Data-Driven Investigation into Variants of Code Writing Questions

Liia Butler*, Geoffrey Challen*, and Tao Xie†
*Department of Computer Science, University of Illinois at Urbana-Champaign
{liiamb2, challen}@illinois.edu
†Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
taoxie@pku.edu.cn

***Index Terms*—Question variants, code writing, exam questions, assessment**

***Abstract*—To defend against collaborative cheating in code writing questions, instructors of courses with online, asynchronous exams can use the strategy of question variants. These question variants are manually written questions to be selected at random during exam time to assess the same learning goal. In order to create these variants, currently the instructors have to rely on intuition to accomplish the competing goals of ensuring that variants are different enough to defend against collaborative cheating, and yet similar enough where students are assessed fairly. In this paper, we propose data-driven investigation into these variants. We apply our data-driven investigation into a dataset of three midterm exams from a large introductory programming course. Our results show that (1) observable inequalities of student performance exist between variants and (2) these differences are not just limited to score. Our results also show that the information gathered from our data-driven investigation can be used to provide recommendations for improving design of future variants.**

I. Introduction

In programming or software engineering courses, exams remain an integral part of assessing student knowledge. In recent years, the practice of paper-based exams has been disrupted by online assessments. In paper-based exams, questions that require code writing to solve, in short as code writing questions, require tedious manual grading, which is prone to grading inconsistencies across student submissions. In contrast, in exams based on online assessments (in short as online exams), a code editor, compiler, and automated test cases not only take the grading burden off instructors but also provide consistent grading and feedback to students. Physical facilities (e.g., computerized assessment centers) designated for these online exams also lift the burden off instructors from the overhead of holding an exam during an instructional period while still having consistency in exam administration.

Asynchronous exams, where students are not taking a particular assessment at the same time, help address an assessment facility's inability to support exams for a large class all at once. Asynchronous exams can be held over a given time period, and a student may choose a time slot during this period to take his/her exam. This form offers flexibility for a student to take an exam at a time most convenient for the student.

However, a major drawback of asynchronous exams is that they are more vulnerable to collaborative cheating than synchronous ones. Collaborative cheating is when one student who has taken the exam supplies information about the exam to another student who has yet to take it. This information creates a clear advantage for the student going into the exam.

In order to defend against collaborative cheating, one strategy of exam design is varying an asked question from student to student by means of *question variants*. We specifically address question variants that are manually written questions being put into a pool with other variants (typically two to four in total) and aim to assess the same learning goal. During exam time, one variant is then randomly assigned to a student from the pool for the particular question. The student sees only that variant but no other variants. Figure 1 shows an example of the flow from question variants to the assigned question.

Although the idea of having multiple variants of a question is desirable, instructors who prepare question variants struggle to accomplish two competing goals: effectively defending against collaborative cheating (e.g., making the variants sufficiently different) and fairly assessing students' same specific knowledge or skill (e.g., making the variants sufficiently similar). In particular, while creating sufficiently different variants for a question to fulfill effective defense, the instructors need to ensure that the variants can maintain fairness of the assessment, regardless of the presence of collaborative cheating. For example, the variants should be sufficiently similar where each variant captures similar knowledge requirements needed to succeed in answering the question. To accomplish these two competing goals, the instructors currently have to rely on only gut feeling or intuition.

Although previous research [1] shows that providing question variants can help defend against collaborative cheating (i.e., the first goal), there exists no previous research on whether and how well question variants can accomplish fair assessment in the context of online asynchronous exam settings (i.e., the second goal) for courses. In practice, the instructors may check to see whether scores attained by students across question variants are roughly equal, but doing so does not offer sufficient insights on the extent of how question variants can impact student performance in the course.

As the first attempt toward designing question variants to accomplish fair assessment in courses, in this paper, we propose to conduct data-driven investigation into analyzing question variants based on the set of submissions produced

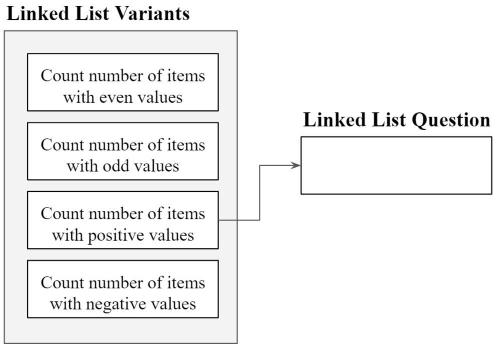
Linked List Variants
Count number of items with even values
Count number of items with odd values
Count number of items with positive values
Count number of items with negative values
Linked List Question

Fig. 1: Example of Linked List variant flow

by all the students in a class for these question variants. We first consider the overall distribution of the scores. We then consider the distribution of time spent on a given variant. Finally, we break down the duration into the amount of time spent in a given student activity such as resolving compiler error and other actions in response to assessment feedback.

We conduct our data-driven investigation into the question variants from three midterm exams from the Fall 2018 semester of a large introductory programming course. Our results show that (1) there are observable inequalities of student performance between variants, (2) variants whose scores are equal can still have significant differences in performance and effort required, indicating that relying on only scores may not be sufficient. Our results also show how information gathered from our data-driven investigation can be used to provide recommendations for improving design of future variants.

II. Background and Related Work

Programming or software engineering education poses its own unique challenges and opportunities in terms of exam creation possibilities and best practices. Our work touches on a number of areas related to programming or software engineering education, including test measurement, question variants, and code writing analysis.

Recent efforts in assessment, such as exam question randomization [2], parameterized questions for variant generation [3], [4], and autograding [5]–[8], have added appeal toward utilizing online and computer-assisted assessment, particularly in computer science settings [9], [10]. These efforts have brought a wave of work related to test measurement that both challenges and/or validates assumptions made regarding these innovations in terms of improving the assessment process, as well as making use of more easily accessible, richer data from the actual online assessments. Numerous efforts have taken a critical eye and evaluated these innovative approaches and revealed lessons and precautions related to these types of assessments, demonstrating that there is a strong need for such studies and that not all technological influence is positive [11]. Jordan et al. [12] point out a number of factors that should be of concern when considering variants of computer-marked questions that can lead to a question becoming unfair in terms of its variants, such as spelling in written responses and rounding in numerical responses. In addition to efforts more prominent in online assessments, a number of other efforts show points of concerns in the e-assessment environments themselves [13], as well as instructor shortcomings when utilizing these assessment approaches [14].

There exists a study of evaluating open-ended question variants [15]. Similar to coding question variants, several open-ended variants are derived from a base question, but question variants are evaluated in the context of natural language questions in a high-stake exam, where the time allowed for completing the essays is controlled and separate from other sections of the exam.

To the best of our knowledge, there exists no prior related work of studying code writing variants in online, asynchronous exams. However, code writing analysis has been widely studied in other aspects outside of our context and serves as important pillars to build upon. Logs and other programming data [16], [17], as well as considering students' perceived level of difficulty [18], provide valuable insight to instructors.

III. Example Question Variants

In this section, we illustrate an example question for a linked list and its four variants selected from the third exam of our dataset described in Section IV.

The value of each item in the linked list of items is stored in an object named "value" and the item includes a reference pointing (named "next") to the next item. The LinkedList class has a member variable named "start" to point to the head of the linked list. The assessment purpose for this question is to assess whether a student could write code to traverse a linked list. The task for each of the variants is to return the count of the number of the items that satisfy a certain criterion specified in the particular variant.

For Variants 1, 2, 3, and 4, students are instructed to count the number of items with even values, odd values, positive values, and negative values, respectively, in the linked list. A sample solution for each of the four variants is shown in Figures 2-5, respectively. In this question, the sample solutions for the four variants can be similarly structured to accomplish the respective variant tasks. Transforming the sample solution from one variant to another requires only a change to Line 8 to express how the current item's value should be compared (to select out those items satisfying the specified criterion) to increase the count variable.

IV. Data Context

We analyze the coding submissions from three 50-minute long, instructor-written midterm exams from the Fall 2018 semester of a large introductory programming course taught in Java. The midterms are taken in a special computerized assessment center (Section IV-A) on an online homework and testing platform (Section IV-B). In addition to evaluating the correctness of code, the course also evaluates code quality by utilizing Checkstyle [19], a linting tool that statically checks code against a specified coding standard (e.g., limiting the number of characters on a single line of source code).

```
protected int countEven() {
   int count = 0;
   Item temp = start;
   if (temp == null) {
      return 0;
   }
   while (temp != null) {
      if (temp.value % 2 == 0) {
         count++;
      }
      temp = temp.next;
   }
   return count;
}
```

Fig. 2: Example solution of count even variant

```
protected int countOdd() {
   int count = 0;
   Item temp = start;
   if (temp == null) {
      return 0;
   }
   while (temp != null) {
      if (temp.value % 2 != 0) {
         count++;
      }
      temp = temp.next;
   }
   return count;
}
```

Fig. 3: Example solution of count odd variant

```
protected int countPositive() {
   int count = 0;
   Item temp = start;
   if (temp == null) {
      return 0;
   }
   while (temp != null) {
      if (temp.value > 0) {
         count++;
      }
      temp = temp.next;
   }
   return count;
}
```

Fig. 4: Example solution of count positive variant

```
protected int countOdd() {
   int count = 0;
   Item temp = start;
   if (temp == null) {
      return 0;
   }
   while (temp != null) {
      if (temp.value < 0) {
         count++;
      }
      temp = temp.next;
   }
   return count;
}
```

Fig. 5: Example solution of count negative variant

A. *Computer-Based Testing Facility*

The Computer-Based Testing Facility (CBTF) [20] allows students to asynchronously take computerized assessments in a controlled environment. Instructors of a course set a range of days when the assessment is open for students to take, typically 3-5 days. From that range, a student may then select any available time slot over the exam period to go to the facility and take the exam. Multiple courses can have overlapping exam periods and exams from different classes can occur in the facility at the same time.

The facility is a computer lab that supports over 80 students at a time. The facility takes measures to decrease cheating risks. The facility staff verify student identities, ensuring that the correct students are at the correct time slot. The students must stow all belongings before heading to their randomly assigned seats. The staff then proctor once the examination period begins. In addition to the physical facility, the computers are also equipped with special control measures in attempt to prevent any unfair advantages including limited and controlled network and file system access, disabled access to removable storage (e.g., flash drives), and special screens on the monitors to prevent neighboring students from seeing what is on the screen.

B. *PrairieLearn*

The course uses PrairieLearn [21], an online educational system, for both homework assignments and as the platform for the online assessments held in the CBTF. The system shows an overview of all questions in the assessment, and a student can select which question to work on. The student can go to the next question from the previous question, or can go back to the overview and choose which question to work on next, so the student does not have to go in the chronological order and can switch between questions freely. The student can see the point value of the question and how many points the student has been awarded for the question at that time.

When a programming question is presented, the interface displays the question, prompt written by the instructor, and a simple code editor with optional starter code. The student can save, or save and submit for grading. When the student submits for grading, the submission is sent to an external server for grading, compiled and run against a set of test cases, and the results are sent back; the process takes just several seconds. The test cases are instructor-defined for the question. These test cases consist of both a large number of randomly generated inputs and manually defined inputs for explicitly checking edge cases, depending on appropriateness for the specific question.

There is some available feedback returned to the student. The student can see the output from the compiler if there is a syntax error, stack traces from runtime errors, Checkstyle errors, and assertion errors for failed test cases run against the autograder (students see only the Assertion Exception stack trace with the expected value and the actual value in the test case, but they do not see the test input or code of the actual test case). The students are allowed unlimited attempts at programming questions.

C. *Exam Descriptions and Question Variants*

In this section, we provide descriptions of the exams as well as a description of the questions and corresponding variants.

1) Exam 1: The first exam assessed imperative programming skills. 827 students attempted the exam. The exam was 17 questions long, with 12 multiple choice, 1 free response, and 4 programming questions, presented in that order to students. 3 of the 4 programming questions were written in variant format with 4 variants each, and were the 14th, 15th, and 16th questions on the exam. Partial credit was not available on the programming questions. The question breakdown is as follows:

- **E1Q1** Arrays - Ability to traverse an array
  - **E1Q1V1** Even Array Sum - Sum even values in array
  - **E1Q1V2** Odd Array Sum - Sum odd values in array
  - **E1Q1V3** Even Index Array Sum - Sum values from even indices in array
  - **E1Q1V4** Odd Index Array Sum - Sum values from odd indices in array
- **E1Q2** 2D Arrays - Ability to traverse a 2D array
  - **E1Q2V1** Count Equal - Count all array values equal to reference value
  - **E1Q2V2** Count Not Equal - Count all array values not equal to reference value
  - **E1Q2V3** Count Greater Than - Count all array values greater than reference value
  - **E1Q2V4** Count Less Than - Count all array values less than reference value
- **E1Q3** String Parsing - Ability to parse a string
  - **E1Q3V1** Get Salary ":" - Get salary from record formatted "Name:Position:Salary"
  - **E1Q3V2** Get Salary ";" - Get salary from record formatted "Name;Position;Salary"
  - **E1Q3V3** Get Yearly Salary ":" - Get salary from record formatted "Name:Position:Weekly Salary" and multiply by a 50 week working year
  - **E1Q3V4** Get Yearly Salary ";" - Get salary from record formatted "Name;Position;Weekly Salary" and multiply by a 50 week working year

2) Exam 2: The second exam assessed object-oriented programming skills. 735 students attempted the exam. The exam was 13 questions long with 10 multiple choice questions and 3 programming questions presented in that order to students. 2 of the 3 programming questions were written in variant format with 4 variants each, and were the 11th and 12th questions on the exam. Partial credit was available on the programming questions. The question breakdown is as follows:

- **E2Q1** Class Design - Ability to create class, implement constructor, area method, and override equals method.
  - **E2Q1V1** Equilateral Triangle - instance variable side length
  - **E2Q1V2** Rectangle - instance variables width and height
  - **E2Q1V3** Right Isosceles Triangle - instance variable side length
  - **E2Q1V4** Circle - instance variable radius
- **E2Q2** Comparable Object - Ability to create class to extend Comparable, implement constructor, getter and setter methods, and compare method.
  - **E2Q2V1** Dog - instance variable age
  - **E2Q2V2** Cat - instance variable height
  - **E2Q2V3** Turtle - instance variable speed
  - **E2Q2V4** Ferret - instance variable length

3) Exam 3: The third exam assessed knowledge over basic data structures and algorithms. 722 students attempted the exam. The exam was 16 questions long with 13 multiple choice questions and 3 programming questions. 2 of the 3 programming questions were written in variant format with 4 variants each, and were the 14th and 15th questions on the exam. Partial credit was available on the programming questions. The question breakdown is as follows:

- **E3Q1** Linked List - Ability to traverse a linked list
  - **E3Q1V1** Count Even - Count number of even-value items
  - **E3Q1V2** Count Odd - Count number of odd-value items
  - **E3Q1V3** Count Positive - Count number of positive-value items
  - **E3Q1V4** Count Negative - Count number of negative-value items
- **E3Q2** Binary Tree - Ability to traverse a binary tree
  - **E3Q2V1** Count Equal - Count number of nodes equal to reference value
  - **E3Q2V2** Count Not Equal - Count number of nodes not equal to reference value
  - **E3Q2V3** Count Greater Than - Count number of nodes greater than reference value
  - **E3Q2V4** Count Less Than - Count number of nodes less than reference value

V. Submission Structure

Each raw entry from the online educational system described in Section IV-B is processed into a Submission Data Type representation. Each Submission has the question id, user id, the score received, the actual submission code, the timestamp of when it was submitted, and Result of submission (discussed in more detail next). We also store the feedback that the student receives (Checkstyle errors, compiler errors, stack traces from exceptions, failed assertions from test cases) for use in manual examination, but is not required as part of our data-driven investigation.

Each Submission contains a Result categorizing the end result of the submission into one of five results: **CheckstyleError**, **CompilerError**, **RuntimeException**, **FailedTestCase**, and **CorrectSolution**. We provide a brief description of each result below:

- **CheckstyleError** - The submission contained code formatted not in compliance with a Checkstyle rule.
- **CompilerError** - The submission contained at least one compiler error.

- **RuntimeException** - An exception was thrown at runtime while test cases were executed on the submission.
- **FailedTestCase** - The submission failed at least one test case, i.e., an assertion in the test case is violated.
- **CorrectSolution** - The submission was considered correct.

Exactly one Result is determined for each submission by examining the score and parsing the grading results and feedback as stored in the raw entry.

VI. Investigated Dimensions of Student Performance

In this section, we discuss how we use three different dimensions of student performance to guide our data-driven investigation into these variants. Since we compare variants in terms of student performance, we do not include students who were assigned to a variant, but never attempted it. We provide the breakdown of overall students assigned and students who attempted in Table I, but during the investigation we consider only the students who attempted the question at least once.

We use the three dimensions to answer the following research questions:

- RQ1: What inequalities of observed student performance exist among question variants?
- RQ2: How can investigation results be used to derive recommendations for writing future question variants?

We next explain each of the three dimensions used in our data-driven investigation.

A. Score

We first consider the overall distribution of the highest scores of those who attempted a variant at least once. Since the score that a student can receive on a question is discrete, we perform a chi-squared test with α value 0.05 and the null hypothesis that there is no difference between scores received from the variants. Should the result show statistical significance, we then perform post-hoc pairwise comparisons of all variant combinations for the given question using the Bonferroni correction [22] where α is divided by the number of comparisons, in order to control the error rate of multiple comparisons. In this particular investigation, all questions have four variants, so we make 6 additional comparisons, so our threshold of statistical significance becomes 0.05/6, i.e., 0.0083.

B. Solving Duration

We approximate duration of attempting a question variant in order to understand whether there are differences in performance among variants behind the final score. Recall that for each exam question, there exists exactly one question variant (among all the question variants for the question) selected for this student to solve in the exam. An exam consists of a sequence of question variants.

1) Solving Duration of a Question Variant by a Student: Assume that a particular student submitted in total n submissions for all question variants in an exam. Then we first order (1) the exam start (treated as a fake submission denoted as $start$ in index 0, whose timestamp is the same as the timestamp of the student's first submission among all of her submissions), (2) all submissions from the student based on their timestamps (in index 1 till index n), and (2) the exam end (treated as a fake submission denoted as end in index $n + 1$, whose timestamp is the same as the timestamp of the student's last submission among all of her submissions) to form a sequence. Then we identify each location denoted as a vector $< i - 1, i >$ in between two nearby submissions s_{i-1} and s_i where $Q(s_{i-1}) \; ! = Q(s_i)$ where $Q(s)$ returns the question variant for which submission s is submitted, and $Q(s_0)$ and $Q(s_{n+1})$ (recall that s_o and s_{n+1} indicate the exam start and end, respectively) return special fake question variants that are different from any real question variant. These locations form a new sequence L.

Note that for each pair of nearby locations in L, denoted as $< i - 1, i >$ and $< j - 1, j >$, there is a property: for each submission k where $i <= k <= j - 1$, $Q(k)$ remains the same, denoted as question variant q being attempted during the pair of nearby locations. For a pair of nearby locations, we calculate the solving duration of q as below. The start time for q, denoted as $Start_q$, is the timestamp for s_{i-1}; here we assume that the student starts working on solving q as soon as the student submits s_{i-1}. The end time for q, denoted as End_q, is the timestamp for s_{j-1}; here we assume that the student stops working on solving q as soon as the student submits s_{j-1}. The solving duration for q during this pair of nearby locations is (End_q - $Start_q$), i.e., the difference between q's start time and end time.

Given that a student may switch back and forth between attempting different question variants, one question variant can be attempted during multiple pairs of nearby locations in L. To calculate the solving duration of q during the whole exam, we sum up q's durations calculated for all pairs of nearby locations where q is attempted. Figure 6 depicts this process.

2) Average Solving Duration for All Students: After we calculate a student's solving duration for question variant q, we further derive the average solving duration for q across all students who attempted to solve q. After we derive the average solving duration for each variant of a question, we perform a one-way ANOVA to test the null hypothesis that the average durations between these variants are equal. We again use α value 0.05. Again should the result show statistical significance, we then perform post-hoc pairwise comparisons using Bonferroni correction for an adjusted threshold of 0.0083.

C. Distribution of Effort

We further break down duration by looking at the distribution of efforts during the time that a student spent on a question variant. We identify five types of student efforts during attempting a question variant based on our Result categorization identified in Section V:

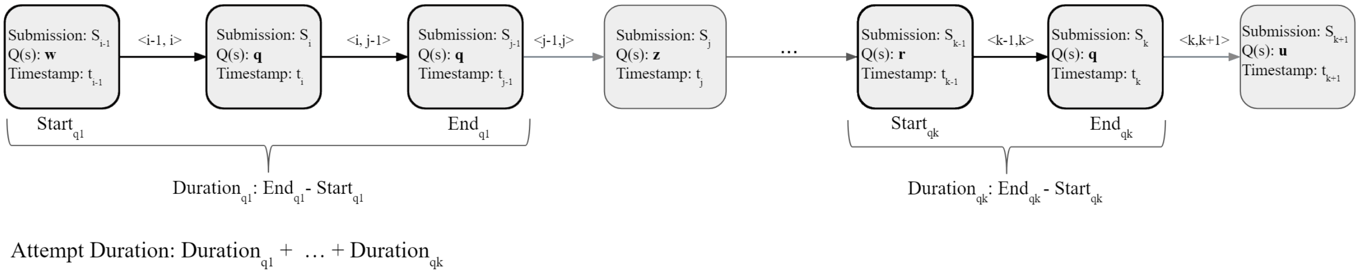

Fig. 6: Duration process

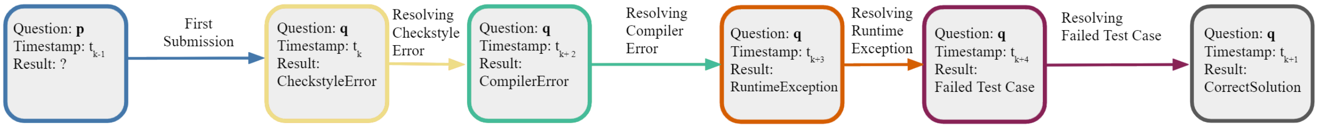

Fig. 7: Example distribution of effort for a submission timeline

- **First Submission**
- **Resolving Checkstyle Error**
- **Resolving Compiler Error**
- **Resolving Runtime Exception**
- **Resolving Failed Test Case**

We use the same submission sequence as the one constructed to approximate solving duration (Section VI-B). Recall that we insert the exam start and exam end as two fake submissions at index 0 and index $n+1$. We again identify each location (in between submissions) denoted as a vector $< i-1, i >$ where submission $Q(s_i)$ returns the question variant q currently under investigation.

If submission s_i is the first submission for question variant q, we use the timestamp from submission s_{i-1} as the *Start* for student effort's **First Submission**.

In the sequence of submissions by a student during an exam, if two nearby submissions s_{i-1} and s_i attempt the same question variant q, i.e., $Q(s_{i-1}) == Q(s_i) == q$, the **Resolving** effort of this student for question variant q is the difference in timestamps $T(s_i) - T(s_{i-1})$. We classify this Resolving effort into one of the four types listed earlier: Resolving Checkstyle Error, Resolving Compiler Error, Resolving Runtime Exception, and Resolving Failed Test Case, based on the Result of s_{i-1} (see Section V for the five possible results). Note that the Result of s_{i-1} shall not be **CorrectSolution** give that the student attempted the same question variant again in s_i.

In the sequence of submissions by a student during an exam, consider that two submissions s_i and s_j for attempting the same specific question variant q are not nearby but separated by other submissions for attempting other question variants, i.e., s_i, s_{i+1}, ..., s_{j-1}, s_j, where (1) $Q(s_i) == Q(s_j) == q$ and (2) $\forall\ k,\ Q(s_k)\ != \ q$ where $i+1 <= k <= j-1$. Here the **Resolving** effort of this student for question variant q is the difference in timestamps $T(s_j) - T(s_{j-1})$. We classify this Resolving effort into one of the four types listed earlier based on the Result of s_i. Here we assume that the student resumed effort on question variant q starting at timestamp $T(s_{j-1})$, and that effort produced s_j to address the issue encountered in submission s_i. Figure 7 shows an example submission timeline with each of the five student efforts for question variant q.

We sum up the time that a student spent in each resolving-effort category for a given question variant. We then record the average time and percentage time spent on a resolving-effort category for all students who attempted a given question variant. This metric helps provide us with further insight on a particular sticking point on a particular variant compared to its alternatives, as well as providing quick insight on student behaviors toward a particular question variant as a whole. We represent this dimension graphically.

VII. Results

We investigate via each of the three dimensions into the dataset described in Section IV. In this section, we describe the results and observations found from conducting our data-driven investigation.

A. Score Results

We apply the Score dimension onto the dataset. We provide the full statistics in Table I and then show the calculated p-values for each question in Table II.

As shown in Table II, one question, E3Q1, has a statistically significant difference in score among those who attempted question variants for this question. We therefore reject the null hypothesis that there is no difference between scores received from the variants for this question. We apply post-hoc Bonferroni tests to this question. We make comparisons between each of the 6 pairs of variants giving the Bonferroni-adjusted p-value of 0.0083 and find that there is statistically significant difference between variants 1 and 2, and between

TABLE I: Statistics of scores across question variants

Question	# Students Assigned	# Students Attempted	% Attempted	Avg Score	Avg Score Attempted
E1Q1V1	182	182	100	93.4	93.4
E1Q1V2	225	223	99.1	88.4	89.2
E1Q1V3	197	195	99.0	92.9	93.3
E1Q1V4	224	219	97.7	87.4	89.0
E1Q1 Total	**827**	**819**			
E1Q2V1	194	187	96.4	84.5	87.7
E1Q2V2	210	203	96.7	86.7	89.7
E1Q2V3	218	217	99.5	84.9	85.3
E1Q2V4	205	201	98.0	88.3	90.0
E1Q2 Total	**827**	**808**			
E1Q3V1	203	154	75.9	43.5	56.5
E1Q3V2	210	182	86.7	51.9	59.9
E1Q3V3	203	155	76.4	37.4	49.0
E1Q3V4	211	163	77.3	38.9	50.3
E1Q3 Total	**827**	**654**			
E2Q1V1	189	187	98.9	91.1	91.9
E2Q1V2	178	176	98.9	90.8	91.8
E2Q1V3	188	186	98.9	93.5	94.5
E2Q1V4	180	178	98.9	93.8	94.8
E2Q1 Total	**735**	**727**			
E2Q2V1	181	179	98.9	81.8	82.7
E2Q2V2	177	176	99.4	81.5	81.9
E2Q2V3	187	185	98.9	80.4	81.3
E2Q2V4	190	188	98.9	81.6	82.4
E2Q2 Total	**735**	**728**			
E3Q1V1	177	173	97.7	88.1	90.2
E3Q1V2	192	187	97.4	82.7	84.9
E3Q1V3	181	177	97.8	89.6	91.6
E3Q1V4	172	170	98.8	89.9	90.1
E3Q1 Total	**722**	**707**			
E3Q2V1	178	175	98.3	82.8	84.2
E3Q2V2	163	159	97.5	80.6	82.6
E3Q2V3	194	191	98.5	84.9	86.2
E3Q2V4	187	181	96.8	81.2	83.9
E3Q2 Total	**722**	**706**			

TABLE II: Question score p-values for attempted

Question	p-value
E1Q1	0.213
E1Q2	0.406
E1Q3	0.197
E2Q1	0.682
E2Q2	0.743
E3Q1	0.013
E3Q2	0.161

TABLE III: E3Q1 - Score pairwise p-values for attempted

E3Q1 Variant pair	p-value
V1 vs. V2	0.0016
V1 vs. V3	0.2745
V1 vs. V4	0.5662
V2 vs. V3	0.0006
V2 vs. V4	0.0345
V3 vs. V4	0.1317

variants 2 and 3. We provide all calculated p-values for these pairs of question variants in Table III.

Variant Count Odd is the variant that remains consistent in the two pairwise significant differences, but there is no significant difference for score pairwise against variant 4, Count Negative. We suspect that this result is due to a misconception that exists with the "%" operator in Java. In Java, "%" is the remainder operator, which is different than the mathematical modulus. The difference between the two reveals itself when students have to check for odd values. Students who fall into the common pitfall of using "n % 2 == 1" to check for odd values fail any test cases where n is negative; therefore, the result of "n % 2" is -1.

```java
public static int evenArraySum(int[] array) {
   int sum = 0;
   if (array == null) {
      return 0;
   }

   for (int i = 0; i < array.length; i++) {
      if (array[i] % 2 == 0) {
         sum += array[i];
      }
   }
   return sum;
}
```

Fig. 8: E1Q1V1 example solution

We are able to use results from the subsequent dimensions to better understand the effect of this pitfall.

B. *Results of Solving Duration and Distribution of Effort*

We apply the dimension of Solving Duration onto the dataset. We provide all p-values in Table IV. As shown in the table, four questions, E1Q1, E1Q3, E2Q1, and E3Q1, have

```java
public static int oddArraySum(int[] array) {
   int sum = 0;
   if (array == null) {
      return 0;
   }

   for (int i = 0; i < array.length; i++) {
      if (array[i] % 2 != 0) {
         sum += array[i];
      }
   }
   return sum;
}
```

Fig. 9: E1Q1V2 example solution

```java
public static int oddIndexArraySum(int[] array) {
   int sum = 0;
   if (array == null) {
      return 0;
   }

   for (int i = 0; i < array.length; i++) {
      if (i % 2 != 0) {
         sum += array[i];
      }
   }
   return sum;
}
```

Fig. 10: E1Q1V4 example solution

a statistically significant difference in duration and we reject the null hypothesis that the durations between these variants are equal for these four questions. We then apply post-hoc Bonferroni tests to the questions. Table V shows pairwise p-values for these four questions.

For E1Q1, we find that there is a statistically significant difference in duration between V1 and V2, as well as V1 and V4. The question across variants is to assess the ability to iterate through an array, access array elements, and using basic operators appropriately. However, we offer a possible explanation for the differences between variants. Programmatically testing for odd numbers can be less intuitive for a novice programmer than testing for an even or multiple of a number, which might require less time to think through. Therefore, the variants that require testing for odd may take more time. We note that variant V1 is the variant that requires a check on whether the value in the current index is even. Checking the value at the index aligns more with typical examples of array usage, rather than caring about the index. This factor could explain why there is statistical significance pairwise with variant V1. Using this duration information and looking at the distribution of efforts, instructors may opt to be mindful of these differences and ensure sufficient time to complete the exam, particularly since the question's average solving duration is about 4 to 6 minutes across the variants. However, there may be other cases where it would be more appropriate to modify variants to offer the same challenges (e.g., all variants require checking for the same remainder value).

For E1Q3, we find no statistically significant differences when comparing pairs of variants. However, we observe overall that the question has the greatest amount of invested time among the code writing questions for the first exam and the lowest performing as shown in Table I. Therefore, during exam construction, instructors shall watch out for cases of adding extra specifications to variants, such as those added to variants V3 and V4.

```java
public class Rectangle extends Shape {
   private int width;
   private int height;

   public Rectangle(int w, int h) {
      super("rectangle");
      this.width = w;
      this.height = h;
   }

   public double area() {
      return this.width * this.height;
   }

   public boolean equals(Object other) {
      if (other == null || !(other instanceof Rectangle
          )) {
         return false;
      }

      Rectangle r = (Rectangle) other;

      return (this.area() == r.area());
   }

}
```

Fig. 11: E2Q1V2 example solution

```java
public class Circle extends Shape {
   private int radius;
   Circle(int r) {
      super("circle");
      this.radius = r;
   }
   public double area() {
      return Math.PI * radius * radius;
   }
   public boolean equals(Object other) {
      if (other == null
         || !(other instanceof Circle)) {
         return false;
      }
      Circle circle = (Circle) other;
      return (this.radius == circle.radius);
   }
}
```

Fig. 12: E2Q1V4 example solution

For E2Q1, there is a statistically significant difference in duration between V2 and V4. A possible explanation is Rectangle being the only variant with two member variables while students may be more familiar with working around circles than other shapes. While for this particular question, instructors may just ensure sufficient time, future variants on class design may require instructors to be mindful of the types of classes to be designed.

Finally, for E3Q1, we find that there is a statistically significant difference in duration between V1 and V2, V2 and V3, and V2 and V4, respectively. Recall that this question is the same as the one with statistical significance in the score dimension.

TABLE IV: Question average duration p-values

Question	p-value
E1Q1	9.250E-05
E1Q2	0.272
E1Q3	0.019
E2Q1	0.049
E2Q2	0.747
E3Q1	6.213E-08
E3Q2	0.246

TABLE V: Question average duration pairwise p-values

Question	Variant pair	p-value
E1Q1	V1 vs. V2	0.0004
	V1 vs. V3	0.0529
	V1 vs. V4	0.0001
	V2 vs. V3	0.0399
	V2 vs. V4	0.7840
	V3 vs. V4	0.0144
E1Q3	V1 vs. V2	0.7974
	V1 vs. V3	0.0116
	V1 vs. V4	0.0399
	V2 vs. V3	0.0163
	V2 vs. V4	0.0539
	V3 vs. V4	0.7176
E2Q1	V1 vs. V2	0.1584
	V1 vs. V3	0.9534
	V1 vs. V4	0.1471
	V2 vs. V3	0.1480
	V2 vs. V4	0.0077
	V3 vs. V4	0.1687
E3Q1	V1 vs. V2	4.7794E-05
	V1 vs. V3	0.6189
	V1 vs. V4	0.6828
	V2 vs. V3	2.8454E-06
	V2 vs. V4	5.4979E-06
	V3 vs. V4	0.9289

We again see that variant 2, Count Odd, remains consistent in the pairwise comparisons across all those with significant differences. We are able to use this dimension and results of the third dimension, visualizing the distribution of effort, shown in Figure 13, to better understand and narrow down the focus to confirm our suspicions of the misconception of the "%" operator as suggested in the previous section. When we utilize the third dimension and examine Figure 13 and the percentage of time spent on various types of resolving effort, we see that considerably more effort is spent on resolving failed test cases than other types of resolving effort. For this variant, upon manual inspection of randomly selected submissions that result in a FailedTestCase result, we see that the pitfall of misunderstanding the "%" is indeed present.

We provide our recommendation toward future questions that contain variants requiring use of the remainder operator but are not purposefully assessing the ability to use the remainder operator. We suggest that these questions use the operator consistently across variants (e.g., testing for multiples of a given number).

VIII. Discussion

While only requiring information extractable from submission records does allow our data-driven investigation to be more generally applicable, external and human factors are not captured in this analysis, with two major limitations.

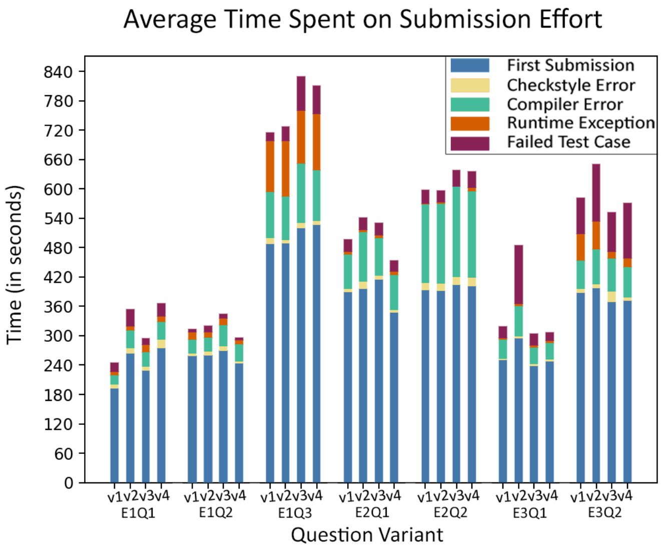

Fig. 13: Distribution of effort results

First, since our analysis requires submissions from a student to analyze, and thus we exclude those variants that do not have any submission, we do not know whether an excluded variant has influence on why a student does not submit. For example, a student could have looked at the prompt and deliberately skip it, run out of time, and do not even see the question, or even attempt to solve the question and simply forget to submit.

Second, our duration approximation is limited. We made the decision that the start time of attempting a question variant begins at the end of the preceding submission or when the student begins the exam. This decision naturally makes the assumption that students immediately begin focusing on the next question variant, while during the exam, there may have been other events that happen before the student begins attempting the question variant, such as deciding which question variant to attempt next. Likewise, duration between attempts for the same question variant considers all time invested on the next attempt, while there may have been other occurrences unaccounted for, such as a student's thinking about a previous question variant while still working on the current question variant. This decision would lead our analysis to overestimate solving duration, particularly for when a student first begins a question variant, as more time is likely to pass during this period that the student is assumed working on the attempt. Likewise, the student may be addressing issues other than/in addition to the issue as indicated in a submission's Result used in the Distribution of Effort dimension.

IX. Conclusion and Future Work

In this paper, we have presented our data-driven investigation for analyzing variants of code writing questions in a setting. We use three dimensions to analyze these variants: Score, Solving Duration, and Distribution of Effort. We have found that there exist observable differences among the variants. Moreover, we have shown that question variants that may be considered equivalent in terms of score could still contain differences in terms of student effort. Finally, we have used information derived from applying our data-driven investigation into a dataset of exam question variants to show that results from our data-driven investigation can be used to provide recommendations for improving future variants.

While our work offers the first step, there is still much more to be done in future work to understand variants of code writing questions. The natural next step would be to attempt to understand how effective these variants are in defending against collaborative cheating. When we sufficiently understand defense effectiveness and fair assessment, respectively, and the relationship between them, we can begin to construct best practices for designing these variants.

Acknowledgment

This work is supported in part by by NSF under grant no. CNS-1564274, CCF-1816615. Tao Xie is the corresponding author.

References

[1] B. Chen, M. West, and C. Zilles, "How Much Randomization is Needed to Deter Collaborative Cheating on Asynchronous Exams?" in *Proceedings of the Fifth Annual ACM Conference on Learning at Scale*, 2018, pp. 62:1–62:10. [Online]. Available: http://doi.acm.org/10.1145/3231644.3231664

[2] M. West, M. Silva, and G. L. Herman, "Randomized Exams for Large STEM Courses Spread via Communities of Practice," in *2015 ASEE Annual Conference & Exposition*, 2015, pp. 26.1302.1–26.1302.15. [Online]. Available: https://peer.asee.org/24639

[3] S. Sosnovsky, O. Shcherbinina, and P. Brusilovsky, "Web-based Parameterized Questions as a Tool for Learning," in *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, 2003, pp. 309–316. [Online]. Available: https://www.learntechlib.org/p/14944

[4] I.-H. Hsiao, P. Brusilovsky, and S. Sosnovsky, "Web-based Parameterized Questions for Object-Oriented Programming," in *Proceedings of E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*, 2008, pp. 3728–3735. [Online]. Available: https://www.learntechlib.org/p/30206

[5] M. Sherman, S. Bassil, D. Lipman, N. Tuck, and F. Martin, "Impact of Auto-grading on an Introductory Computing Course," *Journal of Computing Sciences in Colleges*, vol. 28, no. 6, pp. 69–75, 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2460156.2460171

[6] D. Jackson and M. Usher, "Grading Student Programs Using ASSYST," in *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, 1997, pp. 335–339. [Online]. Available: http://doi.acm.org/10.1145/268084.268210

[7] S.-L. Hung, I.-F. Kwok, and R. Chan, "Automatic Programming Assessment," *Computers & Education*, vol. 20, no. 2, pp. 183 – 190, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/036013159390086X

[8] C. A. Higgins, G. Gray, P. Symeonidis, and A. Tsintsifas, "Automated Assessment and Experiences of Teaching Programming," *Journal on Educational Resources in Computing*, vol. 5, no. 3, pp. 5:1—5:21, 2005. [Online]. Available: http://doi.acm.org/10.1145/1163405.1163410

[9] R. Lobb and J. Harlow, "Coderunner: A Tool for Assessing Computer Programming Skills," *ACM Inroads*, vol. 7, no. 1, pp. 47–51, 2016. [Online]. Available: http://doi.acm.org/10.1145/2810041

[10] K. M. Ala-Mutka, "A Survey of Automated Assessment Approaches for Programming Assignments," *Computer Science Education*, vol. 15, no. 2, pp. 83–102, 2005. [Online]. Available: https://doi.org/10.1080/08993400500150747

[11] J. Bull, "Computer-Assisted Assessment: Impact on Higher Education Institutions," *Journal of Educational Technology Society*, vol. 2, no. 3, pp. 123–126, 1999. [Online]. Available: http://www.jstor.org/stable/jeductechsoci.2.3.123

[12] S. Jordan, H. Jordan, and R. Jordan, "Same But Different, But Is It Fair? An Analysis of the Use of Variants of Interactive Computer-marked Questions," *International Journal of e-Assessment*, vol. 2, no. 1, pp. 6:1–6:12, 2012. [Online]. Available: https://ijea.org.uk/index.php/journal/article/view/28

[13] J. Prather, R. Pettit, K. McMurry, A. Peters, J. Homer, and M. Cohen, "Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 41–50. [Online]. Available: http://doi.acm.org/10.1145/3230977.3230981

[14] J. Wrenn, S. Krishnamurthi, and K. Fisler, "Who Tests the Testers?" in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 51–59. [Online]. Available: http://doi.acm.org/10.1145/3230977.3230999

[15] B. Bridgeman, C. Trapani, and J. Bivens-Tatum, "Comparability of Essay Question Variants," *Assessing Writing*, vol. 16, no. 4, pp. 237 – 255, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1075293511000328

[16] J. P. Munson, "Metrics for Timely Assessment of Novice Programmers," *Journal of Computing Sciences in Colleges*, vol. 32, no. 3, pp. 136–148, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?id=3015220.3015256

[17] N. C. C. Brown, A. Altadmri, S. Sentance, and M. Kölling, "Blackbox, Five Years On: An Evaluation of a Large-scale Programming Data Collection Project," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 2018, pp. 196–204. [Online]. Available: http://doi.acm.org/10.1145/3230977.3230991

[18] D. Spinellis, P. Zaharias, and A. Vrechopoulos, "Coping with Plagiarism and Grading Load: Randomized Programming Assignments and Reflective Grading," *Computer Applications in Engineering Education*, vol. 15, pp. 113 – 123, 2007. [Online]. Available: https://doi.org/10.1002/cae.20096

[19] O. Burn, "Checkstyle." [Online]. Available: http://checkstyle.sourceforge.net/

[20] C. Zilles, R. Deloatch, J. Bailey, B. Khattar, W. Fagen-Ulmschneider, and C. Heeren, "Computerized Testing: A Vision and Initial Experiences," in *2015 ASEE Annual Conference & Exposition*, 2015, pp. 26.387.1–26.387.13. [Online]. Available: https://peer.asee.org/23726

[21] M. West, G. L. Herman, and C. Zilles, "PrairieLearn: Mastery-based Online Problem Solving with Adaptive Scoring and Recommendations Driven by Machine Learning," in *2015 ASEE Annual Conference & Exposition*, 2015, pp. 26.1238.1–26.1238.14. [Online]. Available: https://peer.asee.org/24575

[22] O. J. Dunn, "Multiple Comparisons Among Means," *Journal of the American Statistical Association*, vol. 56, no. 293, pp. 52–64, 1961. [Online]. Available: http://www.jstor.org/stable/2282330