Student Adherence to Test-Driven Development and Evaluating Methods in Computer Science

Kevin Buffardi
Virginia Tech
114 McBryde Hall (0106)
Blacksburg, Virginia
+1 540-231-5723
kbuffardi@vt.edu

1. PROBLEM AND MOTIVATION

Among other objectives, computer science curricula aim to familiarize students with contemporary techniques in software development. Some of these techniques include methods such as Test-Driven Development (TDD) and other software engineering processes.

By practicing conventional techniques, students may produce higher-quality work. However, the beneficial outcomes of practicing software development processes should not end at improved schoolwork. Instead, students should also gain confidence and competence in applying these methods in a practical setting. Accordingly, ABET accreditation requires evidence that students acquire "An ability to use current techniques, skills, and tools necessary for computing practice" [1].

Assessing students' application of methods is another challenge in computer science education. Evaluating programming assignments usually entails reviewing only the final deliverable of students' development. Unfortunately, the final deliverable alone may be inadequate in indicating either how well they followed a method, or even if they adhered to the procedures involved at all.

For example, a student's submission may include software tests, but with no additional information, it is uncertain if the student followed TDD's incremental, test-first approach to develop those tests. Consequently, it is difficult to assess students' aptitude at following Test-Driven Development. Furthermore, without evaluation or feedback on applying TDD, students may not feel compelled to follow the technique. Without incentives to adhere to methods taught in class, students may avoid exercising these methods to the detriment of both practical experience and quality of their work.

At Virginia Tech, introductory computer science courses introduce Test-Driven Development (TDD) early and continue to reinforce it throughout the course. TDD is a development strategy that emphasizes writing unit tests – software tests that validate requirements in small parts, often at the scope of individual functions. In addition, TDD advocates a test-first approach: writing a unit test should precede implementing the solution for its corresponding functionality [3]. The consequence of TDD is incremental development of validated functionality, as illustrated in Figure 1.

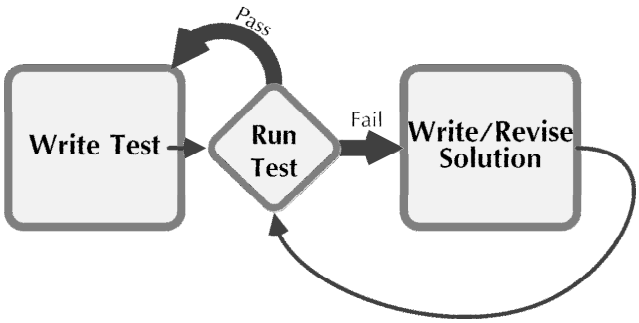

Figure 1. The Test-Driven Development process, emphasizing writing unit tests prior to solution code

Agile methods, such as Extreme Programming (XP), helped popularize TDD in industry [13]. Therefore, exercising TDD methods will help students develop practical skills. However, despite lessons describing and encouraging TDD, we anecdotally observed some students not adhering to TDD principles. Accordingly, multiple academic studies at different schools also identified students' reluctance in adopting TDD [14][17][20].

To address the issue of student adherence to TDD, we face three distinct challenges. First, we need to understand students' attitudes toward adopting TDD. Secondly, we need greater granularity in observing students' software development process to assess their performance of TDD. Finally, we need to synthesize these observations with viable solutions for influencing their behavior. In this paper, I describe how I addressed the first two challenges. Along with my findings, I identify potential solutions (and forthcoming work) for encouraging student adherence to TDD.

2. BACKGROUND AND RELATED WORK

Many leading development organizations in industry adopted Test-Driven Development (TDD), particularly within the scope of Agile development techniques [13]. Additionally, empirical research on TDD use in industry suggests that it improves both the quantity and quality of code and testing [4][6]. Likewise, several academic studies have identified benefits of TDD.

Surveys of TDD education in universities have shown similar results to those in industry: while TDD may increase workload, quality of testing and code improves [8]. One particular study found when pairing TDD with Web-CAT, an automated grading tool, students produced programs with 45% fewer defects per line of code compared to students in a class not using TDD [12]. Moreover, Edwards reasons that by following TDD, students' analytical skills improve by "reflection in action" rather than

depending on trial-and-error strategies [11]. Corresponding with its popularity, several tools have are available to support implementing and teaching TDD.

Both industry and academia use JUnit [15], a unit-testing framework for Java, as a *de facto* standard. Educators have also developed tools to assist TDD in the computer science curriculum. Both Web-CAT [21] and Marmoset [19] are automated grading tools that provide feedback based on instructor-written reference tests in addition to students' own tests. Contrastingly, ComTest uses a simple macro language for writing shorter and easier unit tests than JUnit [16]. However, Lappalainen, *et al.* also acknowledge ComTest's limitation that – like the other test-driven learning tools – while unit testing may be required in assignments, ComTest does not strictly enforce a test-first approach.

Meanwhile in academia, educators have integrated TDD into computer science curricula and observed beneficial results similar to those in industry [7][10]. However, studies also recognize a challenge of promoting student acceptance of TDD. Barriocanal identified a particular need to motivate students to write tests when they are new to TDD [2]. Other academic studies also acknowledged a reluctance to adopt TDD, particularly by younger and less experienced programmers [14][17][20]. Specifically, Spacco and Pugh described a need to use incentives to promote a "test-first mentality" or else students will wait until the end of development to test (i.e. test-late approach).

3. APPROACH AND UNIQUENESS

Current research lacks thorough discussion of student affect—or their attitudes and emotional valence—with regards to how it influences their behaviors, particularly in adhering to Test-Driven Development (TDD). To gauge students' affect, we conducted a survey at the end of the semester of a CS2 (software design and data structures) class. The survey addressed student attitudes and self-reported behaviors including: perceived importance and self-assessed strength of skills; helpfulness and frequency of adherence to TDD behaviors; and intent to follow TDD in the future [5].

One hypothesized explanation for non-adherence to TDD was that some students may have strong, negative reactions to feedback—such as failing software tests—while in the development process. Those students may then delay writing tests until the end to avoid negative feedback as long as possible. This behavior would be particularly concerning with classes that use automated grading systems such as Web-CAT or Marmoset because these systems may provide feedback for both failed student- and instructor-written tests. Accordingly, I investigated students' affect using the Brief Fear of Negative Evaluation Scale (BFNES) [18] and Westside Test Anxiety Scale (WTAS) [9]. Both scales are brief, validated questionnaires. WTAS was adapted to specifically address anxiety on programming assignments [5].

In addition to gaining insight into student affect, we need to observe their software development process beyond just reviewing the final deliverable. Requiring intermediate deliverables would offer more details on students' development processes. However, assessing multiple deliverables would increase the workload of the teaching staff along with each additional. Instead, it would be favorable to automate data collection on the development process without increasing staff workload.

Accordingly, I used Web-CAT to gather multiple milestones in each student's process of developing programming assignments. Students submit their work on-line to Web-CAT at their convenience. Web-CAT provides feedback to the students including: static analysis of their code, test coverage (the percent of statements in their solution that have been tested using students' own JUnit tests), and solution correctness (evaluated by undisclosed instructor-written tests). Hints also accompany instructor-written tests to help students improve their solution. Students may revise and resubmit their programming projects as many times as they wish without penalty [10]. Still, only students' final submissions are considered for grading so as not to discourage submitting work-in-progress.

By gathering multiple submissions per student for every assignment, I gain greater granularity in observing students' individual software development processes. Progress in work over time demonstrates behaviors where a final deliverable alone cannot. In particular, we were able to observe adherence to test-early and test-late strategies along with their consequences. Additionally, we investigated signs of testing in small units (as advocated in TDD) instead of in large portions.

For a thorough analysis of students' development habits, I analyzed two years (four academic semesters) worth of Web-CAT submissions for introductory computer science classes that taught TDD. To discern student adherence to TDD, I concentrated on six measurements:

- NCLOC: The number of non-blank, non-comment lines of code, separated by test code and solution code, for each student submission on every project.
- Test:Solution Method Ratio: The number of test methods relative to the number of solution methods in a particular submission.
- Final Correctness: The correctness of the solution code in each student's last submission for a project, as determined by instructor-written tests.
- Final Coverage: The percent of statements covered by tests on each student's last submission for a project.

NCLOC describes the quantity of code and is useful in indicating how much test code has been developed at the time of each submission. By paying particular attention to *Test NCLOC* in early and late submissions, we can infer test-early or test-late strategies. *Test:Solution Method Ratio* provides insight into the development of unit tests. When using JUnit to test Java code, the smallest unit of testable code is usually a method. Accordingly, unit tests are usually encapsulated within test methods that correspond to the solution methods they validate. Therefore, a *Test:Solution Method Ratio* approaching 1:1 suggests adherence to unit testing.

Final Correctness and *Coverage* are objective, qualitative measurements of student outcomes on each project. By examining relationships between these outcomes and the previously mentioned behavioral observations, one may draw correlations between behavior and outcome.

4. RESULTS AND CONTRIBUTIONS

The results from student surveys at the conclusion of a semester revealed several insights and statistically significant relationships between student attitudes and behaviors. The study found positive correlations between how helpful students perceived different behaviors and how often they demonstrated those behaviors. This relationship held true for both general behaviors—such as beginning work as soon as it is assigned—and for Test-Driven Development (TDD) behaviors, including both following test-first approach and testing in small units. It is especially notable that ratings for both *helpfulness* and *adherence* to test-first behavior were lower than those for test-last [5]. This finding suggests that students do not value the test-first approach and consequently may not strictly follow TDD even if they value and adhere to unit testing.

However, the survey results suggested that those who adhere to TDD also appreciate its contribution to the development process. Frequency of adherence to TDD positively correlated with how much students thought that TDD helped their work in regards to: time management, problem solving, attention to detail, writing the problem's solution, as well as writing tests [5]. From these results, we may hypothesize that the greatest barrier facing TDD education is in its initial adoption. That is, if we can convince students to follow TDD despite their preliminary reluctance, they will likely appreciate its value. Given the demonstrated relationship between attitude and behavior, once students gain appreciation of the TDD technique, they will be more likely to adhere to it.

The study also investigated the role that anxiety may have either on following TDD or on using an automated grader (Web-CAT). However, the survey yielded no observable relationships [5]. In retrospect, I considered the Yerkes-Dodson Law [22]: for optimal performance on a task, a certain amount of stress or arousal is required; however, either too much or too little stress has detrimental effects on performance. Consequently, it is reasonable to expect anxiety to have beneficial or detrimental effects on software development depending on a number of factors, including individual psychological differences and difficulty of the programming assignment. I intend on extending my research on the role of anxiety with particular attention to its relationship with quantitative measurements of performance (rather than relying on self-reported data on behaviors).

To supplement the study of student attitudes, I analyzed a large set of data collected via Web-CAT to examine students' software development processes. With two years of data of multiple courses each with several programming assignments, I analyzed data from over 1700 student projects. On average, each student submitted to Web-CAT approximately thirteen times per assignment.

I found the average *Test:Solution Method Ratio* (see previous section for measurement explanations) improved from 1:1.8 to 1:1.4 when comparing the first submissions to last submissions for an assignment. On the first submission, students averaged 15.37 test methods (s.d.=19.75) to 27.48 solution methods (s.d.=25.92). On final submissions, students averaged 21.68 test methods (s.d.=21.30) to 30.05 solution methods (s.d.=27.54). The individual variation demonstrated by the large standard deviations may be explained—at least in part—by the variety in both assignment difficulty and scope. However, the greater increase in test methods than solution methods over time reinforces the notion that students may follow the principle of unit testing but while following a test-late strategy. The former is in compliance with TDD while the latter violates TDD's test-first principle.

Multiple measurments supported conclusions made by previous studies of TDD in industry and academia. The amount of *Test NCLOC* in a student's first submission for an assignment provides a general measurement of how much work is devoted to testing early in development. I found a postive correlation between *Test NCLOC* of first submissions to assignments and the *Final Solution Correctness* (ρ=0.1344, $p<0.0001$) and *Final Test Coverage* (ρ=0.2981, $p<0.0001$). While the strength of the correlations are not especially strong, the correlations are worth noting provided the beneficial outcomes even from such a general measurement as *NCLOC*.

More significantly, the *Test:Solution Method Ratio* (*TSMR)* of the first submission on an assignment had an even stronger relationship with positive outcomes. To compare to outcome measurements, *TSMR* was represented as a decimal test-methods-per-solution-method (where 1.0 would indicate the same number as test methods as solution methods, values less than 1 would indicate fewer test methods and greater than 1 would indicate fewer solution methods). I found a moderately positive correlation with *Final Solution Correctness* (ρ=0.3310, $p<0.0001$) and strong positive correlation with *Final Test Coverage* (ρ=0.6059, $p<0.0001$). A greater *TSMR* on the first submission suggests both unit testing and test-early strategies, both aspects of TDD. The relationship I found helps validate claims of TDD improving quality of code.

Since *TSMR* on the first submission offers a richer insight into a student's adherence to both principles of TDD, I investigated its relationship with performance outcomes thoroughly. When I grouped students by whether or not they achieved 100% *Final Test Coverage*, I found that those who achieved 100% had a significantly greater first-submission *TSMR* (M=0.66) than those who did not achieve 100% (M=0.55, $p<0.001$). Figure 2 shows the distribution of the groups. However, I considered that these outcomes may have been unduly influenced by the best students.

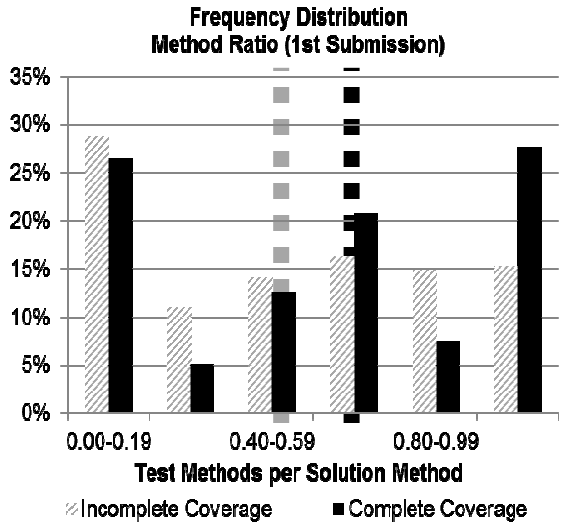

Figure 2. Frequency distribution of first submission *test-methods-per-solution-method*, with color coded means represented by vertical dashes

It was possible that the best students may have done just as well whether or not they followed TDD, but adhered to the principles because they were encouraged to by the instructors. To investigate the possibility of this confounding variable, I categorized students into three groups. The first group, labelled "Overachievers," were students who earned a grade of 80 (out of 100) or above on every programming assignment in the course. "Slackers" were the second group who earned below 60 on every programming assignment. The remaining students were placed in the "Mixed" outcomes group.

Independently, each of the three groups demonstrated positive correlations when considering first submission *TSMR* and the correctness and coverage outcomes. The following table shows the positive correlations for each group.

Group	Final Solution Correctness (ρ, p)	Final Test Coverage (ρ, p)
Overachievers	**0.1830, <.05***	**0.2931, <.001***
Mixed	**0.1511, <.0001***	**0.2998, <.001***
Slackers	**0.6886, <.0001***	**0.4805, <.001***

Table 1. Positive correlations within groups between *Test:Solution Method Ratio* and assignment outcomes.

The results of data analysis of two years of student data provides a strong case for adherence to TDD producing positive outcomes. In addition to immediate advantages of improved correctness and coverage, students who adhere to TDD on programming assignments benefit from practicing a technique they may need outside of educational context. However, the observations of sequences of submissions for assignments also reinforced concerns about some student behavior.

The observations of test-late strategies correspond with the findings of the attitudinal study that students are generally reluctant to adopt a test-first strategy. While it may be concerning to educators that students avoid the test-first principle of TDD, identifying it as an obstacle to adoption gives us some direction on how to improve overall adherence to TDD. Results from the survey demonstrated a strong relationship between student's attitudes about a behavior, and their frequency of following that behavior. Accordingly, it is paramount that we address the attitude toward test-first development in order for students to follow such behavior.

In order to enforce test-first behavior, one may suggest that we require an intermediate submission of tests only before students start on their solution code. However, this strategy has several flaws. Foremost, it should be emphasized that TDD advocates an incremental approach to development: units are tested and implemented before progressing to another unit. Providing *all* tests before doing *any* solution implementation would be a large-scale strategy that does not comply with TDD. Likewise, requiring tests to be submitted independently of solution code has serious drawbacks.

First, tests should be reliant on the solution, so by itself, a test suite cannot be meaningfully evaluated. Even if an assignment strictly requires conformity to a specific design and student tests are paired with a reference solution (presumably written by the instructor) for evaluation, it would not be hard for students to subvert the system by developing both their tests and solutions but only submitting the tests. Therefore, requiring tests to be submitted before solutions neither guarantees test-first behavior nor complies with TDD.

Instead, to persuade more positive attitudes toward test-first approach to development will likely require incentives to foster a "test-first mentality" as Spacco and Pugh suggested [20]. There are different possible approaches to encourage student behavior with incentives. Reinforcement of test-first mentality in classroom activities may help. Instructional technology offers other avenues for persuading behavior.

Since Web-CAT provides insight into students' behaviors as they are working on programming assignments, it also has the potential opportunity to recognize and reinforce good behaviors, while discouraging poor behaviors. Accordingly, I have adapted Web-CAT's mechanism that provides feedback to students on their submissions.

Upon submission, I have programmed Web-CAT to consider multiple measurements of behaviors that indicate both progress on their work and adherence to TDD. By leveraging the automated observations of behaviors, Web-CAT may now provide adaptive feedback purposely to encourage TDD adherence. The adaptive feedback can also strategically leverage the hints Web-CAT usually offers as incentives to follow TDD. I am currently running a study to investigate student interaction with the adaptive feedback and the consequences on their behaviors and outcomes.

This is a unique approach in teaching TDD. In addition to the potential for influencing student behavior, leveraging instructional technology to automatically detect behavior during software development has broader implications. My research uses an automated grading system to assess methods in ways previously unexplored. It introduces the potential for evaluating software development methods taught in computer science with greater discernment than offered through single-deliverable assignments or examinations. This approach need not be restricted to TDD, but may be applied to studying a variety of software development techniques.

My research depicts unique approaches to two challenges in evaluating development processes: understanding the influence of attitude toward adopting new methods, and automatically assessing behaviors as they pertain to the methods. The research already provides meaningful insight into the relationship between affect and behavior in software development. Moreover, I have identified averse attitudes toward test-first approach as the primary impediment to students adopting TDD. In addition to recognizing hurdles in teaching TDD, my studies also provide empirical evidence reinforcing benefits of TDD. Future studies will continue to contribute to novel approaches in TDD education as well as to broadly assessing software development methods in the computer science curriculum.

5. REFERENCES

[1] ABET (2012). "Criteria for Accrediting Computing Programs, 2012-2013." Retrieved April, from http://www.abet.org/computing-criteria-2012-2013/.

[2] Barriocanal, E.G., Urban, M.S, et al., "An experience in integrating automated unit testing practices in an introductory programming course," SIGCSE Bull., vol. 34., no. 4., pp. 125-128, 2002.

[3] Beck, K "Embracing change with extreme programming." Computer, vol. 32, no. 10, pp. 70-77, 1999.

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

[5] Buffardi, K and Edwards, S.H. “Exploring Influences on Student Adherence to Test-Driven Development,” ITiCSE 2012 *(in press)*

[6] Canfora, G., Cimitile, A. et al., "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, pp. 364-371, 2006.

[7] Desai, C., Janzen, D.S., and Clements, J. "Implications of integrating test-driven development into CS1/CS2 curricula," SIGCSE Bull., vol. 41., issue 1, pp. 148-152, 2009.

[8] Desai, C., Janzen, D.S., and Savage, K. "A survey of evidence for test-driven development in academia," SIGCSE Bull., vol. 40, issue 2, pp. 97-101, 2008.

[9] Driscoll, R. (2007). "Westside Test Anxiety Scale Validation." Education Resources Information Center: 6.

[10] Edwards, S.H. “Improving student performance by evaluating how well students test their own programs.” *J. Educational Resources in Computing*, 3(3): 1-24, September 2003.

[11] Edwards, S.H. "Using software testing to move students from trial-and-error to reflection-in-action," SIGCSE Bull., vol. 36, issue 1, pp. 26-30, 2004.

[12] Edwards, S.H., J. Snyder, et al. “Comparing effective and ineffective behaviors of student programmers.” Proceedings of the fifth international workshop on Computing education research workshop. Berkely, CA, USA, ACM: 3-14, 2009.

[13] Fraser, S., Astels, D. et al., "Discipline and practices of TDD: (test driven development)," Companion of the 18th annual ACM SIGPLAN conference, ACM, Anaheim, California, pp. 268-270, 2003.

[14] Janzen, D.S. and Saiedian, H. "A leveled examination of test-driven development acceptance," Proceedings of the 29th international conference on Software Engineering, IEEE Computer Society, pp. 719-722, 2007.

[15] JUnit.org Resources for Test Driven Development. Web page last accessed Sept. 23, 2011: <http://www.junit.org/>

[16] Lappalainen, V. , Itkonen, J. ,et al., "ComTest: a tool to impart TDD and unit testing to introductory level programming," Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, ACM, Bilkent, Ankara, Turkey, pp. 63-67, 2010.

[17] Melnik, G. and Maurer, F. "A cross-program investigation of students' perceptions of agile methods," Proceedings of the 27th international conference on Software engineering, ACM, St. Louis, Missouri, pp. 481-488, 2005.

[18] Rodebaugh, T. L., C. M. Woods, et al. (2004). "More Information From Fewer Questions: The Factor Structure and Item Properties of the Original and Brief Fear of Negative Evaluation Scale." Psychological Assessment 16(2): 169-181.

[19] Spacco, J., Hovemeyer, D. et al., "Experiences with marmoset: designing and using an advanced submission and testing system for programming courses," SIGCSE Bull., vol. 38, issue 3, pp. 13-17, 2006.

[20] Spacco, J. and Pugh, W. "Helping students appreciate test-driven development (TDD)," Companion to the 21st ACM SIGPLAN symposium, ACM, Portland, Oregon, pp. 907-913, 2006.

[21] Web-CAT. Web page last accessed Sept. 23, 2011: <http://web-cat.cs.vt.edu>

[22] Yerkes, R.M. and J.D. Dodson. “The relation of strength of stimulus to rapidity of habit-formation.” *Journal of Comparative Neurology and Psychology* 18(5): 459-482, 1908.