Superficial Code-guise: Investigating the Impact of Surface Feature Changes on Students' Programming Question Scores

Max Fowler
University of Illinois
Urbana, IL, USA
mfowler5@illinois.edu

Craig Zilles
University of Illinois
Urbana, IL, USA
zilles@illinois.edu

ABSTRACT

Assessing student performance on programming questions is important for introductory computer science courses, both for student learning and for ensuring students demonstrate competence. Part of being a competent programmer includes the ability to transfer learning from solved to analogous problems. Additionally, particularly in computer-based and online assessment, mitigating cheating efforts is another important consideration. One way to mitigate cheating is by randomly selecting from large pools of equivalent questions.

In order to produce large pools of questions quickly, we used a permutation strategy to rapidly make new question variants by altering existing questions' surface features. In this work, we present the results of our first set of surface feature permuted questions in an introductory Python course. We find surface feature permutations to be an effective way to produce questions of a similar difficulty to other *new* questions for students while mitigating potential cheating. However, we also see permutations expose potential student knowledge fragility and transfer concerns, as performance on permutations of homework questions is not strictly better than performance on questions that are entirely new on assessments.

CCS CONCEPTS

• **Social and professional topics → CS1**; **Student assessment**.

KEYWORDS

CS1, introductory computer science, online assessment, programming surface features

ACM Reference Format:
Max Fowler and Craig Zilles. 2021. Superficial Code-guise: Investigating the Impact of Surface Feature Changes on Students' Programming Question Scores. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432413

1 INTRODUCTION

Assessing student performance is a key component of education. Not only is it an important tool for determining student success, well crafted assessments can also provide direct learning benefits [16, 24, 38]. Of particular interest is how students perform on new questions they *should* have learned the solution to: that is, new questions with analogous solutions to questions students have already solved. Students' success on analogous problems may also indicate where students suffer from fragile knowledge, lacking the depth of understanding needed to transfer what they already "know" to analogous problems [3].

However, for assessments to be most effective, some care must be taken to mitigate cheating. This is particularly true in computer-based and online testing environments, in which students both perceive more ability to cheat and self-report cheating at a higher rate [8, 40]. One way to potentially mitigate student cheating efforts is to use randomized question pools in the construction of assessments, thereby building variance into the assessments students receive and reducing the chance of unauthorized collaboration and answer copying from other sources. Building a large enough question pool for randomization can be a challenge, particularly for the assessment of students' programming skill in introductory computer science. Writing question prompts that necessitate students answering programming prompts with small code snippets or small, custom functions is a non-trivial task.

In order to find an efficient way to produce sets of programming questions for creating large random pools on online assessments while still allowing students to see problems similar to previously solved ones, the research team implemented a structured process to create question variants through the application of surface feature permutations. Specifically, new permuted variants are written based on existing questions by changing their surface features, the specific elements of a problem such as variable name and data type, without significant structural change to possible programming solutions. We aim to answer the following research questions:

- **RQ1**: Does surface feature permutation produce question variants of similar difficulty to their base forms?
- **RQ2**: Does surface feature permutation create sufficient variants to mitigate cheating?

The rest of the paper is organized as follows. In Section 2, we briefly address related work. Section 3 is dedicated to the permuted questions written in the study, and Section 4 provides the context for the data collection. We present results in Section 5 and discuss them in Section 6. Limitations are addressed in Section 6.5. Finally, we conclude.

2 RELATED WORK

2.1 Question surface features and knowledge fragility

Novices, in fields from physics [7] and math [20] to computer engineering and computer science [9, 19, 21, 30, 33], have been known to struggle with questions' surface features due to lacking the encoding of experts. For example, Johnson-Glauch et al. found digital logic students conflated inputs and circuit states due to their shared position in a table, attending more to location than what the information was [21] and Ichinco and Kelleher found that experts attended to and recalled structural information and meaning more than novices when recalling memorized code snippets [19]. Novices may also attend to the wrong information in a problem or struggle to focus on relevant details. Eye-tracking studies in novice programmers show novices tend to have a linear focus, rather than an execution order or pattern-based focus as found in experts [4, 34].

There is some existing literature on the effect of surface features on students' performance on questions. Hernandez et al. found surface feature-born confusion to be consistent and pervasive in a physics course between electricity and magnetism questions [17]. Weston et al. investigated surface features' impacts on students tracing photosynthesis reactions [42]. Students luckily did determine the prompt was the important part of these questions, rather than the specific plant and how the plant details were presented, but were confused by and led astray by the word "process".

Related to this surface feature focus is the concept of knowledge fragility [3]. Knowledge fragility is one of the reasons Kennedy and Kraemer present for misconceptions in introductory computer science [22]. Luxton-Reilly and Petersen found this to be particularly troublesome in cases where knowledge fragility compounds with multiple concepts in a question [26].

2.2 Transfer and problem solving

Students' ability to transfer previous learning to new problems is of relevance. Bassok describes problem-solving transfer as the process by which students are reminded of a previous problem and retrieve that problem's solution to solve a novel problem. Key to transfer's success is how similar the original and new problem are to each other: that is, if the problems are analogous [1]. Structural-mapping theory draws a distinction between problems' surface features and structures, where structural similarity allows for successful transfer between problems [1, 14]. Despite structural similarity being more useful for the successful application of analogical solutions, both structural and surface features of problems contribute to students' selection of potential solutions [18].

Some work has looked at how to structure content for analogical transfer in computer science. Muller's pattern-oriented instruction (POI) was based around helping students perform analogical transfer by explicitly instructing students in reusable algorithmic patterns [29]. Haberman and Muller present POI and abstract data type based approaches to teaching abstraction. One of the difficulties students faced was distraction with surface features leading to incorrect selection of patterns [15]. Enström considers how POI can help students with dynamic programming problems [12].

Transfer research in computer science also considers conceptual transfer between natural language and mathematics concepts and computer science problem solving, especially in regards to where students struggle with differences in definition and behavior [2, 31, 35]. More recently, work has begun to investigate how novices transfer between their first and second programming language, as well [37].

2.3 Cheating and cheating mitigation

Student cheating is a well documented phenomenon. The Fraud Triangle Theory suggests that choosing to cheat requires three factors: perceived incentive to cheat, perceived opportunity to cheat, and rationalization of the act [11]. Feinman [13] provides a detailed literature review on proctoring and cheating; here, we highlight a few examples of student cheating and mitigation strategies. Myriad studies show a significant body of students admitting to cheating in online assessment, with students self-report cheating at a higher rate [23, 36, 39, 40]. There is also a perception that cheating is easier in online and electronic exam contexts, although also that some fraud detection may be easier in such environments [8]. In addition, students may exhibit a "learning to cheat" behavior if repeated exposure to an insecure testing environment increases the perceived opportunity to cheat [5].

Proctoring is one obvious way to dissuade cheating behavior [28]. Authorship checks are another way to dissuade some forms of cheating, although may still be ineffective at catching students sharing materials [27]. Honor codes and fostering good campus moral culture are shown to decrease the amount students cheat [28, 32], although are not as effective as stern warnings [10] in online contexts. Finally, question pool randomization is shown to have some mitigating impact on cheating and is pursued both through large question pools and through algorithmic question generation [5, 6, 25, 28].

3 SURFACE FEATURE PERMUTATIONS

This paper analyzes a collection of questions with surface feature permutations that were written during the Spring 2020 semester. We detail the development process below.

First, we defined a set of surface feature permutations. These permutations were decided by the research team based on what details of a problem we could change such that the surface details were different without significantly impacting the structure the question's solution code would require. Table 1 gives a list of the surface features and a description of what the permutation entailed. Keys were used to mark which permutations a given question variant has applied.

After deciding on permutations to use, we selected a set of base questions. Base questions are the questions the permuted variants are created from. These were selected both from existing questions from Fall 2019 and new questions written in Spring 2020. Further, some base questions were used on homework assignments in the Spring, while other base questions were "hidden" questions - that is, questions that only appeared on exams.

Variants were created by selecting which permutations were given to which base questions. A variant could have 1–3 permutations applied at once. Because not every permutation is applicable to every base question, permutations were not uniformly distributed

Table 1: The surface feature permutations used for question development in Spring 2020. While not ordered on empirical study, they are listed in the research team's perception of loosely ascending order of complexity. This complexity is in terms of the surface change's impact on the base question's prompt.

Permutation	Key	Description
Base	None	The base form of a question.
Var Name Change	V	Change the name of parameters used in a question prompt.
Function Name Change	F	Change the name of the function to be written in a question prompt.
Order Swap	O	Swap the order of parameters in a function.
Prototype Added	A	Add a function prototype to a prompt without one.
Prototype Removal	R	Remove the provided function prototype from a prompt.
Constant Change	C	Change constants involved in a prompt, e.g., "first 5 elements ..." to "first 3 elements ...".
Polarity Reverse	P	Change question "polarity" e.g., "positive numbers" to "negative numbers", "first" to "last".
Data Type Change	D	Change the data types of one or more parameters, e.g., strings to lists.

during the semester or on each assessment. The count of how many times a permutation appeared in a given assessment and what base type they had is shown in Table 2.

To provide an example of the permutation writing process, we present an examples of two bases and two permuted variants below. Note that the permuted variant features each permutation's keys in the question name in order to track which features were permuted. The surface features which change between the questions are italicized and bolded and function prototypes, when provided, are marked with an asterisk:

> **progAddLastThreeOfList**: Create a function called sum_last_three that takes a single argument of type list of numbers. Your function should return the sum of the ***last three*** elements of the given list. You can assume that the list always has at least three elements.
> **progAddLastThreeOfList_PC**: Create a function called sum_first_five that takes a single argument of type list of numbers. Your function should return the sum of the ***first five*** elements of the given list. You can assume that the list always has at least five elements.

> **progCondStringIsSingleLetter**: In the function below, return True if the input string parameter consists of a single lowercase letter; otherwise, return False.
> * def ***is_a_single_letter***(***string_arg***):
>
> **progCondStringIsSingleLetter_VF**: In the function below, return True if the input string parameter consists of a single lowercase letter; otherwise, return False.
> * def ***typographic_checker***(***word***):

4 COURSE CONTEXT AND DATA

All questions were developed for and deployed in a large enrollment (capped at 600), introductory, non-major CS course at a large U.S. university. The course is focused on basic programming principles in Python and Excel. Students in the course typically do not have prior experience in programming. Our questions were developed and deployed in the Spring 2020 semester. The course's student population is predominantly freshmen (67%) and sophomores (21%) and is reasonably gender balanced (258 women/307 men).

Summative assessment takes the form of exams and quizzes. With COVID-19, both quizzes and exams were online and self-proctored, but quizzes were asynchronous and lower stakes and exams were synchronous and higher stakes. When referring to a specific assessment, we will use quiz # or exam #, but otherwise we generalize to exam. All of these exams, as well as student homework, were deployed using the online platform PrairieLearn [41].

All programming questions, whether or not they were permuted, were one of two types. Questions are called *homework* questions if they appeared first on homework in the course. In this course, some fraction of the exam questions were drawn from the homework to motivate students to understand the homework. Questions which appeared for the first time on exams are called *hidden* questions. This technically includes all permuted variants, but does not include all *bases*. Further, appearing on an exam once does not preclude questions from appearing on later exams. In order to reduce the likelihood of students being able to memorize exact answers, collaborate with classmates, or copy from a saved list of answers, exams are written using question pools. Each student receives a random question from each pool when their exam is constructed. Partial credit is assigned by automated test cases and multiple attempts are given to students to complete questions for a reduced number of points on each attempt.

The programming questions in this study were deployed in two fashions during the Spring semester. First, the majority (n = 29) of *base* questions used were part of students' formative homework assignments. Following deployment of *base* questions on homework, surface feature permutations of these questions appeared on subsequent exam question pools. The first *base* question appeared on a homework in the 4th week, while the first use of a permuted question was on an exam in the course's 6th week.

The second deployment condition for questions was less common (n = 15), where *base* questions were deployed for the first time on exams rather than on homework. In this case, the *base* questions are just as hidden as the surface permuted variants of the questions. While every student had access to every homework deployed *base* question, not all students would have seen each hidden *base* and permuted question during the semester because students receive different draws from question pools.

Table 2: The amount of times each permutation appears in a random question pool on a given assessment, as well as how many bases appear and the total number of questions with that permutation. Quiz 1 (Q1) and Exam 0 (E0) are omitted, as they did not have any permuted questions. The table also gives the count for how many were based on homework questions versus hidden questions.

Permutation	E1	Q2	E2	Q3	E3	Q4	E4	Total	Homework Based	Hidden Based
Base	9	14	12	25	16	25	35	44	29	15
Var Name Change	4	3	1	4	1	4	10	11	6	5
Function Name Change	6	3	3	5	2	5	19	21	15	6
Order Swap	3	3	0	2	1	2	6	8	3	5
Prototype Added	2	0	1	2	1	2	7	7	6	1
Prototype Removal	3	3	1	5	0	5	9	12	5	7
Constant Change	4	2	5	5	1	5	21	21	13	8
Polarity Reverse	7	4	4	7	1	7	18	21	11	10
Data Type Change	0	0	1	8	2	8	16	16	12	4

Of the almost 37,000 programming question submissions this semester, 22,454 submissions were collected from just the questions used for feature permutation. There are a total of 213 questions in the exams' question pools: 14% are homework base questions, 22% are permutations of homework bases, 7% are hidden base questions, 17% are permutations of hidden bases, and the remaining 40% are questions that don't have permutations. Student scores are reported on a scale from 0 (no credit) to 1 (full credit).

5 RESULTS

5.1 Homework base questions always have higher average scores

Our first observation is that base questions from the homework pools have the highest average score compared to hidden bases and variants to a statistically significant degree. Figure 1 shows the average score out of 1 for four types of questions: homework bases, their permutations, hidden bases, and their permutations. The questions are bucketed by permutations. A Kruskal-Wallis test was used to check if there were significant differences between the buckets for each permutation. All permutations reported $p < 0.001$, indicating that each bucket was significantly different from at least one other bucket.

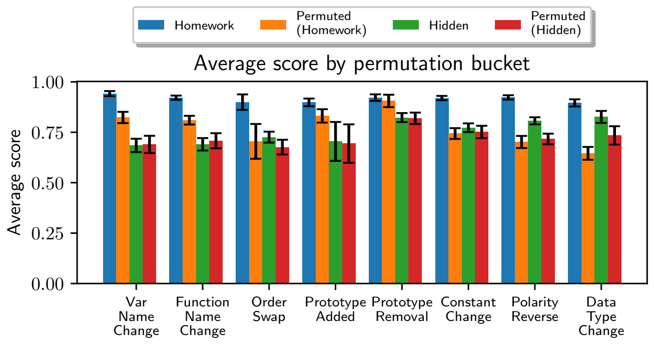

Figure 1: The average score by surface feature bucket with 95% confidence intervals. Questions previously used on homework always had a higher average score.

5.2 Only some permutations have a significant impact on student performance

Our second observation is that only some permutations have a significant impact on a student's ability to get a question correct. The permutations whose regression coefficients were significant are marked in 2. Specifically, we find *Order Swap*, *Polarity Reverse*, and *Data Type Change* decrease and *Prototype Removal* increases a student's chance of getting a question correct. We find this result by fitting a logistic regression model to our data. The model is written in terms of the surface features permuted, whether questions were used as bases, whether questions were homework or hidden questions, and the anonymized student to whom the submission belongs. Our model, over all submissions, is detailed below:

$$c = \sigma\left(\sum_{i=1}^{m} \alpha_i s_i + \sum_{k=1}^{t}\sum_{j=1}^{n} \phi_k p_{kj} + \beta q_j\right) \tag{1}$$

In the formula above, σ is the logistic function, m is the total number of students, n is the total number of questions, and t is the total number of permutation types. We enumerate the model's inputs, output, and resulting regression coefficients below:

- Givens
  - s_i: the student the submission belongs to
  - q_j: a categorical combination of the question's base and whether that base was hidden (0 or 1) in the form *base_hidden*, e.g. the question **progAddLastThreeOfList_PC**'s q is progAddLastThreeOfList_1
  - p_{kj}: whether question j has permutation k applied
  - c: Whether or not a particular question submission was correct, 0 or 1
- Regression Coefficients
  - α: an estimate of a given student's ability
  - ϕ: permutation k's estimated impact on question difficulty
  - β: the estimated difficulty for question j's base and whether j was hidden

It is not straightforward to translate ϕ directly into student performance. To better estimate their impact on average student performance we used our regression model to predict whether a random student would get a question correct with and without a specific permutation. We found that the change in probability of submitting

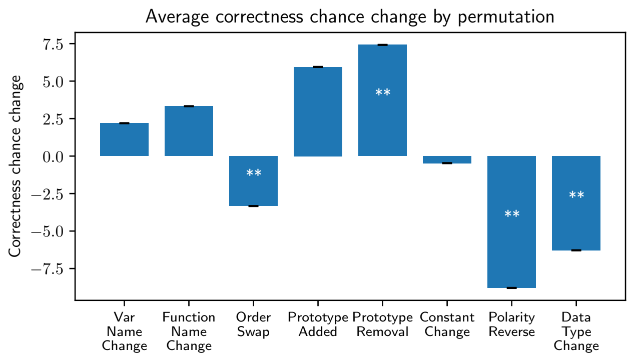

Figure 2: The predicted change in correctness chance with a given permutation applied. The permutations with significant regression coefficients are marked, with $p < 0.01$ (*) and $p < 0.001$ (**).

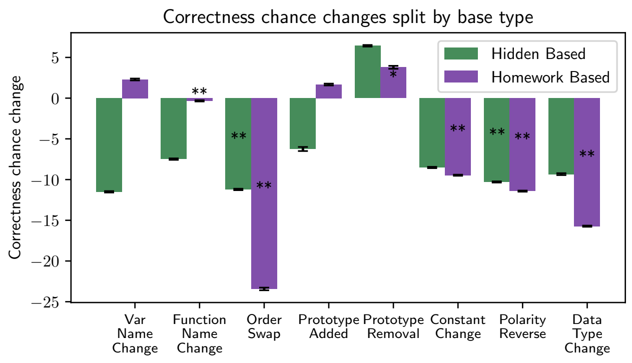

Figure 3: Predicted correctness changes when base conditions are split, with regression coefficient significance indicated ($p < 0.01$ (*) and $p < 0.001$ (**)).

a correct answer to a question based on the surface features permuted is <=±9%. Figure 2 plots the average change in correctness by permutation. Each bar represents an average over 100 simulated question sets.

5.3 Permutations have a more dramatic impact when we consider only one kind of base

We were curious how specific permutations predicted student correctness if we *only* considered one kind of base. We split the data into two sets and then refit our regression model on both sets. From the first set, we removed all homework questions and their permutations. From the second set, we removed all hidden bases and their permutations and all homework questions but *not* their permutations. This makes it a set of all hidden questions where permutations were *only* applied to homework questions. We again predict the chance in correctness with these two smaller models.

Permutations have the most dramatic effect on students' correctness when applied to homework questions. Figure 3 presents the estimated correctness change averages with significant regression coefficients marked. The majority of changes to correctness chances remain fairly small in each split, at <=±11%. However, the split does present interesting directional shifts from the full question set: hidden based *Var Name Change*, *Function Name Change*, and *Prototype Added* questions are harder to get correct than their base, rather than easier as indicated by Figure 2.

There are some startling impacts with the permutations of homework based questions, specifically *Order Swap*'s 23% chance decrease and *Data Type Change*'s 15% chance decrease in getting a permuted question correct. Additionally, when split, we see *Constant Change* has a larger impact on predicted correctness than when the model is fit on all the data.

6 DISCUSSION AND LIMITATIONS

6.1 Permutations are mostly neutral as far as question difficulty

Our first takeaway from the results is that the permutations are mostly neutral in their impact on question difficulty. If we consider just the hidden bases, this neutrality is shown in the averages from Figure 1, where the averages of hidden bases and their permutations are close together. We also saw this neutrality when predicting the change in submission correctness, as the impact on correctness from each permutation was only <=±9%. Even when only considering hidden questions only, the chance was still <=±11%.

This answers **RQ1** in that, when questions are in the same condition (hidden), permutations do not appreciably affect difficulty in aggregate. In general, if we change a problem in a small way, the variant remains about as difficult.

6.2 Homework questions are much easier than others - exposure or cheating?

We see base questions from homework have higher performance across the board than other types of questions. Given permutations do not appreciably impact question difficulty in Section 6.1, these higher average scores likely have a different explanation than "permutations make these questions difficult."

The higher average score on homework base questions could indicate an exposure effect. Students may have practiced a number of times on these specific questions in the process of doing their homework and preparing for exams. It is also generally easiest to perform an already completed task again than a new task. As such, the high average score may be a result of experience with these specific questions.

The other possible explanation is that some students may have cheated. The exams were online due to COVID-19, increasing the ability for students to access lists of their previous answers. Further, quizzes in particular are asynchronous, which further opens the door to not only accessing an answer list, but also unauthorized collaboration between students who finished the quiz and students who had not yet taken the quiz.

6.3 Prototype Removal may be easy to cheat

With respect to **RQ2**, only one permutation consistently increased the likelihood a student would get a question correct regardless of if we used all the data or split our data: *Prototype Removal.* We believe this permutation may specifically be an easy permutation to cheat on in our data set. Of the 12 questions with *Prototype Removal* as a permutation, 5 of them *only* had the prototype removed and another 3 of them *only* had the prototype removed and function name changed. This would have been trivial for any student with access to a list of answers or a classmate to copy-and-paste an answer in. As such, we do not believe *Prototype Removal* is particularly successful cheating deterrence.

6.4 Some permutations may stop cheating or indicate fragile learning

The results are not all discouraging in regards to **RQ2**. *Order Swap*, *Constant Change*, *Polarity Reverse*, and *Data Type Change* all have negative regression coefficients and deccrease the chance of a student getting a question correct. This impact is more dramatic when we consider only questions with the same base condition, particularly homework based permutations as seen in Figure 3. This supports Figure 1, particularly in how much lower the homework based permutations' averages are compared to their base questions. If students were attempting to cheat on these questions, it appears that these particular permutations did help thwart those efforts.

These results potentially paint a concerning picture with respect to student learning for some students. Ideally, students who mastered homework questions would be able to transfer their knowledge to those questions' permutations. This does not seem to be the case with the "more complex" surface feature permutations of *Constant Change*, *Polarity Reverse*, and *Data Type Change*, as these permutations always reduced submission correctness chance. This is also supported by the score difference between the base and permutations for homework bases versus the smaller gap between hidden bases and their permutations for these three surface features in Figure 1. For these permutations, permuted homework questions were functionally just as difficult to students for these surface features as the hidden bases. We are unwilling to claim *Order Swap*'s results could imply fragile knowledge given the significantly large error bar homework based questions have.

6.5 Limitations

The current work has some clear limitations. The nature of question development leads to a few core issues. Firstly, the permutations are not equally represented across the question set used in the various course assessments, both within a given assessment as well as across the semester. Part of this is due to the course content, as permutations like *Data Type Change* were better suited for later content. While the research team tried to write enough questions for each permutation to be represented at least 10 times, we failed in the case of *Order Swap* and *Prototype Added* due to few questions featuring multiple arguments or lacking prototypes at first. Some permutations which are easier to feature, such as *Var Name Change*, were also underrepresented.

As question variants could feature multiple surface feature permutations, the question set also did not guarantee that permutations appeared both individually and with other permutations in relatively equal amounts. To better be sure of the impact individual changes have on questions, it would have been preferable to have a better spread of variants which permute a single surface feature. This also would have helped address some of the underrepresented permutations in the question set.

The data collected here does not conclusively give us the reason why students did worse on permutations of questions than on homework questions. The driving factor could be being a hidden question or it could be that we missed some impact of the permutations because all of the variants appear only on exams. In a future semester, we would like to swap questions such that permutations appear on homework and their bases appear on exams. This can help further tease out if there is an actual difference in difficulty between these homework bases and the surface feature variations or if a question being hidden is the prime decider of student performance. Having a small group of students complete a base question, then a permutation of that question, in an interview setting would also be valuable. These interviews would let us qualitatively analyze what knowledge students retain from the base question and what parts of a surface feature change cause students difficulty.

7 CONCLUSION

We present the results of using surface feature permutations to produce larger randomized programming question pools on computer-based and online assessments. From the results, it appears that specific surface feature changes do not produce questions appreciably more difficult than other new *hidden* questions in random pools. It also appears that surface feature permutation is generally effective at thwarting simple memorization or answer list use on the part of students to cheat. However, these results may also point to novice knowledge fragility, as in some instances permuting homework questions still produces student performance in the same range as regular hidden questions.

Future work avenues are abundant. For starters, running a similar study with more numerically equivalent permutation counts, more single permutation variants, and variance in terms of allowing for permuted questions to appear on homework can help further clarify permutations' impacts on question difficulty. Following up from that, qualitative analysis to investigate *why* students may struggle to complete permutations of questions they have already seen can help elucidate how students learn coding snippets and programming solutions. Finally, it would be interesting to see the same permutation process applied in other classes, both Python-based (for replication) and those using other languages (to investigate the transferability of these permutation strategies to other languages).

ACKNOWLEDGMENTS

We thank the graduate students of the Computing Education Research group at the University of Illinois at Urbana-Champaign and Profs. Geoffrey Herman, Colleen Lewis, Karrie Karahalios, and Matthew West for their feedback on the analysis and multiple drafts of the paper.

REFERENCES

[1] Miriam Bassok. 2003. *Analogical Transfer in Problem Solving.* Cambridge University Press, 343–370. https://doi.org/10.1017/CBO9780511615771.012
[2] Jeffrey Bonar and Elliot Soloway. 1985. Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers. *Human–Computer Interaction* 1, 2 (1985), 133–161. https://doi.org/10.1207/s15327051hci0102_3

[3] Guy Brousseau and Michael Otte. 1991. The Fragility of Knowledge. In *Mathematical Knowledge: Its Growth Through Teaching*, Alan J. Bishop, Stieg Mellin-Olsen, and Joop Van Dormolen (Eds.). Springer Netherlands, Dordrecht, 11–36. https://doi.org/10.1007/978-94-017-2195-0_2

[4] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *2015 IEEE 23rd International Conference on Program Comprehension*. 255–265.

[5] Binglin Chen, Sushmita Azad, Max Fowler, Matthew West, and Craig Zilles. 2020. Learning to Cheat: Quantifying Changes in Score Advantage of Unproctored Assessments Over Time. In *Proceedings of the Seventh ACM Conference on Learning @ Scale* (Virtual Event, USA) *(L@S '20)*. Association for Computing Machinery, New York, NY, USA, 197–206. https://doi.org/10.1145/3386527.3405925

[6] Binglin Chen, Matthew West, and Craig Zilles. 2018. How Much Randomization is Needed to Deter Collaborative Cheating on Asynchronous Exams?. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale* (London, United Kingdom) *(L@S '18)*. Association for Computing Machinery, New York, NY, USA, Article 62, 10 pages. https://doi.org/10.1145/3231644.3231664

[7] Michelene T. H. Chi, Paul J. Feltovich, and Robert Glaser. 1981. Categorization and Representation of Physics Problems by Experts and Novices*. *Cognitive Science* 5, 2 (1981), 121–152. https://doi.org/10.1207/s15516709cog0502_2

[8] Aparna Chirumamilla, Guttorm Sindre, and Anh Nguyen-Duc. 2020. Cheating in e-exams and paper exams: the perceptions of engineering students and teachers in Norway. *Assessment & Evaluation in Higher Education* 0, 0 (Jan. 2020), 1–18. https://doi.org/10.1080/02602938.2020.1719975

[9] Michael J. Clancy and Marcia C. Linn. 1999. Patterns and Pedagogy. In *The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education* (New Orleans, Louisiana, USA) *(SIGCSE '99)*. ACM, New York, NY, USA, 37–42. https://doi.org/10.1145/299649.299673

[10] Henry Corrigan-Gibbs, Nakull Gupta, Curtis Northcutt, Edward Cutrell, and William Thies. 2015. Deterring Cheating in Online Environments. https://doi.org/10.1145/2810239

[11] Donald R. Cressey. 1950. The Criminal Violation of Financial Trust. *American Sociological Review* 15, 6 (1950), 738–743.

[12] E. Enström. 2013. Dynamic programming - Structure, difficulties and teaching. In *2013 IEEE Frontiers in Education Conference (FIE)*. 1857–1863. https://doi.org/10.1109/FIE.2013.6685158

[13] Lena Feinman. 2018. *Alternative to Proctoring in Introductory Statistics Community College Courses*. Ph.D. Dissertation. Walden University.

[14] Dedre Gentner. 1983. Structure-Mapping: A Theoretical Framework for Analogy*. *Cognitive Science* 7, 2 (1983), 155–170. https://doi.org/10.1207/s15516709cog0702_3

[15] B. Haberman and O. Muller. 2008. Teaching abstraction to novices: Pattern-based and ADT-based problem-solving processes. In *2008 38th Annual Frontiers in Education Conference*. F1C–7–F1C–12. https://doi.org/10.1109/FIE.2008.4720415

[16] M. K. Hartwig and J. Dunlosky. 2012. Study strategies of college students: Are self-testing and scheduling related to achievement? *Psychonomic Bulletin and Review* 19 (2012), 126–134.

[17] Eder HernÃ¡ndez, Esmeralda Campos, Pablo Barniol, and Genaro Zavala. 2020. The effect of similar surface features on studentsâ€™ understanding of the interaction of charges with electric and magnetic fields. (2020). http://hdl.handle.net/11285/636305

[18] Keith J. Holyoak and Kyunghee Koh. 1987. Surface and structural similarity in analogical transfer. *Memory & Cognition* 15, 4 (July 1987), 332–340. https://doi.org/10.3758/BF03197035

[19] Michelle Ichinco and Caitlin Kelleher. 2017. Towards better code snippets: Exploring how code snippet recall differs with programming experience. In *Visual Languages and Human-Centric Computing (VL/HCC), 2017 IEEE Symposium on*. IEEE, 37–41.

[20] Matthew Inglis and Lara Alcock. 2012. Expert and Novice Approaches to Reading Mathematical Proofs. *Journal for Research in Mathematics Education* 43, 4 (2012), 358–390. https://doi.org/10.5951/jresematheduc.43.4.0358

[21] Nicole Johnson-Glauch, Dong San Choi, and Geoffrey Herman. 2020. How engineering students use domain knowledge when problem-solving using different visual representations. *Journal of Engineering Education* 109, 3 (2020), 443–469. https://doi.org/10.1002/jee.20348

[22] Cazembe Kennedy and Eileen T. Kraemer. 2018. What Are They Thinking? Eliciting Student Reasoning About Troublesome Concepts in Introductory Computer Science. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '18)*. Association for Computing Machinery, New York, NY, USA, Article 7, 10 pages. https://doi.org/10.1145/3279720.3279728

[23] Mark M Lanier. 2006. Academic integrity and distance learning. *Journal of criminal justice education* 17, 2 (2006), 244–261.

[24] J.T. Laverty, S.M. Underwood, R.L. Matz, L.A. Posey, J.H. Carmel, M.D. Caballero, C. L. Fata-Hartley, D. Ebert-May, S. E. Jardeleza, and M. M. Cooper. 2016. Characterizing college science assessments: The three-dimensional learning assessment protocol. *PLoS ONE* 11, 9 (2016), e0162333. https://doi.org/10.1371/journal.pone.0162333

[25] Chang J. Lee. 2018. Automated Randomization of Test Problems for Cheating Prevention. *World Journal of Research and Review* 6, 2 (2018).

[26] Andrew Luxton-Reilly and Andrew Petersen. 2017. The Compound Nature of Novice Programming Assessments. In *Proceedings of the Nineteenth Australasian Computing Education Conference* (Geelong, VIC, Australia) *(ACE '17)*. Association for Computing Machinery, New York, NY, USA, 26–35. https://doi.org/10.1145/3013499.3013500

[27] Harvey Mellar, Roumiana Peytcheva-Forsyth, Serpil Kocdar, Abdulkadir Karadeniz, and Blagovesna Yovkova. 2018. Addressing cheating in e-assessment using student authentication and authorship checking systems: teachers' perspectives. *International Journal for Educational Integrity* 14, 1 (Feb. 2018), 2. https://doi.org/10.1007/s40979-018-0025-x

[28] Timothy B. Michael and Melissa A. Williams. 2013. Student Equity: Discouraging Cheating in Online Courses. *Administrative Issues Journal: Education, Practice, and Research* 3, 2 (2013). https://eric.ed.gov/?id=EJ1057085

[29] Orna Muller. 2005. Pattern Oriented Instruction and the Enhancement of Analogical Reasoning. In *Proceedings of the First International Workshop on Computing Education Research* (Seattle, WA, USA) *(ICER '05)*. Association for Computing Machinery, New York, NY, USA, 57–67. https://doi.org/10.1145/1089786.1089792

[30] Nancy Pennington. 1987. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology* 19, 3 (1987), 295–341.

[31] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017), 24 pages. https://doi.org/10.1145/3077618

[32] Joacim Ramberg and Bitte Modin. 2019. School effectiveness and student cheating: Do students' grades and moral standards matter for this relationship? *Social Psychology of Education* 22, 3 (July 2019), 517–538. https://doi.org/10.1007/s11218-019-09486-6

[33] Anthony Robins, Janet Rountree, and Nathan Rountree. 2003. Learning and Teaching Programming: A Review and Discussion. *Computer Science Education* 13, 2 (2003), 137–172. https://doi.org/10.1076/csed.13.2.137.14200 arXiv:https://doi.org/10.1076/csed.13.2.137.14200

[34] Kshitij Sharma, Patrick Jermann, Marc-Antoine Nüssli, and Pierre Dillenbourg. 2012. Gaze Evidence for Different Activities in Program Understanding. In *PPIG*.

[35] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *ACM Trans. Comput. Educ.* 13, 4, Article 19 (Nov. 2013), 40 pages. https://doi.org/10.1145/2534973

[36] Jason M Stephens, Michael F Young, and Thomas Calabrese. 2007. Does moral judgment go offline when students are online? A comparative analysis of undergraduates' beliefs and behaviors related to conventional and digital cheating. *Ethics & Behavior* 17, 3 (2007), 233–254.

[37] Ethel Tshukudu and Quintin Cutts. 2020. Understanding Conceptual Transfer for Students Learning New Programming Languages. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 227–237. https://doi.org/10.1145/3372782.3406270

[38] Gary M. Velan, Philip Jones, H. Patrick McNeil, and Rakesh K. Kumar. 2008. Integrated online formative assessments in the biomedical sciences for medical students: benefits for learning. *BMC Medical Education* 8, 1 (Nov. 2008), 52. https://doi.org/10.1186/1472-6920-8-52

[39] Chelley Vician, Debra D Charlesworth, and Paul Charlesworth. 2006. Students' perspectives of the influence of web-enhanced coursework on incidences of cheating. *Journal of Chemical Education* 83, 9 (2006), 1368. https://doi.org/10.1021/ed083p1368

[40] George R Watson and James Sottile. 2010. Cheating in the digital age: Do students cheat more in online courses? *Online Journal of Distance Learning Administration* 13, 1 (2010).

[41] Matthew West, Geoffrey L. Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning. *ASEE Annual Conference and Exposition, Conference Proceedings* 122nd ASEE Annual Conference and Exposition: Making Value for Society, 122nd ASEE Annual Conference and Exposition: Making Value for... (2015). 2015 122nd ASEE Annual Conference and Exposition ; Conference date: 14-06-2015 Through 17-06-2015.

[42] Michele Weston, Kevin C. Haudek, Luanna Prevost, Mark Urban-Lurain, and John Merrill. 2015. Examining the Impact of Question Surface Features on Students' Answers to Constructed-Response Questions on Photosynthesis. *CBE Life Sciences Education* 14, 2 (June 2015). https://doi.org/10.1187/cbe.14-07-0110