouped students according to the time of their threshold milestone.

Students with a threshold milestone at or below 0.2 belong to the Early group. Students with a threshold at or above 0.8 belong to the Late group and everyone in between belongs to the Intermediate group. Approximately 32% of assignments are in the Early group, 34% in the Intermediate group, and 23% in the Late group. 11% of assignments never reached the 85% threshold for coverage so we grouped them as Neglectful testers. We also identified that only 3% of students were in the Early group for all of their assignments; 1% of students were in the Intermediate group for all assignments, and 1% of students belonged to the Late group for all assignments. The remaining 95% of students belonged to at least two different groups across different assignments. It is safe to conclude that students practice a variety of testing strategies for different assignments.

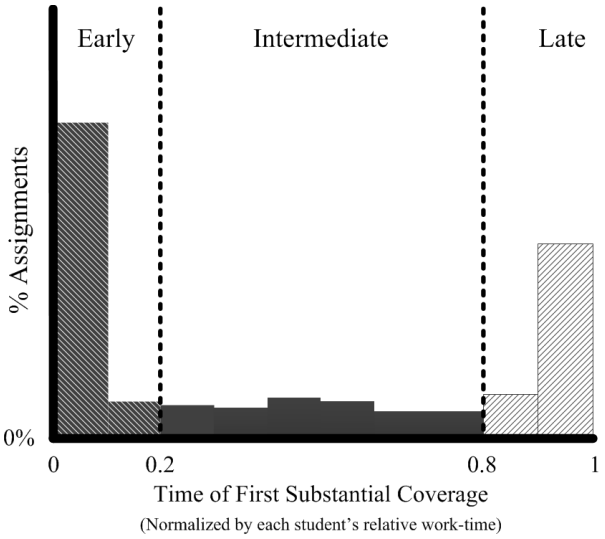

Figure 2. Grouping and distribution of when students first achieve at least 85% coverage within their relative work-time

5. Results

5.1 Intervention Experiment

The results described in "Impacts of Adaptive Feedback on Teaching Test-Driven Development" outline a series of hypotheses tested that found no measurable difference in the control and experimental groups. Students exposed to the adaptive feedback system demonstrated no significant differences in either correctness or coverage. Likewise, there were no significant differences in other metrics for measuring incremental testing, such as average coverage and average TMSM.

The surveys provided mixed results where the control group considered unit testing more helpful than the experimental group. However, the experimental group rated the helpfulness of test-first development higher than the control group. Lastly, although the adaptive feedback could have potentially increased anxiety for those adverse to negative evaluation, the experimental group showed no significant difference from the control group's fear of negative evaluation or project anxiety.

While it is disappointing to find that the adaptive feedback system did not affect improvements in TDD adherence, perceptions, or outcomes, it is important to note that despite earning fewer hints, the adaptive feedback system did no apparent harm either. Even more importantly, the null results indicate a need to investigate why the model failed to affect student behavior. The experimental results magnified the importance of data mining for clues about students' interaction with the system.

5.2 Behavior Model Analysis

After categorizing each assignment into Early, Intermediate, Late, and Neglectful testing groups, we compared the each group's trends. Figure 3 illustrates the timing and coverage patterns for initial, threshold, and maximum milestones.

The Early group averaged submitting fewer times (M=23.16, sd=15.96) than the Intermediate group (M=27.06, 18.19) approaching significance (p=0.09). However, the Early group earned hints on a significantly higher percentage (p<0.0001) of their submissions (M=0.6722, sd=0.2275) than the Intermediate group (M=0.5025, sd=0.2755). Likewise, Intermediate earned hints more often (p<0.001) than Late (M=0.3591, sd=0.2529) and Late more often (p<0.001) than Neglectful (M=0.1984, sd=2022).

As illustrated above in Figure 3, the Early group tended to earn hints earlier in their development as well. The Early group reached the Threshold milestone (M=0.0553, sd=0.0536) significantly sooner (p<0.0001) in their development than the Intermediate (M=0.4966, sd=0.1737), and Late (M=0.9280, 0.0641) groups. Likewise, Intermediate reached the Threshold significantly sooner in their development process than Late (p<0.0001).

From the clear differentiation in the temporal proximity and frequency of earning hints between groups, we can conclude that the adaptive system is successfully identifying different levels of adherence to incremental testing. This observation supports the first of four system objectives, as previously stated: "The system

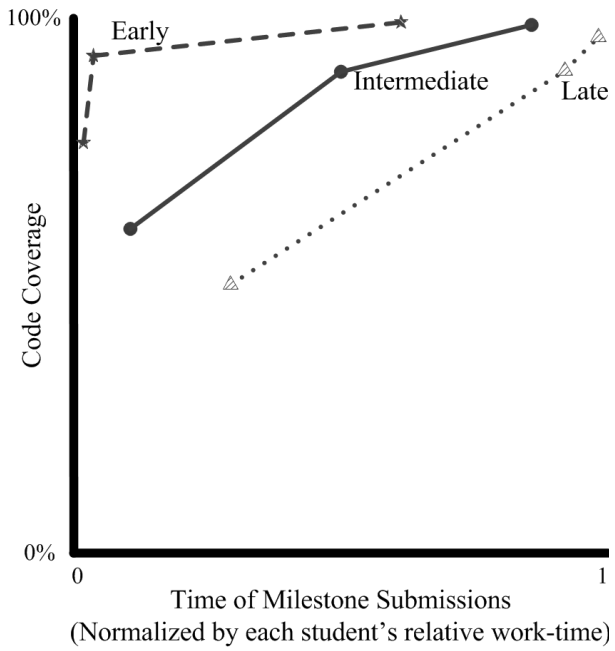

Figure 3. Progress in code coverage between groups of different development patterns at *initial*, *threshold*, and *maximum milestones*.

accurately measures and identifies development patterns that indicate incremental testing."

By reviewing the feedback that students received, we can also confirm that the system accurately observed progress in correctness and coverage and prepared hints corresponding with failed instructor tests. Therefore, the second objective is also satisfied: "The system matches observed behaviors with corresponding feedback." However, since the adaptive feedback system did not accomplish its goal of encouraging incremental testing, we can investigate potential problems with the two remaining factors:

3. The adaptive feedback provides sufficiently valuable incentives to motivate student behavior
4. The behavior is executed effectively to impact measurable improvement in code quality

If we trust our previous findings [4][5], then we should be confident that Test-Driven Development improves test and solution quality. Therefore, it appears as though the trouble with the adaptive feedback is that the incentives and reinforcement are not motivating the target behavior.

6. Discussion

Although we demonstrated that the Early group—who showed the closest adherence to early and incremental testing—received hints earlier and more often in their development process, it is possible that the difference in the incentives they received were insufficient to influence measurable behavior change. A potential remedy would be to amplify the value of the incentives by accumulating hints. If sequential submissions each earned two hints, by accumulating the hints earned, the student would receive four hints on their latter submission instead of two. In Figure 4, we illustrated the number of hints earned by each group if the system worked exactly the same but accumulated earned hints.

In this hypothetical scenario, the Early group (M=27.93, sd=20.83) would accumulate significantly more ($p<0.05$) hints than the Intermediate group (M=23.26, sd=19.23). Likewise, the

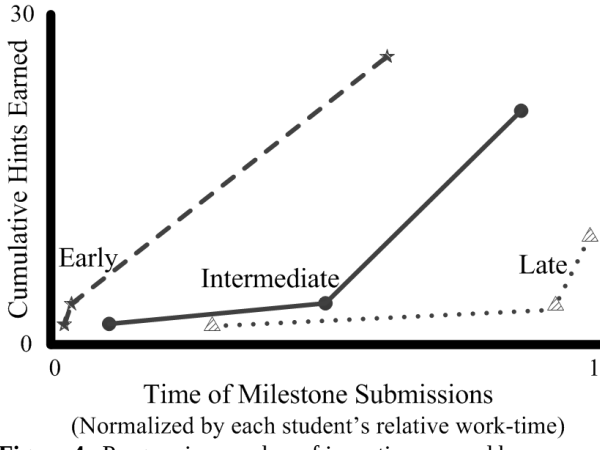

Figure 4. Progressive number of incentives earned by accumulating hints at development milestones.

Intermediate group would accumulate significantly more ($p<0.0001$) hints than the Late group (M=9.59, sd=8.86), which in turn would accumulate more hints ($p<0.0001$) than the Neglectful group (M=1.00, sd=0.00). However, we do not predict that accumulating hints will solve the problem alone. In fact, we recognize distinct problems with the accumulating hints approach. Firstly, while groups eventually accumulate a significant different number of hints over the period of their development, the difference in early incentives are still negligible. Are three accumulated hints much more influential than two or one when members of the different groups are exhibiting different testing behaviors?

Secondly, by accumulating hints over time, the incentives do not really pay-off until late in development. That means that Early testing strategies do not really receive substantial reinforcement (compared to the other groups) until well into their development. Such delayed reinforcement is not likely to be influential. Furthermore, as students reach later in their development processes, the hints earned likely offer diminishing returns.

The further students are in their development, typically their correctness scores are higher as well. Higher correctness score indicates that the code is failing fewer tests and consequently has fewer hints to generate. Even if an Early tester has accumulated 23 hints when she is most of the way through her development, it is unlikely that she is simultaneously failing enough tests to generate 23 hints. Lastly, hints are likely most helpful early in development so that problems in the solution can be identified and fixed when the solution is less complete.

Our adaptive feedback system purposely rewarded the first submission where students demonstrated at least a little coverage with the hopes that it would reinforce testing early. However, the flaw with that design is it rewards Intermediate and Late testers with negligible coverage with the identical incentive as an Early tester who may already have near 100% coverage. While the initial incentive was designed with good intentions, it is likely detrimental to the goal of reinforcing early, thorough testing more than lackluster testing.

Consequently, our diagnosis of the adaptive feedback system is that its incentives need greater differentiation according to the measured behavior. Reinforcing early behavior may be the best opportunity to convey value in incremental testing. Therefore, it is important to make a strong impact particularly on early feedback. Setting low standards for incentives on early submissions is especially counterproductive.

In conclusion, the adaptive feedback system shows promise in measuring incremental testing patterns. However, new reinforcement schedules and incentives need to be explored to improve the likelihood of influencing student adherence to target behaviors. With future research, we plan on experimenting with progressively stronger incentives for early submissions. Furthermore, students may benefit from receiving salient, concrete goals for improving their testing. As we discover better methods for reinforcing behavior, adaptive feedback has potential as a novel approach to automatically measuring and encouraging development methods.

References

[1] ABET (2013). "Criteria for Accrediting Computing Programs, 2013-2014." Retrieved May, 2014, from http://www.abet.org/

[2] Beck, K. (2003). Test-Driven Development by Example, Addison Wesley.

[3] Bhat, T. and N. Nagappan (2006). Evaluating the efficacy of test-driven development: industrial case studies. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. Rio de Janeiro, Brazil, ACM: 356-363.

[4] Buffardi, K. & Edwards, S.H. (2013) "Impacts of Adaptive Feedback on Teaching Test-Driven Development." Proc. of SIGCSE, Denver, Colorado.

[5] Buffardi, K. and S. H. Edwards (2012). Exploring influences on student adherence to test-driven development. Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. Haifa, Israel, ACM: 105-110.

[6] Buffardi, K. and S. H. Edwards (2012). "Impacts of Teaching Test-Driven Development to Novice Programmers." International Journal of Information and Computer Science 1(6): 9.

[7] Canfora, G., A. Cimitile, et al. (2006). Evaluating advantages of test driven development: a controlled experiment with professionals. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. Rio de Janeiro, Brazil, ACM: 364-371.

[8] Edwards, S. H. "Web-CAT." 2013, from https://web-cat.cs.vt.edu.

[9] Edwards, S. H. (2004). "Using software testing to move students from trial-and-error to reflection-in-action." SIGCSE Bull. 36(1): 26-30.

[10] Fraser, S., D. Astels, et al. (2003). Discipline and practices of TDD: (test driven development). Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Anaheim, CA, USA, ACM: 268-270.

[11] Janzen, D. S. and H. Saiedian (2007). A Leveled Examination of Test-Driven Development Acceptance. Proceedings of the 29th international conference on Software Engineering, IEEE Computer Society: 719-722.

[12] Linehan, C., B. Kirman, et al. (2011). Practical, appropriate, empirically-validated guidelines for designing educational games. Proceedings of the 2011 annual conference on Human factors in computing systems. Vancouver, BC, Canada, ACM.

[13] Melnik, G. and F. Maurer (2005). A cross-program investigation of students' perceptions of agile methods. Proceedings of the 27th international conference on Software engineering. St. Louis, MO, USA, ACM: 481-488.

[14] Spacco, J. "Marmoset." 2013, from http://marmoset.cs.umd.edu/.

[15] Spacco, J. and W. Pugh (2006). Helping students appreciate test-driven development (TDD). Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. Portland, Oregon, USA, ACM: 907-913.