Effective and Ineffective Software Testing Behaviors by Novice Programmers

Kevin Buffardi
Virginia Tech
2202 Kraft Drive
Blacksburg, VA 24060
+1 540-231-5723
kbuffardi@vt.edu

Stephen H. Edwards
Virginia Tech
2202 Kraft Drive
Blacksburg, VA 24060
+1 540-231-5723
edwards@cs.vt.edu

ABSTRACT

This data-driven paper quantitatively evaluates software testing behaviors that students exhibited in introductory computer science courses. The evaluation includes data collected over five years (10 semesters) from 49,980 programming assignment submissions by 883 different students. To examine the effectiveness of software testing behaviors, we investigate the quality of their testing at different stages of their development. We partition testing behaviors into four groups according to when in their development they first achieve substantial (at least 85%) test coverage.

The study reveals significant results regarding effective and ineffective testing behaviors. A within-subjects comparison finds that higher coverage in early development is associated with higher quality code and with completing work earlier. Post-hoc analysis also suggests that the relationship between early testing and positive outcomes is independent of time management and effects of individuals' abilities. However, roughly 76% of students exhibit different testing behaviors on different assignments, demonstrating an opportunity to foster better, more consistent testing habits among computer science students.

Categories and Subject Descriptors

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

Keywords

Software testing, software development, test-driven development.

1. INTRODUCTION

Software testing is a common technique for systematically verifying adherence to program specifications. Accordingly, many universities include software testing in the computer science curriculum. With the particular popularity of test-driven development (TDD), there are several benefits to imparting students with effective testing habits during software development.

First, since TDD emphasizes incremental development of tests and code side by side in a "test a little, code a little" pattern [5], it encourages programmers to test small units of code before completing an entire solution. Consequently, students may demonstrate reflection-in-action [12] instead of less effective trial-and-error strategies. Reflection engages higher-order thinking skills valuable to developing problem solving skills [2].

Second, TDD's incremental approach should build confidence in the code while improving maintainability [4]. Studies have shown improved quality of tests and code when following TDD in both academia [8][10][20] and industry [6][9]. Accordingly, students who follow TDD for programming assignments should produce better programs and tests.

Third, TDD is used in industry [14], especially with the popularity of Agile Development and eXtreme Programming [4]. In accordance with accreditation guidelines [1], computing programs should prepare students with relevant professional skills and techniques. By practicing TDD in school, students gain valuable experience with a technique used in industry. While some universities are integrating TDD into their computer science curricula, there are unique challenges to teaching it.

Primarily, reports indicate that novice programmers in particular are reticent to adopt TDD [3][7][16]. Several independent accounts have indicated a special need to motivate programmers to adhere to the test-first technique [7][16][17]. In addition, traditional evaluations of school programming projects only account for a single deliverable. Unfortunately, this product-oriented evaluation lacks any information about the *process* of software development. Without insight into students' testing behaviors throughout development, we cannot conclude whether they adhere to TDD methods.

This paper describes a study conducted on a 5-year (10-semester) dataset involving introductory programming assignments completed by 883 unique students. After being taught TDD, students were required to write software tests as part of each solution, and an automated grading system was used to collect their work. Students were allowed to make multiple submissions to receive feedback and refine their work as they developed their solution. As a result, the dataset includes a total of 49,980 separate attempts at the programming assignments given to the students in the study, representing a series of snapshots of the work-in-progress of each student. By examining relationships between when students add software tests to their projects and how thoroughly they test their own code, this study shows that there is a positive relationship between early testing and more positive student outcomes on programming assignments, including better scores and reduced likelihood of turning in work late.

2. PREVIOUS WORK

There are several tools and studies investigating students' programming development habits. Some tools extend the integrated development environments (IDE) that students install on their own computers to log their development activities. Systems using this approach include ClockIt [13] and Retina [19]. Jadud identified common syntactic errors and motivations for compiling within the BlueJ IDE[15].

Meanwhile, using ClockIt, Fenwick [13] revealed students' struggles to manage their time wisely and the negative impact that procrastinating programming assignments had on their performance. Murphy's analysis also found that students routinely underestimate how much time programming assignments demand [19]. Likewise, managing time is a clear challenge in students' development. Consequently, it is also important to investigate whether students appropriately allocate time for software testing.

Unlike IDE-integrated tools, both Marmoset [21] and Web-CAT [11] are online, automated grading systems. Using these systems, students may submit their work at their own convenience and receive feedback from an automated evaluation of their code. These automated grading tools evaluate students' submissions using static analysis tools, software reference tests for verifying the solution's correctness, and dynamic analysis of the students' own software tests.

After receiving feedback, students may repeatedly revise and resubmit their code until they are satisfied with their work. By making multiple submissions per assignment, evaluation analytics provide snapshots of each student's development activities. Consequently, a series of snapshots show the *process* of development—exposing students' behaviors.

3. METHOD

3.1 Data Collection

In introductory computer science courses at our university, students submit their programming assignments to Web-CAT. After submitting, students receive prompt feedback, including scores for the correctness of their code and their test coverage. The percentage of instructor-provided reference tests that pass when run against a student's code determines the correctness score. Meanwhile, test coverage is calculated based on how much of the student's code—a composite of methods, statements, and branches—is executed by their tests.

After receiving feedback from Web-CAT, students may revise their code and resubmit as many times as they like without penalty. Their assignment grade is determined by their final submission. The score calculation includes: correctness, test coverage, and style. Instructors and/or assistants often assign a portion of each program grade by hand by evaluating documentation and design; however, we exclude this aspect of assignment scores in our analysis to avoid issues with subjectivity and consistency between human graders. In addition, assignments are sometimes submitted after the deadline, which may involve a score penalty depending on a course's policies regarding late work. However, late submission penalties are disregarded in our analysis of student program scores to avoid misrepresenting the actual quality of the solution and software tests produced.

To perform a comprehensive analysis of student testing behavior, we used a five-year dataset of programming assignment submissions from our CS1 course: Introduction to Object-oriented Development I. The dataset includes each full-length semester from Spring 2004 to Fall 2008. We excluded data from summer semesters because summer schedules are abbreviated and consequently the courses differ considerably in pace and structure.

The ten-semester data set includes 49,980 submissions for 3,715 scored assignments, with an average of over 12 submissions by each individual student completing each separate assignment. The data reflects the work of 883 unique students. While several assignments were used for more than one semester, both the assignment instructions and number of assignments varied between semesters.

To observe students' behaviors during development, we analyzed each of their submissions for each assignment. In addition to quality of code—as measured by correctness and test coverage—each submission also included data on the time of submission, amount of solution code, and amount of test code. In particular, we concentrated on the following metrics:

- NCLOC: Non-comment lines of code, separated into lines that are part of the student's solution and lines that are part of the student's software tests.
- Time Remaining: The amount of time between when a submission was made and the assignment deadline. Negative values represent submissions made after the deadline.
- Time Elapsed: The amount of time between the students' first submission for that assignment and the current submission in question.
- Relative Worktime: The amount of time elapsed, expressed as a percentage of the total duration over all of the student's submissions for an assignment. Zero- and one-values represent the first and final submissions by that individual, respectively. This metric disregards the relationship between submission time and the assignment deadline. Instead, it represents the progression of time within the workflow of development.

Since assignments vary in scale and complexity between courses and semesters, the amount of code varies greatly between assignments; the final submission NCLOC mean is 587.6 and standard deviation is 729.6. In addition, this study concentrates on *testing* behaviors so we are more concerned with the amount of test code compared to the size of the solution at any given time.

Accordingly, we concentrate on the test NCLOC *relative* to the amount of solution NCLOC. Low relative NCLOC (nearing zero) indicates a lack of test code relative to the amount of solution code that has been written. Meanwhile, high relative NCLOC indicates that the student has developed comparable amounts of both test and solution code. In some cases, students produce more test code than solution code and consequently have a relative NCLOC value greater than one.

By capturing multiple submissions per assignment, snapshots of students' development provide insight into their behaviors over time. We analyzed student submissions over time and identified significant milestones in their development processes. The **first submission** within an assignment offers a preliminary glimpse of a student's code. While students often write a considerable amount of code before their first submission, it is the earliest available snapshot of their work and comparisons to later submissions indicate changes and behaviors that occur as the student continues to refine his or her work.

Since students were taught test-driven development (TDD), which encourages early and incremental testing, we identified when in their development they first achieved substantial test coverage (at least 85% of their code at that moment is covered) on each assignment. In this paper, we refer to this submission as the **test threshold milestone**. When students follow TDD strictly and write quality tests, their test threshold milestone should take place very early in development. However, postponing testing pushes the test threshold milestone to later submissions, perhaps even to a student's final submission.

Students sometimes continue to submit their work after they have achieved their highest correctness score. This occurs because their score reflects more than just correctness, and other aspects of their solution (or testing) may still need improvement. We refer to the first time they achieve their highest correctness as the **maximum score milestone**. Some submissions following this milestone may represent attempts (but failures) to raise their correctness score. However, some students continue to submit after achieving perfect correctness scores, usually in an attempt to improve another aspect of their grade beyond correctness.

Finally, a student's **final submission** marks the milestone of completing the assignment. This milestone is particularly valuable in observing students' qualitative and quantitative outcomes—final correctness and coverage in particular. Additionally, by comparing an earlier submission to the final milestone, we can observe students' progress relative to their completed work.

3.2 Grouping Data

To extract effective and ineffective behaviors, we must first identify assignments with positive and negative outcomes. First, we investigate the final correctness score of each assignment to distinguish good- and poor-quality solutions. Our courses use a 10-point graduated letter-grade scale, where A/B/C/D/F grades are designated by 90/80/70/60% and below, respectively. We identified A/B correctness scores (80% and above) as good outcomes and C/D/F (below 80%) as poorer outcomes.

Often, researchers compare behaviors of higher-performing students with those of lower-performing students. However, it would be hasty to make conclusions about the effects of behaviors based *only* on correlations with outcomes. Outcomes may reflect differences in undetected behaviors or other external factors. For example, students who score well on assignments may devote more time to their projects or demonstrate other "good work habits." Meanwhile, students with worse scores may exhibit "bad work habits" such as procrastinating or neglecting the amount of time a project requires. However, an entirely different type of behavior (such as the adherence to effective software engineering strategies) may result in the differences in their outcomes as well.

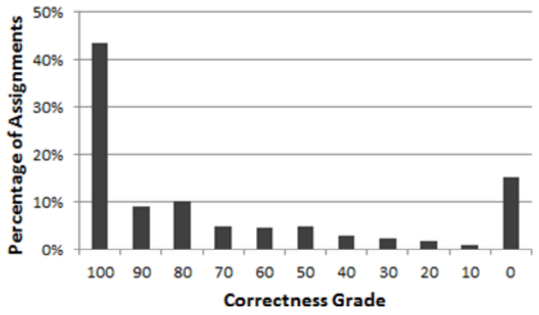

Figure 1. Distribution of final correctness scores for all assignments

To address this concern, we grouped students by how they performed on all of their assignments. The **A/B group** includes students who earned correctness scores of 80% or better on each of their assignments. Likewise, the **C/D/F group** consists of students who never scored 80% or better on any of their assignments. Lastly, the **varied-performance group** submitted at least one assignment earning 80% or better and at least one assignment below 80%. **Figure 1** shows the distribution of assignment correctness scores.

Of the 883 unique students, 307 (35%) belong to A/B group, 170 (19%) to C/D/F group, and 406 (46%) belong to varied-performance group. **Table 1** summarizes mean outcomes (correctness, coverage, relative NCLOC, time remaining, and time elapsed) of each of these groups. By concentrating on the **varied-performance group**, we can perform within-subject comparisons to identify behavioral differences between when a student scored well and when he/she performed poorly.

We compared behaviors of assignments earning A/B and C/D/F correctness outcomes using an unbalanced, partial-factorial design. Assignment correctness group (A/B vs. C/D/F), semester, and subject (individual student) were the factors in a 2x10x406 experimental model. Analysis of variance was performed between groups, but within-subjects. Since each semester used a unique set of assignments, students only enrolled in one semester, and students had varying numbers of A/B or C/D/F scores, subjects could not appear in each combination of factors. Although the

Table 1. Summarized final outcomes, with students grouped by their performance on all their assignments

	Students	Completed Programs	Correctness (M, sd)	Coverage (M, sd)	Test:Solution Relative NCLOC (M,sd)	Time Remaining (Hrs) (M, sd)	Time Elapsed (Hrs) (M, sd)
A/B only	307	1324	0.97, 0.06	0.95, 0.06	0.92, 0.45	30.72, 64.15	37.82, 48.71
Varied	406	1891	0.76, 0.33	0.87, 0.20	0.77, 0.40	18.97, 61.28	28.56, 42.65
C/D/F only	170	500	0.27, 0.29	0.64, 0.31	0.53, 0.35	6.74, 46.48	21.92, 34.94
Total	883	3715	0.76, 0.34	0.87, 0.21	0.79, 0.43	21.51, 61.07	30.97, 44.33

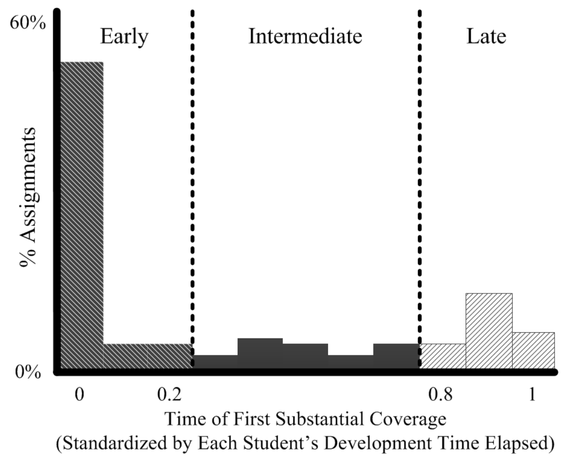

Figure 2. Distribution and grouping of first moment reaching 85% coverage by standardized time elapsed within assignment submissions

unbalanced, partial-factorial design weakens the power of the statistical results, the statistical tests also benefit from the dataset's large size.

After investigating varied-performance, within-subject differences, we also compared trends of the consistently scoring A/B and C/D/F groups. By comparing when these groups start and finish their development and how their testing changes over that time, we can isolate behaviors specific to well- and poor-performing students.

Furthermore, we examined the testing metrics at the development milestones of the varied-performance group. There is a clear bimodal distribution of the *relative worktime* of the *testing threshold* milestone. In other words, there are distinct periods within an assignment's development when students achieve substantial coverage. We partitioned test development time into four groups: early, intermediate, late, and neglectful. The "early testers" reached the *testing threshold* within the first 20% of their *relative worktime*. "Late testers" reached the same milestone within the last 20% of their relative worktime. "Intermediate testers" reached the milestone between those groups, while "neglectful testers" never achieved 85% coverage. **Figure 2** shows the distribution and partitioning of the *testing threshold relative worktime*.

4. RESULTS

The Friedman's test for investigating within-subjects differences of the **varied-performance group** compared timing of development milestones (first submission, testing threshold, maximum scoring, and final submission), quality of code (correctness and coverage), as well as quantity of test code (relative NCLOC). Likewise, we compared the same metrics *between* exclusively A/B and C/D/F groups using the Wilcoxon signed-rank test. The following subsections describe the findings for each area of analysis, before detailing the results of a post-hoc analysis of testing strategy.

4.1 Time of Development Milestones

A within-subjects comparison reveals that the first submission time for A/B assignments (M=62.26, sd=7.29) is earlier (F(1,1)=36.96, p<0.0001) before the deadline than for the same students' C/D/F assignments (M=32.11, sd=58.52) from the varied-performance group. Likewise, the first submissions of consistent A/B students (M=68.54, sd=75.08) takes place earlier (p<0.001) than that of consistent C/D/F students (M=28.66, sd=51.11) according to the Wilcoxon test.

We compared the time remaining for the final submissions on well- and poor-performing assignments. In the within-subjects comparison, well-performing assignments were finished (M=28.02 sd=63.73) significantly earlier (F(1,1)=32.56, p<0.0001) than poor-performing assignments (M=3.88, sd=52.68). Consistent A/B students submitted their final submissions (M=30.72, sd=64.15) with significantly more time remaining (p<0.0001) than C/D/F students (M=6.74, sd=46.49).

To compare the timing of the *testing threshold* and *maximum scoring* milestones, we examined their *relative worktime*, or the percentage of the time elapsed between first and last submissions. Varied-performing students reached the testing threshold earlier in development (F(1,1)=1.26, p<0.01) on poor-performing assignments (M=0.55, sd=0.45) than on well-performing assignments (M=0.65, sd=0.43). The C/D/F students who attained the testing threshold (M=0.40, sd=0.39) did so earlier in their development process (p<0.0001) than A/B students (M=0.53, sd=0.37). However, interpretations of testing threshold relative worktime comparisons should use particular caution because these tests exclude all assignments that failed to achieve 85% coverage.

Varied-performing students reached their maximum scoring milestone earlier (F(1,1)=55.78, p<0.0001) within their development on poor-performing assignments (M=0.55, sd= 0.45) than on well-performing assignments (M=0.65, sd=0.43). Correspondingly, A/B students reached their maximum scoring milestone (M=0.70, sd=0.40) later (p<0.0001) than C/D/F students (M=0.41, sd=0.45).

Both of these last findings may contradict expectations: poorer assignments reached the testing threshold milestone earlier in development than better assignments, and poorer assignments reached the maximum scoring milestone earlier in development as well. Since testing early is supposed to benefit students, one would hypothesize that better scoring assignments would be associated with demonstrating substantial correctness and test coverage earlier during development compared to weaker assignments. Consequently, we investigated this issue further in *Section 4.4: Testing Strategies*.

Lastly, with concern to time spent in development, we examined the overall time elapsed from first to last submission for each assignment. The within-subjects comparison of varied-performing students shows that the amount of time a student spent (in hours) on his or her well-performing assignments (M=27.31, sd=43.22) was slightly less than (F(1,1)=7.51, p<0.01) the time spent by the same individual on poor-performing assignments (M=30.66, sd=41.61). However, students who consistently achieved A/B scores (M=37.82, sd=48.71) spent nearly twice as much time (p<0.001) as students who consistently achieved C/D/F scores (M=21.92, sd=34.94).

4.2 Code Quality and Test Quality

We measured quality of solution code by tracking correctness during development. Meanwhile, we evaluated quality of testing with testing coverage. Final correctness is used to identify assignment groups in the within-subject Friedman's test and to identify student groups for the Wilcoxon between-groups comparison. Therefore, for the final submission, we only compared test quality within-subjects and between groups.

The within-subject comparison of the varied-performance group, demonstrated significantly higher coverage (F(1,1)=684.44, p<0.0001) for assignments with high correctness (M=0.94, sd=0.07) than those with low correctness (M=0.75, sd=0.28). In other words, when students in the varied group achieved higher test coverage scores, they also produced solutions with fewer bugs (Spearman's ρ=0.67, p<0.0001). As expected, the A/B students' final submissions also had significantly (p<0.0001) higher coverage (M=0.94, sd=0.06) than those of C/D/F students (M=0.64, sd=0.31). Over the entire dataset, there was a strong positive correlation between higher test coverage and higher correctness scores (Spearman's ρ=0.71, p<0.0001).

For the first submission, varied-performance students had significantly higher correctness (F(1,1)=46.45, p<0.0001) and coverage (F(1,1)=93.88, p<0.0001) on their well-performing assignments (M=0.45, sd=0.42; M=0.71, sd=0.33) than on poor-performing assignments (M=0.14, sd=0.24; M=0.46, sd=0.24, respectively). The between group analysis also found that the first submissions for A/B students had higher (p<0.0001) correctness (M=0.40, sd=0.34) than on C/D/F student assignments (M=0.08, sd=0.18). Furthermore, the A/B's also had higher (p<0.0001) initial coverage (M=0.68, sd=0.35) than the C/D/F group (M=0.49, sd=0.34).

We also investigated the quality of code at the point of reaching the testing threshold milestone. Satisfying our definition for the testing threshold milestone already requires that coverage is at least 85%. However, not all assignments achieve this milestone. Varied-performance students reach the testing threshold on 63% of their well-performing assignments but on only 43% of their poor-performing assignments.

When considering only those assignments that reach the milestone, a within-subject comparison reveals that the solution correctness at the time of the testing threshold milestone is significantly higher (F(1,1)=36.51, p<0.0001) on well-performing assignments (M=0.74, sd=0.32) than on poor-performing assignments (M=0.42, sd=0.26). Similarly, assignments from A/B students achieve the testing threshold 65% of the time, while C/D/F students only achieve it 32% of the time. Moreover, A/B assignments (M=0.76, sd=0.39) also have higher (p<0.0001) correctness at the testing threshold milestone than do C/D/F assignments (M=0.39, sd=0.24).

Finally, we compared code quality at the time of achieving the highest correctness within each assignment. Since we already know by definition that well-performing assignments have higher correctness than poor-performing assignments, we only compare test coverage at the maximum score milestone. At the time of the maximum score milestone, varied-performance students have higher (F(1,1)=542.15, p<0.0001) coverage on well-performing assignments (M=0.94, sd=0.07) than on poor-performing assignments (M=0.71, sd=0.31). Accordingly, assignments from consistent A/B students (M=0.95, sd=0.06) have higher (p<0.0001) coverage than those from consistent C/D/F students (M=0.61, sd=0.34). It should not be surprising that throughout development, consistent A/B students regularly demonstrate better solutions and testing than consistent C/D/F students do. However, it is more telling that varied-performance students also demonstrate a positive relationship between maintaining good coverage throughout development and producing higher-quality results. This within-subject comparison rebukes the hypothesis that the relationship between high-quality testing and high-quality solutions only reflects inherent study/work habits of "good" and "bad" students.

4.3 Quantity of Code

In addition to code quality, we are concerned with how much test code students write at different stages of development. By comparing the amount of test NCLOC to solution NCLOC, we can observe—at any given moment in development—the amount of work on testing relative to that of the solution. To inspect this amount of testing throughout submissions to Web-CAT, we compared *relative NCLOC* at the times of: first submission, testing threshold milestone (when available), maximum scoring milestone, and final submission.

In within-subject comparisons, the fit for each threshold moment was calculated independently since not all assignments reach the testing threshold milestone. On the first submission, varied-performance students had significantly higher (F(1,1)=35.25, p<0.0001) *relative NCLOC* (more test code per line of solution code) on well-performing assignments (M=0.61, sd=0.41) than on poor-performing assignments (M=0.47, sd=0.38).

Well-performing assignments (M=0.81, sd=0.39) also had significantly higher relative NCLOC (F(1,1)=35.81, p<0.0001) at the testing threshold milestone than poor-performing assignments (M=0.76, sd=0.32). Likewise, well-performing assignments (M=0.80, sd=0.38) also had significantly higher relative NCLOC (F(1,1)=109.73, p<0.0001) at the maximum scoring milestone than poor-performing assignments (M=0.64, sd=0.42).

However, there is no significant difference (F(1,1)=1.37, p=0.24) in relative NCLOC between consistently well-performing assignments (M=0.81, sd=0.38) and consistently poor-performing assignments (M=0.76, sd=0.33) at the time of testing threshold milestone. Since the milestone requires substantial testing, poor assignments with less test code (and that never reach the milestone at all) are excluded, consequently artificially inflating the mean. To further support this explanation, within varied-performing students, fewer poor-performing assignments (35%) achieved testing threshold than well-performing assignments (57%).

Figure 3 illustrates the change in relative NCLOC during development for well-performing and poor-performing assignments of varied-performing students. Both groups show a

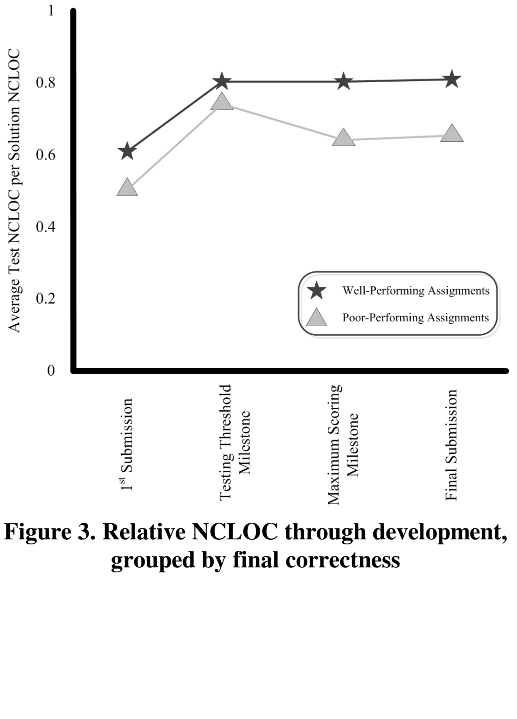

Figure 3. Relative NCLOC through development, grouped by final correctness

relative increase in testing between first submission and the submission where the assignment first achieves 85% coverage. However, the well-performing assignments tend to maintain (or slightly increase) amount of test code through the rest of development while poor-performing assignments do not.

4.4 Testing Strategies

While the majority of analyses in this study support the argument that early, incremental testing scaffolds higher-quality tests and solutions, aberrations emerge when inspecting relative worktime (between first and last submissions) of the testing threshold milestone. These peculiarities are highlighted by the results of a post-hoc Wilcoxon each pairs test comparing quantity and quality of code between early, intermediate, late, and neglectful testing groups (see: **Figure 2**). To account for chances of type I error with multiple comparisons, we used the Bonferroni correction to set a stricter (α=0.008) confidence interval for post-hoc tests.

When comparing the final correctness between each group, early testing (M=0.80, sd=0.34) and intermediate testing groups (M=0.87, sd=0.20) performed comparably well (p=0.55), but both worse (p<0.001 and p<0.003) than the late testing group (M=0.88, sd=0.21). All three of these groups performed better (each p<0.0001) than the neglectful group (M=0.65, sd=0.39). For final coverage, the late (M=0.95, sd=0.06) and intermediate (M=0.95, sd=0.05) groups scored comparably (p=0.11). While the late group's mean coverage is greater than the early group's (M=0.88, sd=0.24), this difference is not significant (p=0.018). Likewise, there is no significant difference (p=0.44) between the intermediate and early groups. All three groups performed better (p<0.0001) than the neglectful group (M=0.78, sd=0.25) in terms of final test coverage.

Similarly, late (M=0.88, sd=0.40) and intermediate (M=0.87, sd=0.42) groups had higher (but not significant) final relative NCLOC (p=0.02 and p=0.21) than the early testing group (M=0.83, sd=0.44). There was no significant difference between the intermediate and late groups (p=0.23). All three groups had higher final relative NCLOC (p<0.0001) than the neglectful group (M=0.70, sd=0.42). From these comparisons alone, one might conclude that either early testing really is no better than later testing strategies, or that there is a previously undetected phenomenon affecting the early testing group scores.

To investigate the relationship between testing threshold relative worktime—which the testing groups are based on—with other timing factors, we performed a Spearman's correlation with: time remaining at the first and last submissions, final time elapsed, and maximum scoring relative worktime. **Table 2** summarizes the correlations with the testing threshold relative worktime. Given the moderately positive correlation between the relative worktime of the test threshold and maximum score threshold, it is not surprising to find that the maximum score milestone also has a bimodal distribution, as demonstrated in **Figure 4**.

Table 2. Spearman Rho correlation with testing threshold relative worktime

			Testing Threshold Relative Worktime (M=0.53, sd=0.39)	
	M	sd	Spearman Correlation (ρ)	p
Time Remaining (1st Submission)	52.48	70.01	-0.01	0.52
Relative Worktime (Maximum Scoring Milestone)	*0.62*	*0.44*	*0.34*	*<0.0001*
Time Remaining (Last Submission)	*21.51*	*61.07*	-0.04	0.09
Time Elapsed (Last Submission)	30.97	44.33	*0.07*	*<0.01*

A quarter of relative worktimes for maximum score threshold are 0.0, or the very first submissions to Web-CAT. These assignments reflect instances where students never improve upon their correctness after their first submission. We will refer to these assignments as the *complacent* group. As one might expect from assignments without correctness improvement, these assignments averaged very few submissions (M=4.69, sd=6.31), and little time elapsed between first and last submissions (M=11.85, sd=30.83). The correctness (M=0.52, sd=0.47) and coverage (M=0.69, sd=0.37) on average are also quite poor. However, the wide standard deviations reflect divergent trends within this segment.

Half of the complacent assignments stagnate with either 100% or 0% correctness. The 25% who began with 0% correctness and never improved reflect students who either neglected the effort necessary for the assignment or seem to have given up on it. Meanwhile, the 25% of the complacent group who achieved 100% correctness on their first submission clearly demonstrated substantial work before submitting to Web-CAT and were nearly—if not totally—finished. Unfortunately, the data reflecting these assignments do not expose much of their development process.

Accordingly, with poor granularity of the complacent group's development process, it would not be appropriate to categorize their testing strategies with those assignments with finer granularity. For instance, if a student only submits once but achieved at least 85% coverage, does that indicate early testing? Likewise, imagine a student facing an impending deadline and submitting four times over the span of an hour. Is there a discernible difference in testing strategies between taking 10 and 60 minutes to achieve 85% coverage? Even in non-complacent assignments with few submissions or short duration, we can compare development patterns through differences in changes to their work. However, complacent assignments suffer from a lack of both granularity and change.

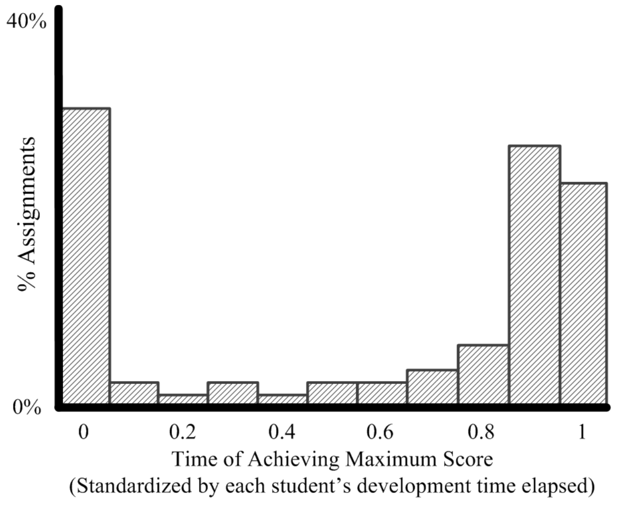

Figure 4. Distribution of time of maximum scoring milestone, by relative worktime

These complacent assignments appear to reflect at least two distinct habits. In some situations, students complete the vast majority of work on well-performing assignments before submitting. In other situations, students seem to give up shortly after their first submission. It is unlikely that testing strategies influenced the outcomes of those complacent with poor correctness. Instead, their outcomes likely reflect personal characteristics or study habits such as lack of motivation, poor time management, or inadequate preparation. Meanwhile, it would be intriguing to investigate the development habits of those in the complacent group who started with perfect correctness. However, since our data are limited to their submissions to Web-CAT that began at the tail end of the development process, it would not be fair to categorize their testing habits as either early or late with so little information.

If we prune complacent assignments from the dataset and repeat the Wilcoxon test, we observe a notable change in results. Now, when comparing the final correctness, the early group (M=0.90, sd=0.18) performs comparably (p=0.48) to the late (M=0.89, sd=0.19) group, better (p<0.001) than the intermediate group (M=0.87, sd=0.19), and still considerably better (p<0.0001) than the neglectful group (M=0.79, sd=0.26). However, testing quality effects are even more salient.

When comparing the final coverage of the pruned dataset, the early group (M=0.96, sd=0.08) performs significantly better than the intermediate (M=0.95, sd=0.05, p<0.0001) and late (M=0.95, sd=0.06, p<0.001) groups. There is no significant difference (p=0.43) between coverage of intermediate and late groups. All three groups produced significantly higher coverage (p<0.0001) than the neglectful group (M=0.84, sd=0.16).

The final relative NCLOC for early (M=0.92, sd=0.42) is insignificantly higher than intermediate (M=0.86, sd=0.42, p=0.02), and late (M=0.87, sd=0.42, p=0.17) groups. All three groups produce higher (p<0.0001) relative NCLOC than the neglectful group (M=0.75, sd=0.42).

With these improved groupings, **Figure 5** illustrates the timing and coverage at each milestone for each group (excluding pruned submissions). The milestones follow, in order: first submission, testing threshold milestone, maximum scoring milestone, and final submission. Note that the neglectful group, by definition, does not have a second (testing threshold) milestone.

Finally, to investigate the relationship between testing strategy and outcome, independent of time management behaviors, we performed post-hoc analysis excluding submissions with poor habits. Specifically, we included only assignments with the following credentials: one (or more) submissions preceding the deadline by at least 48 hours; multiple submissions spanning development over 24 hours; and final submission before the deadline. This pruned dataset included 2,727 assignments from 790 unique students.

Using the Wilcoxon test on each pair of groups (early, intermediate, late, neglectful), we found early-tested assignments (M=0.92, sd=0.17) had higher correctness (p<0.01) than intermediate-tested assignments (M=0.90, sd=0.15) approaching significance (α=0.008). The intermediate group also approached higher (p=0.09) correctness than the late group (M=0.89, sd=0.15). Accordingly, the aforementioned groups also had higher correctness (p<0.0001) than the neglectful group (M=0.76, sd=0.32).

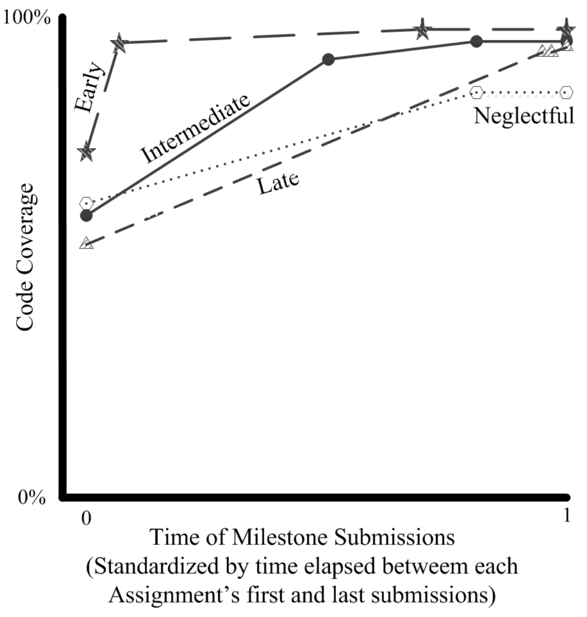

Figure 5. Distribution of time of maximum scoring milestone, by relative worktime

The final test coverage comparison between groups revealed similar results. The early group (M=0.96, sd=0.08) had significantly higher coverage (p<0.0001) than both the intermediate (M=0.94, sd=0.05) and late (M=0.94, sd=0.05) groups. The intermediate and late groups had no significant difference in coverage (p=0.64). Finally, the intermediate and late groups also had significantly higher coverage (p<0.0001) than the neglectful group (M=0.84, sd=0.15).

Finally, we reviewed students' testing behaviors across assignments. Approximately 18% of students demonstrated neglectful testing practices on all of their assignments. Meanwhile, 2% consistently tested late, 3% consistently tested early, and 1% consistently tested in-between ("intermediate"). The remaining 76% of students followed different testing behaviors on different projects.

5. CONCLUSIONS

Results showing strong positive correlations between early test quality and quantity and the consequential assignment outcomes corroborate with previous studies suggesting benefits of test-first and test-early strategies [3][8][10]. However, we focused on within-subject comparisons of students who performed well on some assignments and poorly on others. By contrasting their testing behaviors on well-performing assignments with those on poorer-performances, we control for students' personal *traits* and isolate the impact of differences of *behaviors*.

We characterized divergent testing strategies based on analysis of how early within an assignment's development a student *first* achieves substantial (>85%) test coverage of their code. By considering timing relative to individuals' development time—opposed to in absolute time before the deadline—we were able to compare testing strategies independently of general time management skills. Furthermore, to investigate the effects of testing independently of time management behaviors, we compared a subset of only assignments that started early, continued to develop for substantial time, and submitted their final work on time.

We found that on assignments where students reached this *testing threshold* milestone *early* in their submissions (<20% within *relative worktime*), they usually produced higher quality code and tests than those who demonstrated later testing strategies. Surprisingly, we also found that assignments achieving the testing threshold *late* (>80% within *relative worktime*) produced higher quality code and tests than those who did so in the middle of their development.

This study benefits from the advantage of observing students' testing behaviors with finer granularity than reviewing a solitary assignment deliverable. However, we also identify small segments of submission behaviors with inadequate detail to infer testing strategies. Approximately one-in-every-eight varied-performance assignments had very few submissions with no measurable progress from their first submission. For the purpose of this study, these complacent assignments lacked enough detail to categorize their testing strategies confidently. Consequently, we excluded these outliers as we found that they were unduly influencing our statistical results.

However, there is potential value in learning more about these assignments with unknown testing and development strategies. Extending the snapshots of students' work into earlier stages of development (i.e. before first submissions) will promote more detail in models of testing behavior. Therefore, we hope this study represents a step in the progression of techniques and tools that enable increasing granularity in observing software development behaviors.

6. REFERENCES

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

[2] Anderson, L. W., D. R. Krathwohl, et al. (2001). A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives, Addison Wesley Longman.

[3] Barriocanal, E. G., M.-Á. S. Urbán, et al. (2002). "An experience in integrating automated unit testing practices in an introductory programming course." SIGCSE Bull. 34(4): 125-128.

[4] Beck, K. (1999). "Embracing change with extreme programming." Computer 32(10): 70-77.

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

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

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

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

[9] Canfora, G., A. Cimitile, et al. (2006). Evaluating advantages of test driven development: a controlled experiment with professionals. Proc. of ECEM 2006, Rio de Janeiro, Brazil, ACM: 364-371.

[10] Desai, C., D. Janzen, et al. (2008). "A survey of evidence for test-driven development in academia." SIGCSE Bull. 40(2): 97-101.

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

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

[13] Fenwick, J. B., Norris, C., Barry, F. E., Rountree, J., Spicer, C. J., and Cheek, S. D. Another look at the behaviors of novice programmers. In Proc. 40th ACM Tech. Symp. Computer Science Education (SIGCSE), ACM, New York, NY, 2009, pp. 296–300.

[14] 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.

[15] Jadud, M. A first look at novice compilation behaviour using BlueJ. Computer Science Education, 15(1):25–40, March 2005.

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

[17] Lappalainen, V., J. Itkonen, et al. (2010). ComTest: a tool to impart TDD and unit testing to introductory level programming. Proc. of ITiCSE 2010. Bilkent, Ankara, Turkey, ACM: 63-67.

[18] Melnik, G. and F. Maurer (2005). A cross-program investigation of students' perceptions of agile methods. ICSE 2005. St. Louis, MO, USA, ACM: 481-488.

[19] Murphy, C., Kaiser, G., Loveland, K., and Hasan, S. (2009). Retina: helping students and instructors based on observed programming activities. Proc. of SIGSCE 2009, ACM, New York, NY, 2009, pp. 178–182.

[20] Spacco, J. and W. Pugh (2006). Helping students appreciate test-driven development (TDD). Comp. to SIGPLAN. Portland, Oregon, USA, ACM: 907-913.

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