Diagnosability, Adequacy & Size: How Test Suites Impact Autograding

Benjamin S. Clegg
University of Sheffield

Phil McMinn
University of Sheffield

Gordon Fraser
University of Passau

Abstract

Automated grading is now prevalent in software engineering courses, typically assessing the correctness of students' programs using automated test suites. However, deficiencies in test suites could result in inconsistent grading. As such, we investigate how different test suites impact grades, and the extent to which their observable properties influence these grades. We build upon existing work, using students' solution programs, and test suites that we constructed using a sampling approach. We find that there is a high variation in grades from different test suites, with a standard deviation of $\sim$*10.1%. We further investigate how several properties of test suites influence these grades, including the number of tests, coverage, ability to detect other faults, and uniqueness. We use our findings to provide tutors with strategies for building test suites that evaluate students' software with consistency. These strategies include constructing test suites with high coverage, writing unique and diverse tests which evaluate solutions' correctness in different ways, and to run the tests against artificial faults to determine their quality.*

1. Introduction

Automated grading is often used in software engineering courses, offering a means to quickly assess a large number of students' programs [1], which is especially important with ever growing student cohorts.[1] A common approach is to use automated test suites to evaluate the correctness of students' code [2]. However, test suites can vary in quality, preventing them from detecting some faults [3]. This produces a source of potential inconsistency, inaccuracy and unfairness in grades generated by these suites.

We illustrate such grading test suite issues with the method definition in Figure 1a, which should return the greatest of two integer parameters, but only ever returns the first. If we consider grades to be calculated as the percentage of tests that pass, the suite in Figure 1b yields a grade of 100%; it only includes one test that never exercises the fault. If we extend this suite to execute more code, increasing *coverage*, it generates a more reasonable grade of 50% (Figure 1c). However, even with full coverage, extreme grades can still be generated; in Figure 1d, both of the tests are very similar, and make assertions that exercise the faulty line, so they both fail, generating a grade of 0%. Instead, we can evaluate a suite's ability to isolate individual potential faults using a diagnosability metric, such as *uniqueness*. Here, the suite in Figure 1d has poor uniqueness, since every line is covered the same way by each test. Comparatively, the suite in Figure 1c achieves the best possible uniqueness and generates the most reasonable grades; the return statements are covered by different tests.

```
int max(int a, int b) {
  if (a > b)
   return a;
- return b; // Correct
+ return a; // Fault
}
```

(a) Example program containing a mistake.

```
@Test void testA() {
  assertEq(3, max(3, 2));
} // Passes
@Test void testB() {
  assertEq(3, max(2, 3));
} // Fails
```

(c) Resulting grade: 50%.

```
@Test void testA() {
  assertEq(3, max(3, 2));
} // Passes
```

(b) Resulting grade: 100%.

```
@Test void testC() {
  assertEq(2, max(2, 1));
  assertEq(2, max(1, 2));
} // Fails (2nd assert)
@Test void testD() {
  assertEq(3, max(3, 1));
  assertEq(3, max(2, 3));
} // Fails (2nd assert)
```

(d) Resulting grade: 0%.

Figure 1. Example test suites and a faulty method, illustrating an impact on generated grades.

We seek to understand how such differences in test suites affect students' grades. We previously investigated this, by measuring grades generated for artificial faulty program variants, called mutants [4]. However, these mutants do not always perfectly reflect students' faults [5]. As such, we build upon our previous work in this paper, using real students' solution programs in place of artificial mutants, to better reflect the real

[1] BCS Press Office, "Record numbers choosing Computer Science degrees", https://www.bcs.org/more/about-us/press-office/press-releases/record-numbers-choosing-computer-science-degrees-new-data-reveals/

URI: https://hdl.handle.net/10125/79438
978-0-9981331-5-7

influence of test suites on grades. In addition, we hypothesise that suites which can isolate individual faults should also produce fair and consistent grades. Since fault diagnosability metrics can provide an estimate of this quality, we expand upon our previous work by also investigating the grading impact of three such metrics: density, diversity, and uniqueness [6]. We also revise our analysis technique, employing a relative importance analysis instead of examining the coefficients of linear models, in order to gain an accurate estimate of the impact of various test suite properties on grading consistency, even if the properties are correlated to one another. We consider the following research questions:

RQ1*: Do grades vary with different test suites?*

We conducted a standard deviation analysis on the grades generated by sampled test suites for students' programs. We found that the mean standard deviation of grades for each solution is $\sim 10.1\%$; different suites generate a wide variety of grades.

RQ2*: Which properties of test suites impact grades?*

To further investigate exactly how test suites produce the effect we observed in RQ1, we performed a relative importance analysis using measurements of various test suite properties and the changes in generated grades. We observe that several factors of test suites influence the generated grades for students' solutions, including the quantity of tests, coverage, uniqueness, and the ability to detect other students' faults and artificial mutants. Using our findings, we formed strategies for tutors to use in order to construct test suites that grade students' programs consistently.

We provide three key contributions in this paper: (1) evidence that different suites generate varying grades for students' programs; (2) a relative importance analysis revealing *how* a suite's properties influence these grades, with different results to the existing work; and (3) five strategies for tutors to improve their grading test suites.

2. Background

2.1. Existing Work

We previously investigated how different test suites generated varying grades, and how properties of the suites influence this variation [4]. We constructed test suites by randomly sampling from a large set of tests, and executed these test suites on simple artificial faulty variants of their associated subject programs, called mutants. Our results revealed that these varying suites had a significant variation in grades, with a mean standard deviation of 2.94%, despite the mean grade only being $\sim 96.5\%$. While we later found that the detection of mutants and students' faults are correlated [5], there are still some differences between mutants and real faults. In particular, mutants typically contain only one fault, while students' faults can contain several; they would likely produce different test results and grades. As such, we build upon our previous work by using students' solution programs in place of mutants. Our original investigation particularly focused on *adequacy* metrics; estimates of how effective a test suite is in identifying faults.

2.2. Coverage

Code coverage is a fairly simple and widely used adequacy metric; it evaluates how many components of a program are executed when running a test suite [7]. While many types of components can be used to form a coverage metric (e.g. conditional branches), we explore line coverage (C_τ) in this study due to its simplicity:

$$C_\tau = \frac{|\mathbb{C}^m|}{|\mathbb{L}^m|},$$

where $\tau =$ a given test suite, such that $\tau \subset \mathbb{T}$ (where $\mathbb{T} =$ set of all unit tests for the subject class); $m =$ subject class's model solution; $\mathbb{C}^m =$ model solution's lines covered by τ; and $\mathbb{L}^m =$ all lines in the model solution.

We found that coverage had the most significant impact on generated grades in our previous study. We also considered repeated coverage with *recoverage*, R_τ, but found that it had the lowest impact on grades.

$$R_\tau = \frac{\sum_{l \in \mathbb{C}^m} (|\mathbb{c}_\tau^l| - 1)}{|\tau| \cdot |\mathbb{L}^m|},$$

where $\mathbb{c}_\tau^l =$ set of tests in τ that cover l.

2.3. Mutation Score

Another means to evaluate adequacy is by executing a test suite on a series of artificial faults called mutants, which are generated using a mutation tool such as Pit [8]. The proportion of the mutants that are detected by the test suite is referred to as the mutation score (M_τ). Higher mutation scores indicate greater test adequacy; a suite that detects more mutants should detect more real faults.

$$M_\tau = \frac{|\mathbb{F}_\tau^{\mathbb{M}}|}{|\mathbb{M}|},$$

where $\mathbb{F}_\tau^{\mathbb{X}} =$ set of programs in $\mathbb{X}$ detected by τ; and $\mathbb{M} =$ set of mutants for the subject class.

We previously found that mutation score impacts generated grades, but to a lesser extent than coverage.

2.4. Diagnosability

Test suites can be used to estimate the location of a fault in a program, using a technique called fault localization [6]. A suite's fault localization accuracy—its ability to distinguish between possible faults—can be estimated using *diagnosability metrics*. We consider three such metrics in addition to the adequacy metrics that we previously explored; we hypothesise that a suite that can isolate particular faults generates different grades than one that cannot. *Density* (ρ_τ) measures the lines that are covered across every test in a suite:

$$\rho_\tau = \frac{\sum_{l|l\in\mathbb{L}^m}^{t|t\in\tau} A_{tl}}{|\tau| \cdot |\mathbb{L}^m|},$$

where A = an activity matrix ($|\tau| \times |\mathbb{L}^m|$), A_{tl} denotes whether line l was executed by test t.

When $\rho_\tau = 0$, no lines are ever covered; when $\rho_\tau = 1$, every test covers every line. Gonzalez-Sanchez et al. [9] show that the optimal density to isolate faults is $\rho_\tau = 0.5$. We use normalised density, $\rho'_\tau = 1-|1-2\rho_\tau|$, in this study; $\rho'_\tau = 1$ indicates ideal density [6].

Diversity evaluates the probability that two randomly selected tests differ in their coverage behaviour, measured by the Gini-Simpson index, $\mathcal{G}_\tau$ [6, 10].

$$\mathcal{G}_\tau = 1 - \frac{\sum_{\mathbb{a}\in\mathbb{A}} |\mathbb{a}| \cdot (|\mathbb{a}| - 1)}{|\tau| \cdot (|\tau| - 1)},$$

where $\mathbb{a}$ = set of all tests, $t \subseteq \tau$, that cover the same lines in m, $\forall l \in \mathbb{L}^m, \forall t, t' \in \mathbb{a}$: $A_{tl} = A_{t'l}$; and $\mathbb{A}$ is the set of all possible $\mathbb{a}$ for τ and $\mathbb{L}^m$.

It is possible for some lines to share the same coverage for every test in a suite; these lines form an *ambiguity group* ($\mathbb{g}$). Having few, large ambiguity groups poses a potential issue for grading; if the lines within an ambiguity group implement different parts of a specification, tests may not be able to distinguish which aspect a fault is associated with. *Uniqueness* ($\mathcal{U}_\tau$) reveals how many ambiguity groups are present in the program.

$$\mathcal{U}_\tau = \frac{|\mathbb{G}|}{|\mathbb{L}^m|},$$

where $\mathbb{g}$ = set of lines, $l \subseteq \mathbb{L}^m$, that are covered in the same way by all tests in τ, $\forall t \in \tau$, $\forall l, l' \in \mathbb{L}^m$: $A_{tl} = A_{tl'}$; and $\mathbb{G}$ = set of all possible $\mathbb{g}$ for τ and $\mathbb{L}^m$.

3. Research Methodology

3.1. Experiment Procedure

In this empirical study, we use students' implementations of five Java classes from three real programming assignments, and a set of unit tests for each (Section 3.3). Since we require a variety of test suites to investigate how suites influence grades, we generate test suites by sampling from this wider set of unit tests (Section 3.4). For each subject class, we execute every test on all of the students' solutions, as well as a correct model solution.

We store the results of these tests, and use the proportion of a suite's tests that pass for a solution to produce a *generated grade* (G^s_τ) [2]:

$$G^s_\tau = \frac{|\mathbb{P}^s_\tau|}{|\tau|},$$

where s = the student's solution under test; G^s_τ = grade generated by τ for s; and $\mathbb{P}^s_\tau$ = tests in τ that pass for s.

RQ1 As we aim to investigate how much different test suites generate different grades, we calculate the standard deviation of grades generated by our sampled test suites for each solution. We use this standard deviation instead of the absolute range of grades since some suites may only include tests that pass or fail, and would therefore have a typical grade range of $[0\%, 100\%]$. We also remove any test suites that only generate such extreme grades for every solution.

RQ2 In order to identify how different properties of test suites cause this grade variation, we perform a relative importance analysis on normalised test suite properties and changes in grades for each suite execution (Section 3.5). We estimate a change in grades by computing the *grade delta* (ΔG^s_τ); the difference between the execution's generated grade and the median generated grade for every execution of the same solution:

$$\Delta G^s_\tau = |G^s_\tau - \tilde{G}^s_{\mathrm{T}}|,$$

where T = set of all test suites for the subject class; and $\tilde{G}^s_{\mathrm{T}}$ = median grade generated for s by every suite in T.

3.2. Test Suite Properties

In order to address RQ2, we observe various properties of the sampled test suites, allowing us to evaluate the impact they have on the generated grades for each student's solution. In particular, we use coverage and mutation score, alongside the three diagnosability metrics that we discussed in Section 2. In addition, we include the detection of other students' solutions:

$$D^{\mathbb{S}\setminus\{s\}}_\tau = \frac{|\mathbb{F}^{\mathbb{S}\setminus\{s\}}_\tau|}{|\mathbb{S}\setminus\{s\}|},$$

where $\mathbb{S}$ = set of all solutions for the subject class; $D^{\mathbb{S}\setminus\{s\}}_\tau$ = the proportion of other solutions detected by τ; and $\mathbb{S}\setminus\{s\} = \mathbb{S}$, excluding s. While not every solution contains a fault, the least effective test suite will yield the minimum value for this metric, and the most effective will yield the maximum for a given solution.

We also consider the size of a test suite ($|\tau|$) as a property, for two main reasons. First, the size of a suite may directly influence grades. For example, if a large test suite has one failing test suite

Table 1. Subject Classes. We only include mutants that are detected by at least one test, and merge any mutants with equivalent test traces.

Task	Subject Class	Stdnts.' Sltns.	Muts	Tests		LoC
				Man.	*Evo*	
Chess	`Board`	45	55	18	14	26
	`Queen`	40	46	9	2	41
Wine	`Cellar`	36	40	16	15	272
Fitness	`DataLoader`	38	19	7	1	71
	`Questions`	38	65	20	30	263

for a solution, it will generate a higher grade than one with few tests and a single failure. Second, a suite's size may influence other properties, such as coverage and mutation score, as shown by Namin and Andrews [11]. Since the relative importance analysis that we employ is robust to correlated variables, by including $|\tau|$ as a property we can effectively control for its contribution to the other properties. We exclude the recoverage metric that we used in our original study [4], since diagnosability metrics also evaluate the repeated coverage of a program's lines, and are more well established in other studies [6, 9].

With the exception of the detection of other students' solutions, we measure all of the properties using the model solution, to simulate a tutor developing a new grading test suite. For metrics that require coverage information, we use JaCoCo [12] to record the coverage for every test execution, and store which lines are covered and uncovered by each test for the model solution. In order to evaluate the mutation score of a suite, we generate mutants with Pit [8]. We remove any mutants that are not detected by any tests, effectively normalising the mutation score to a range of $[0, 1]$. We also merge any mutants with the same behaviour for every test into a single mutant, such that every remaining mutant passes for a unique set of tests. This removes a potential source of bias, since some types and locations of mutants would otherwise be considerably more prevalent than others.

3.3. Dataset

We use students' solutions from three end of year assignments for an introductory undergraduate Java programming module in this study, as outlined in Table 1. Each assignment was completed by a different cohort of students. We conduct our empirical study on five subject Java classes from these assignments. For each subject class, we use a series of JUnit tests. Where available, we use the tutor's original grading tests for the assignments. We extend these test sets, to ensure that every line of code in each model solution that is relevant to its task's specification is covered. To do this, we use EvoSuite, an automatic test generation tool [13]. We also manually define new tests for each class, with the aim of maximising the variety of tests. We ensure that every test is valid by removing any that fail on the model solution. Table 1 summarises our subject classes, mutants, and test sets; *Man.* shows the manually defined tests, while *Evo.* shows the tests that we generated using EvoSuite.

3.4. Grading Test Suites

To investigate the possible changes in grades for our first research question, we designed our test suite sampler to simulate the iterative development of various individual test suites. In order to do this, we utilised the suite growth technique described by Chen et al. [14], in which a test suite is extended by randomly selecting an additional test that increases a given criterion, generating a new suite whenever a test is added. The first test suite is created by simply randomly selecting any test from the whole set. Our generator then selects tests that increase coverage; since a model solution implements the whole specification of a task, increasing coverage simulates adding tests that cover more of the specification. However, some of our subject classes can be fully covered by few tests, so the generator would quickly run out of new tests that would further enhance a suite. As such, once 100% coverage is reached, we change the generator's target to the suite's mutation score, as this is often a harder criterion to fulfil. This can be considered as more rigorously exercising a specification, ensuring that solutions do not include subtle mistakes. As with the mutation score property, we only use mutants with unique combinations of passing tests, to avoid bias from different proportions of similar mutants. Since it is possible for suites to detect every mutant, we use a "stacking" approach to continue growing the suite [14,15]; the suite's current mutation score is reset to zero by the generator, so every available test that detects any mutants can be selected. Once there is only one unselected test, our generator ends the generation run. We halt the generation here to avoid bias from a large number of identical suites that contain every available test. Similarly, we remove the initial suites with only one test each, since they will introduce a sampling bias by only generating grades of 0% or 100%. In order to account for the random element of suite generation, we repeat this process across 100 generation runs, and use all of the resulting test suites to generate grades for each of the solutions.

In order to investigate the impact of test suite properties on grades for RQ2, we must use a different approach, since constructing test suites with the goal to optimise coverage or mutation score would influence their impacts on grade delta. As such, we instead opt to construct test suites by randomly sampling tests from the pool. For each of 100 generation runs, our

random sampler constructed $|\mathbb{T}|$ test suites, the number of available unit tests for a subject class. Our generator split these $|\mathbb{T}|$ test suites equally between several test suite sizes; 20%, 40%, 60%, and 80%. Our generator generates each individual test suite by constructing a pool of the available test suites, and randomly selecting a test from it, removing the test from the pool in the process. Once the target number of tests is reached, the generator compares the constructed suite against other suites constructed during the same generation run, and adds it to this wider set if no equivalent suites are present. This is repeated for the run until the target number of tests, or an iteration limit of $4|\mathbb{T}|$, is reached.

3.5. Relative Importance

To evaluate the impact of each test suite property on generated grades for RQ2, we perform a relative importance analysis [16] on linear models with the observed properties as predictor variables, and the grade delta as the response variable. Relative importance analysis allows for the impact of a set of predictors on the response to be compared directly. Specifically, we use the relative importance measure first proposed by Lindeman et al. [17, 18]. This approach effectively calculates the average change in the R^2_{adj} of a linear model when a predictor is added, by adding the predictors to a linear model in different orders. This allows the relative importance of predictors (i.e. test suite properties) to be compared, even if they have some degree of correlation to one another, as is often the case for our properties. For example, test suites that have a higher coverage tend to have a higher mutation score [11]. This offers a benefit over simply comparing the magnitudes of a linear model's normalised (β) coefficients, which we used in our original study [4]; β coefficients do not accurately capture the contributions for correlated predictors. This measure also provides estimates of the relative importance in terms of the predictors' impacts on the variance of the response variable. This reveals the proportional impact of the properties on the change in grades, even if the linear model does not perfectly predict the change in grades. This also allows us to compare the impact of each property across different subject classes. We use bootstrapping to derive a confidence interval for this analysis, with 2000 runs per subject class and a confidence interval of 95%

We also calculate the Spearman's correlations (r_s) between the test suite properties and grade delta. We do not use these correlations to determine the impact of the properties, but instead use them to further explain the impacts of the test suite properties, such as if higher measurements of a property correspond to increasing or decreasing the divergence in generated grades. In addition, we include the β coefficients and p-values of each property for simple linear models that we derive with grade delta as the predicted variable, as in the analysis of our previous study. We use these observations to identify limitations with our previous analysis.

Table 2. Median grades of each solution and their median standard deviations, across all 30 runs. Rounded to 1 d.p.

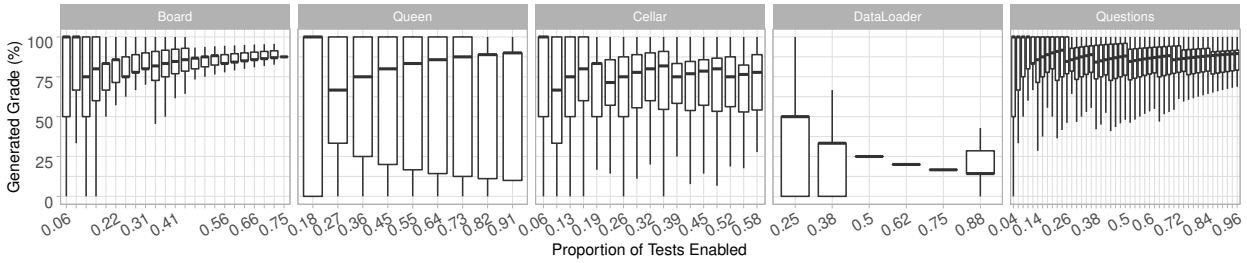
Subject Class	Median Grade, $\tilde{g}$	Std. Dev, σ_g
`Board`	83.3%	8.7%
`Queen`	100.0%	9.9%
`Cellar`	78.9%	13.6%
`DataLoader`	20.0%	14.2%
`Questions`	87.8%	3.8%
Mean	74.0%	10.1%

3.6. Threats to Validity

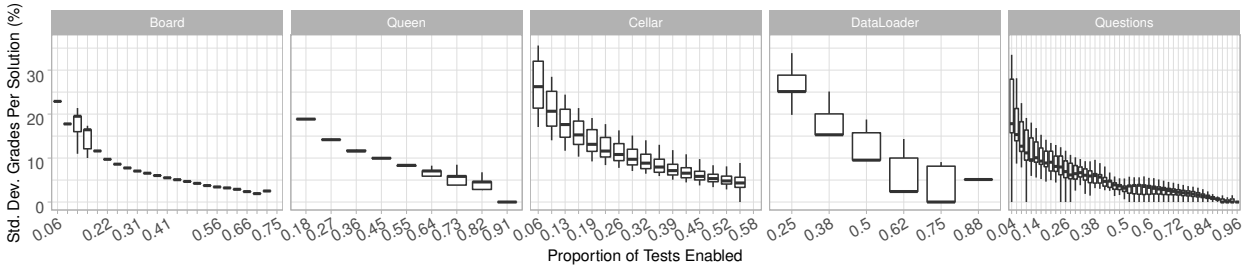
One potential threat to validity is that sampled test suites may not necessarily reflect the construction of real grading test suites. We mitigated this by explicitly choosing a guided sampling technique for RQ1, as a test suite that a tutor would write to cover more learning outcomes should, in principle, increase in coverage and mutation score as more tests are added. However, for RQ2 we cannot use this approach, as we are investigating the impact of several properties, including those which guide the suites for RQ1. As such, a random sampling approach is the only viable option for this dataset.

Another possible threat to validity is that some students' faults may not be detected by our tests. This is unlikely to impact our results in a meaningful manner, however; we manually analysed the faults present in the students' solutions and found that the vast majority of faults cause tests to fail.

For RQ2, we use bootstrapped confidence intervals, shown by the range bars in Figure 3. Bootstrapped confidence intervals may not truly reflect their target confidence level [16], posing a potential threat to validity. As such, we note that some properties may be more similar in how they impact students' grades than they appear in the data; properties with slightly less high importance estimates are possibly more important than those ranked above them. This should not heavily affect the general trends of relative importance; high or low estimates still provide a reliable indication of how the observed properties influence generated grades.



(a) Generated grades, per solution, per test suite.



(b) Standard deviation of generated grades for each solution.

Figure 2. Generated grade statistics of solutions for each subject class, across all 100 repetitions of suite generation. For ease of presentation, we removed the outliers.

4. Results

4.1. RQ1: To what extent do different test suites generate varying grades?

Table 2 shows the median grades and their standard deviations for all solutions and test suites, and their means across all subject classes. The mean standard deviation of grades per solution is 10.1%; different test suites yield grades that vary drastically for the same solution program. This is greater than the median standard deviation that we observed in our previous study ($\sim 2.6\%$), likely due to students' solutions containing several faults, while generated mutants only each include one. This may also be due to these subject classes having fewer tests; smaller test suites induce a greater change in grades if a single test fails.

Our results also reveal a limitation of our previous work; it is unreliable to evaluate how much of the possible change in grades from the median is represented by the standard deviation. For example, for `Queen` the median grade is 100%, and as such the possible change in grades must be 100%. Considering how we defined the possible change in grades, if its median grade was 99%, the possible change would be 1%; this minor difference in the median grade would produce a greatly different proportional impact of standard deviation. Instead, it is better to directly consider the standard deviation in a solution's grades alone.

Figure 2 shows the individual generated grades and grade standard deviations for each solution. We find that the subject classes have some variation in their behaviour, such as in the range of standard deviations at each test suite size, or the median grades. We conjecture that a programming task itself may affect how suites evaluate students' solutions, perhaps because the specification of how a class should be implemented may influence the mistakes that students make. For example, in `Cellar`, grades generated for some solutions by suites with 26% of the tests enabled have standard deviations of $\sim 16\%$, while others have standard deviations of $\sim 8\%$. This solution dependent variation in grades generated by different suites is a source of potential unfairness; some solutions' grades are affected by suites more than others. Comparatively, this effect is less prevalent for `Board`, where most solutions have similar standard deviations in grades; the influence of the test suites on their grades is similar between different solutions. We note that the specification of `Cellar` is more complex than that of `Board`. As such, some students' solutions may contain more faults for particular aspects of the program's specification, and thus would be more susceptible to differences in test suites than other solution implementations. However, even for `Board`, different suites still generate varying grades for a given solution; suites themselves have an influence on grades. We consider how suites influence such behaviours in more detail in RQ2 and Section 5.

RQ1 Results: Grades generated by different suites vary considerably, with a standard deviation of $\sim 10.1\%$ per solution. This standard deviation also varies between different solutions; the grades of some solutions are affected by the test suite more than others.

4.2. RQ2: Which properties of test suites impact grades?

Table 3 shows the results of our analysis for RQ2. This analysis differs from that of our original study in two key ways. The first difference is that we use a relative importance analysis instead of comparing the β coefficients of linear models. This is a more reliable approach, since the β coefficients of correlated predictors may not accurately reflect their contributions to the predicted variable, or one of the correlated predictors may not make a statistically significant contribution to a linear model. This can be observed for mutation score and the detection rate of other solutions in `Cellar`; these properties are correlated to one another ($r_s = 0.8$), and the predictor for mutation score is not statistically significant ($p = 0.64$). However, if the detection rate of other solutions is not included as a predictor, mutation score's contribution to the linear model becomes statistically significant ($p \leq 0.01$). Relative importance does not suffer from this problem, by virtue of summarising the contribution of a single predictor across linear models constructed by adding predictors in every possible order. Accordingly, we observe that the impact of mutation score is greater than the detection rate of other solutions for `Cellar`, and their impacts are similar overall.

The second change to our original analysis is that we use grade delta as the response variable of our analysis, rather than the simple generated grades. By assuming that the median grade of a solution is a fair grade, the distance of individual grades to this median represent their inconsistencies. As such, grade delta provides a metric of grading consistency, whereas the simple grades that we used in our previous study only provide an indication of the proportion of tests that fail for a solution. Grade deltas do have a limitation, however; if a solution's median grade is 0% or 100%, grade delta becomes a one-sided metric, equivalent to the proportion of tests that pass or fail for the solution; it becomes equivalent to our original analysis. This property is not what we aim to capture in our analysis. This issue occurs for `Queen`, where our randomly sampled test suites generate median grades of 100% for most solutions, similarly to the test suites that we use in RQ1. This is reflected in comparatively low relative importance estimates (and accordingly, R^2_{adj}) for the subject. Similarly, this effect also affects the correlations of the properties to grade delta. For other subject classes, the correlations are typically negative, indicating that suites with higher measurements of the respective properties produce lower grade deltas, and as such generate more consistent grades. However, for `Queen` these correlations are typically positive; instead this only reveals that increasing the values of the properties increases the proportion of tests that fail for most solutions and test suites. This subject also affects the mean observations; the mean p-value of uniqueness's correlation to grade delta ($\bar{p} = 0.17$) is heavily inflated by its correlation for `Queen` ($p = 0.87$). Excluding `Queen` from our results shows that the other correlations for uniqueness are significant ($\bar{p} \approx 0.00$). In effect, for `Queen`, grade delta does not truly measure grading consistency, since it is skewed by such an extreme median grade. As such, it essentially represents an outlier in our dataset.

The adjusted R^2 of each linear model represents the grade delta's variance that is captured by the model. This is equivalent to the sum of the relative importance estimates for each property; it represents how much the combined properties influence grading consistency. This is shown as a percentage by R^2_{adj} in Table 3; the mean across all five models is 25.08%; together, the properties account for 25.08% of the change in grades.

When evaluating the relative importance estimates, we find that, on average, the detection rate of other students' solutions is the most influential property with respect to a change in a solution's grades, accounting for 4.86% of the variance in grade delta. This is followed by the suite's size, uniqueness, and mutation score; with impacts of 4.66%, 4.43%, and 4.21% respectively. Code coverage has a lesser impact on generated grades (2.74%), followed closely by diversity (2.59%). Finally, density has the least impact on the change in grades; 1.6% on average. These results differ considerably to those of our previous work, where we instead found coverage to be the most important property with respect to grading consistency, followed by mutation score, and recoverage (which is analogous to diversity in this updated study). Aside from the changes to our experimental procedure, correlations between coverage and the diagnosability metrics may be responsible for this difference; part of the variance explained by diagnosability metrics may have been subsumed by the sole use of coverage in our previous study. The properties' contribution estimates also vary between the subject classes. For example, diversity has a very high contribution for `DataLoader`, but a very low contribution for the other classes. These differences are reflected in the contributions of the complete linear models; the R^2_{adj} for `DataLoader` is the highest of all of the subject classes. It is possible that this divergence in behaviour could be due to aspects of the subject class itself having an impact on grading consistency.

As Figure 3 shows, there is some overlap between the bootstrapped confidence bounds for some of the

Table 3. Summary of RQ2 analysis for all 100 random suite generation runs; including relative importance estimates (Est.), linear model normalised coefficients (β), and mean Spearman's correlations (r_s). Significance levels of β and r_s are reported as: * $= p < 0.05$; ** $= p < 0.01$; * $= p < 0.001$. $p < 0.01$ for each linear model.**

Subject Class	R^2_{adj}		Suite Size $\lvert\tau\rvert$	Coverage C_τ	Mut. Score M_τ	Other $D_\tau^{\mathbb{S}\setminus\{s\}}$	Density $\rho_\tau{}'$	Diversity $\mathcal{G}_\tau$	Uniqueness $\mathcal{U}_\tau$
`Board`	31.24	*Est.*	8.7%	3.13%	6.42%	6.54%	1.21%	0.7%	4.55%
		β	***-4.82	***6.11	***-5	***-4.69	***11.25	***10.05	***-8.58
`Queen`	15.62	*Est.*	1.51%	1%	3.24%	3.95%	4.09%	0.84%	1.01%
		β	***-8.26	*5.3	***-15.62	***18.58	***13.38	***4.21	*-5.82
`Cellar`	18.96	*Est.*	4.32%	3.67%	4.01%	2.81%	0.37%	0.06%	3.74%
		β	***-7.24	***-5.99	-0.17	***-6.83	***16.47	-0.67	***7.66
`DataLoader`	39.68	*Est.*	4.65%	2.55%	3.18%	7.03%	2.25%	11.27%	8.77%
		β	***11.43	***12.57	***-16.78	***-10.4	***-2.28	***-41.15	***-128.78
`Questions`	19.88	*Est.*	4.11%	3.38%	4.21%	3.96%	0.09%	0.07%	4.07%
		β	***-2.87	***3.13	***-6.15	***-5.79	***8.14	***9.02	*-3.83
Mean	25.08	*Est.*	4.66%	2.74%	4.21%	4.86%	1.6%	2.59%	4.43%
		β	***-2.35	***4.22	-8.74	***-1.83	***9.39	-3.71	*-27.87
		r_s	***-0.3	***-0.3	***-0.3	***-0.25	*0	***-0.02	-0.27

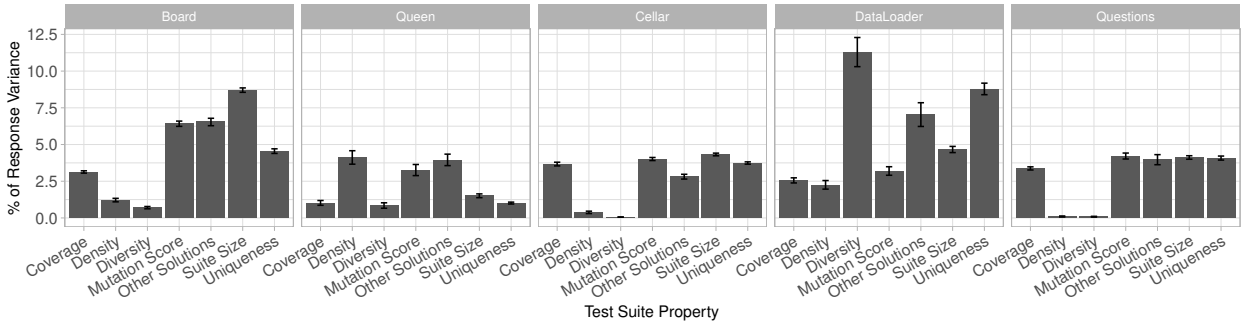


Figure 3. Relative importance of each test suite property, with respect to grade delta. The range bars denote the upper and lower bounds of a bootstrapped 95% confidence interval.

test suite properties. In these cases, the true order of relative importance for the properties may be slightly different, with one of the overlapping properties possibly outperforming the other. For example this effect can be observed for mutation score and the detection rate of other solutions for `Board`, `Queen`, and `Questions`. In these cases, these two properties should be considered as having a similar impact on grading consistency, since despite the overall estimate for one property being higher, the true order of their importance could be the opposite.

RQ2 Results: Most properties have a statistically significant impact on grades, especially suite size, the detection of other solutions, mutation score, and uniqueness. Suites with a higher measurements of these properties tend to generate more consistent grades.

5. Discussion

Since our results indicate that different test suites generate varying grades for the same solution, and that the properties of these suites influence grading, it would be beneficial to use our observations to control this effect as much as possible in grading. In this section, we offer strategies for tutors to improve the quality and consistency of their grading test suites. While tutors could simply write tests based solely on expected input and output, they may not evaluate solutions fairly. As such, our suggestions assume that a tutor executes their tests on a gold-standard model solution, as this ensures that tests are correct, and allows for the use of metrics to guide fair test suite design.

Suite Size Since the number of tests in a test suite is correlated to other properties of the test suite, such as its coverage or mutation score [11], we include it as a property for our relative importance analysis, as a means of controlling for its impact. As such, while a test suite's size has a relatively high impact on generated grades, accounting for $\sim$4.66% of the variance in grade delta, we cannot provide a specific recommendation for how many tests a tutor should include in their grading test suite. Instead, we note that fulfilling our other suggestions will likely require a tutor to create a series of high quality tests, the quantity of which will depend on the programming task that they assess.

Coverage Coverage has a moderate impact on grading consistency compared to the other properties, with a relative importance estimate of $\sim 2.74\%$. As such, while coverage does impact grading consistency, some other properties of a test suite have a greater impact.

Coverage has a negative correlation with grade delta, indicating that test suites with higher coverage produce grades that are closer to the median for the subject class; grading is more consistent. This is likely due to uncovered faults being impossible for a test suite to detect; covering more lines improves a suite's ability to detect faults, and as such generate grades that are not 100%, and closer to the median grade of the solution across every sampled suite in this study.

Suggestion 1: Tutors should aim to cover every line of the model solution with their grading test suite.

Mutation Score Mutation score has a fairly high impact on grades, with a relative importance estimate of $\sim 4.21\%$. In addition, mutation score is negatively correlated with grade delta; improving a suite's ability to detect mutants also results in more consistent grading. Like coverage, this is likely due to the property's ability to predict the adequacy of a test suite; detecting more mutants will improve a suite's ability to detect students' faults, and as such produce more consistent grades. This impact is greater for mutation score than coverage for each of the subject classes; it is more important to detect artificial faults than to cover lines of code in order to create more consistent test suites.

Suggestion 2: Tutors should use mutation testing to improve their grading test suites' abilities to detect faults, since this can prevent unfairness from some students' mistakes being missed.

Detection Rate of Other Students' Solutions The detection rate of other students' solutions has the greatest impact on grading consistency of any test suite property on average, accounting for $\sim 4.86\%$ of the variance in grade delta. This metric reflects the true adequacy of a test suite; its ability to detect students' faults. Since this metric is negatively correlated to grade delta, we can conclude that a test suite which detects more students' faults will produce more consistent grades.

However, this metric may be hard for tutors to use to improve their test suites. The metric would allow tutors to understand how many solutions have faults that are detected, but without manually identifying individual faults that are present in students' solutions, but it does not provide information for unknown faults that are present in students' solutions; these can only be identified and understood by using manual analysis. Comparatively, artificial mutants serve as known faults; it would be easier for a tutor to write tests that target undetected, but known mutants than unobserved students' faults.

Suggestion 3: If available, tutors can use existing students' solutions to inform the design of their grading test suites, but this could be challenging in practice; attaining an understanding of every fault in existing solutions requires manual analysis.

Density Normalised density has the lowest impact on grading consistency of any test suite property, with a relative importance estimate of ~ 1.6, reflecting its lack of correlation to grade delta. As such, we can conclude that the average proportion of lines that each test in a suite covers has little bearing on the suite's ability to generate consistent grades. Instead, other qualities related to coverage—such as diversity and uniqueness—may be more important.

Diversity A test suite's diversity can have an impact on its grading consistency, representing ~ 2.59 of the variance in grade delta, though this can be attributed exclusively to `DataLoader`, where it has the single greatest estimate of any property for any subject class, 11.27. From this, we can conclude that a test suite's diversity typically has almost no impact on grading consistency, except for in very specific circumstances. It is possible that this effect is related to how many tests in a sampled suite behave differently, and how well this sampled suite represents a typically sampled test suite (i.e. generate grades that are close to the median for each solution). For example, if the typical sampled test suite only contains tests that each have unique coverage behaviour, then test suites which have multiple tests with the same behaviour will generate grades that differ considerably from the median grade. This may be especially relevant to `DataLoader`, since it includes several tests that cover the same code, by virtue of the specification only defining the use of a single public method which can be called in a test.

Suggestion 4: Tutors should avoid writing disproportionately many tests which only exercise the same aspects of a programming task. If this is necessary, weights can be assigned to limit the impact that each similar test has on grading.

Uniqueness Uniqueness has a considerable impact on grading consistency, with a relative importance estimate of $\sim 4.43\%$. It is also negatively correlated to grade delta, indicating that test suites with more unique tests generate more consistent grades. Uniqueness likely leads to higher grading consistency since low uniqueness indicates that

some aspects of a programming task are evaluated by every test, solutions with faults in such aspects may be overly punished by being more likely to be detected than solutions with different faults. High uniqueness also indicates that every aspect of a program is evaluated at least once; no fault would be completely uncovered by any test, and as such would be more likely to be detected.

Attaining high uniqueness may pose a challenge for tutors however, since it is possible for a reference solution class to only have a single entry point, and as such this entry point must be evaluated by every test. This may require some redesign of the task's specification and reference solution to avoid this problem, such as requiring that additional public methods are used. It may be beneficial for tutors to run an analysis to identify lines or methods that are executed or missed by every test, such as by adapting and using Perez's diagnosability tool [6].

Suggestion 5: Tutors should avoid covering some lines of their model solution with every test, though this may require redesigning the programming task.

6. Conclusions & Future Work

In this paper, we have provided empirical evidence that different test suites generate varying grades for students' programs, and that observable properties of suites can influence these generated grades. Our findings differ from those our previous study [4]. We attribute this to both differences in our updated analysis, and differences between students' solutions and mutants, such as the quantity of faults, or their subtlety. We have also offered strategies for tutors to write fair and consistent grading test suites: (1) achieve 100% coverage on a model solution; (2) run tests against mutants to ensure that they detect faults; (3) analyse students' faults if possible, to gain an understanding of their mistakes; (4) avoid tests that cover the exact same lines unnecessarily; and (5) write tests that each exercise the program in unique ways.

Our study reveals several avenues for future research. First, replicating this study with test suites written by different tutors for the same programming tasks would be beneficial, since this would reveal how the properties impact grading consistency for real test suites. Second, further properties of test suites can be investigated, such as variants of diagnosability metrics that use mutants as test goals instead of covered lines; such metrics may be more relevant since they are grounded in fault detection rather than coverage. Finally, an automated tool to evaluate and identify deficiencies in a test suite with respect to these properties may be useful for tutors aiming to develop fair and consistent grading test suites.

References

[1] S. Krusche and A. Seitz, "ArTEMiS - An Automatic Assessment Management System for Interactive Learning," in *SIGCSE '18*, ACM, 2018.

[2] D. Insa and J. Silva, "Automatic assessment of Java code," *Comput. Lang. Syst. Struct.*, vol. 53, 2018.

[3] K. Dewey, P. Conrad, M. Craig, and E. Morozova, "Evaluating Test Suite Effectiveness and Assessing Student Code via Constraint Logic Programming," *ITiCSE 2017*, vol. 6, 2017.

[4] B. S. Clegg, P. McMinn, and G. Fraser, "The Influence of Test Suite Properties on Automated Grading of Programming Exercises," in *CSEET '20*, IEEE, 2020.

[5] B. S. Clegg, P. McMinn, and G. Fraser, "An Empirical Study to Determine if Mutants Can Effectively Simulate Students' Programming Mistakes to Increase Tutors' Confidence in Autograding," in *SIGCSE '21*, ACM, 2021.

[6] A. Perez, R. Abreu, and A. Van Deursen, "A Test-Suite Diagnosability Metric for Spectrum-Based Fault Localization Approaches," in *ICSE '17*, IEEE, 2017.

[7] Q. Yang, J. J. Li, and D. M. Weiss, "A Survey of Coverage-Based Testing Tools," *Comput. J.*, vol. 52, no. 5, 2009.

[8] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, "PIT: A practical mutation testing tool for Java (Demo)," in *ISSTA '16*, ACM, 2016.

[9] A. Gonzalez-Sanchez, H. G. Gross, and A. J. Van Gemund, "Modeling the diagnostic efficiency of regression test suites," in *ICSTW '11*, 2011.

[10] L. Jost, "Entropy and diversity," *Oikos*, vol. 113, no. 2, 2006.

[11] A. S. Namin and J. H. Andrews, "The Influence of Size and Coverage on Test Suite Effectiveness," in *ISSTA '09*, ACM, 2009.

[12] M. R. Hoffmann, E. Mandrikov, and M. Friedenhagen, "JaCoCo." http://eclemma.org/jacoco/, 2016.

[13] G. Fraser and A. Arcuri, "EvoSuite: Automatic test suite generation for object-oriented software," *SIGSOFT/FSE '11*, 2011.

[14] Y. T. Chen, A. Tadakamalla, M. D. Ernst, R. Holmes, G. Fraser, P. Ammann, R. Just, Y. T. Chen, R. Gopinath, A. Tadakamalla, M. D. Ernst, G. Fraser, P. Ammann, and R. Just, "Revisiting the Relationship Between Fault Detection, Test Adequacy Criteria, and Test Set Size," *ASE '20*, 2020.

[15] M. Harder, J. Mellen, and M. D. Ernst, "Improving test suites via operational abstraction," in *ICSE '03*, IEEE, 2003.

[16] U. Grömping, "Relative Importance for Linear Regression in R: The Package relaimpo," Tech. Rep. 1, 2006.

[17] R. H. Lindeman, P. F. Merenda, and R. Z. Gold, *Introduction to bivariate and multivariate analysis.* 1980.

[18] J. W. Johnson and J. M. Lebreton, "History and Use of Relative Importance Indices in Organizational Research," *Organ. Res. Methods*, vol. 7, no. 3, 2004.

[19] R. J. Lipton and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, 1978.

[20] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Trans. Softw. Eng. Methodol.*, vol. 1, no. 1, 1992.