Harnessing the Wisdom of the Classes

Classsourcing and Machine Learning for Assessment Instrument Generation

Sam Saarinen
Brown University
Providence, Rhode Island, USA

Shriram Krishnamurthi
Brown University
Providence, Rhode Island, USA

Kathi Fisler
Brown University
Providence, Rhode Island, USA

Preston Tunnell Wilson
Brown University
Providence, Rhode Island, USA

ABSTRACT

Generating questions to engage and measure students is often challenging and time-consuming. Furthermore, these questions do not always transfer well between student populations due to differences in background, course emphasis, or ambiguity in the questions or answers. We introduce a contributing student pedagogy activity facilitated by machine learning that can generate questions with associated answer-reasoning sets. We call this process Adaptive Tool-Driven Conception Generation. A tool implementing this process has been deployed, and it explicitly optimizes the process for questions that divide student opinion. In a study involving arrays in Java, this novel process: generates questions similar to expert-designed questions, produces novel questions that identify potential student misconceptions, and provides statistical estimates of the prevalence of misconceptions. This process allows the generation of quiz and discussion questions with less expert effort, facilitates a subprocess in the creation of concept inventories, and also raises the possibility of running reproduction studies relatively cheaply.

CCS CONCEPTS

• **Social and professional topics** → **Student assessment**; • **Computing methodologies** → Machine learning;

KEYWORDS

Contributing Student Pedagogy, Instrument Development, Learning Analytics

ACM Reference Format:
Sam Saarinen, Shriram Krishnamurthi, Kathi Fisler, and Preston Tunnell Wilson. 2019. Harnessing the Wisdom of the Classes: Classsourcing and Machine Learning for Assessment Instrument Generation. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287504

1 INTRODUCTION

Student conceptions and learning benefit from a diversity of assessment instruments. In an ideal world, educators would have access to a concept inventory (CI) for each topic they cover. A CI [8] is a *validated*, *robust*, and *interpretable* instrument that pinpoints student conceptions. It can examine the knowledge of students coming into a course [7], measure their change across the course [7], and help identify misunderstandings on the spot, e.g., in conjunction with clickers [13]. Therefore, there is value to having a large number of CIs across the whole spectrum of computer science topics.

Unfortunately, generating a CI or similar instrument is painstaking. Furthermore, a CI for a certain topic may not be very useful in a particular instructor's setting due to differences of covered material, prior preparation, choice of programming language, and numerous other factors. Finally, the number of actual CIs in computer science is fairly small [9] [14] and are relatively concentrated in introductory computing, failing to cover large parts of the field. Thus, CIs appear to be difficult-to-attain holy grails.

Absent such a rigorous instrument, an instructor might have to make do with a quiz of their design, which lacks any of the properties listed above. Due to the expert blind spot [11] the quiz may fail to cover some topics. It may also miss many mistakes that students have, may not necessarily clearly line up mistakes with misconceptions, and so on.

In this paper we attempt to find a happy middle between these extremes. We define the process of *Adaptive Tool-Driven Conception Generation* (ATCG), which uses a pair of techniques in conjunction:

crowdsourcing to obtain a large number of contributors with diverse views, but using students in a class; and,

machine learning to efficiently infer which contributions are most robust.

It is instructive to compare the ATCG process with a traditional CI process. Both are shown in Figure 1, with CI on the left and ATCG on the right. The CI steps are clearly much more heavy-weight, resulting in valuable instruments generated at great human (especially expert) expense. The ATCG steps are cheap and benefit from automation. They do not offer the same guarantees, but can be applied in many settings easily.

This naturally raises the question, what is the quality of the results from ATCG? We present a first set of results in this direction. First, we describe a tool, Quizius, that implements ATCG (section 3). We have used Quizius in three settings, two of which have nothing remotely resembling a computing CI: higher-order functional programming (which is used in many contexts, and even supported by the Snap! block programming language), and quantifier use in an upper-level course on mathematical logic in computer science. Due to space limitations we will focus on the third: generating an instrument for arrays in a Java CS2 course. We pick this because we can compare the result against an existing CI [9]. Perhaps somewhat surprisingly, the ATCG output fares quite positively (section 4), showing that this process has potential and is worth studying further.

Due to space requirements, we cannot include the full details here. A repository containing relevant documents is available at http://cs.brown.edu/research/plt/dl/quizius-2019/. Additionally, the Web-hosted Quizius tool is available (with a demo) at www.quizi.us, which other instructors are welcome to use to generate instruments for their courses.

2 RELATED WORK

Concept Inventories in Computer Science. Taylor et al. [14] provide a survey of the concept inventory efforts in computer science. They acknowledge that these are expensive to run due to the strong validation demands. In this paper we focus specifically on the instrument of Kaczmarczyk et al. (2010), based on extensive student interviews on topics validated by experts as important and difficult for beginning programming students, resulting in 16 questions, with most referencing snippets of Java code. We focus on the array portion of it, which is represented as four questions in Figure 3 (reproduced here with permission).

Contributing Student Pedagogy. ATCG can be viewed as a form of Contributing Student Pedagogy (CSP) [6]. This literature suggests that writing questions and answering other students' questions has a positive effect on course outcomes. ATCG has most of the attributes of a CSP, though simply generating an instrument provides only limited opportunity to "value the contribution of others", which for now has to be done through other channels. (ATCG also shares elements with *action research* [2], including a focus on action and reflection, and students as the main driver of scope.)

Quizius shares much with the CSP system PeerWise [3], but:

- When answering questions in our system, all answers are free-response so that the student answers will be unbiased.
- Rather than using student ratings of problem difficulty and usefulness, we empirically measure the informativeness of each question by using machine learning to examine the diversity of answers and optimize answer collection.
- After Quizius has run, an expert curates the most informative contributed questions for the purpose of creating an instrument, whereas no such engagement is necessary with PeerWise, where the activity is the goal in itself.

Bandit Processes and Machine Learning. To construct a quality instrument, we need robust questions. But measuring robustness accurately requires collecting responses from many students for that question. Additionally, out of consideration both for the students and the quality of data, we would like to not waste students' time on questions that are not likely to be interesting. One approach is to continually estimate the likelihood that a question is interesting, and prioritize data collection for questions that are already suspected to produce interesting responses. But we should not prematurely rule out a question because the first few responses were all the same (suggesting the question is uninteresting).

The problem of choosing an action (*a question*) out of many, where taking the action produces some reward (*we might get an interesting response*) and also gives us more information about the action (*we have a better estimate of how interesting the question is*), is a well-studied reinforcement learning problem called the Multi-Armed Bandit. Efficient solutions have been derived that are guaranteed to converge to selecting the best action as more data are acquired ([1], for example), which we use in Quizius.

3 THE QUIZIUS TOOL

To facilitate ATCG, we created the Web-based Quizius tool (Figures 4 and 5). It allows students to join classes, participate in quiz activities, write questions, answer questions with rationales, and review question and answer distributions after the quiz ends. Quizius also allows instructors to contribute their own questions, to review the distribution of student responses, and to automatically grade students based on participation and late-day use.

In addition to streamlining the data collection process, the tool implements the critical machine learning step to filter out informative questions from the large pool, spending as few student answers as possible on questions that are not very informative. Quizius models picking questions for students to answer as a multi-armed bandit problem. We define the reward (the objective we are seeking to maximize) for the bandit problem as the likelihood that the next response duplicates a non-majority answer. We believe this is a good proxy for questions that elicit diverse student mental models (section 4.5). Note that the objective of the ATCG method is to identify misconceptions for the purpose of evaluating pedagogical efficacy or tailoring instruction, not to place students on a scale of ability, as in item response theory [10]. This necessitates a different measure of question reliability than is typically used for so-called high-stakes assessments.

Ordinarily, upper-confidence-bound algorithms (what we are using to solve the multi-armed bandit problem) converge to selecting the single best option. Since we are trying to find not the best question but rather a small set of questions, we select from among the top $\sqrt{n}$ questions, where n is the number of students who have used the system so far.

One difficulty is that students may write the same answer in different formats (e.g., "1000" versus "1,000"). We found it annoying to manually cluster these. Therefore, after students answer a question, they are shown previous representative answers and asked whether theirs is the same as any of these. In effect, we ask the students to cluster their answers themselves. This is easy for them, and in practice catches most duplicates. Of course, on seeing the previous answers, students may change their response (e.g., if they feel they were mistaken). Thus, Quizius also lets them explicitly mark that they are changing their answer. This switch provides a very useful signal about student understanding that we have not yet exploited.

When the student process is done, the expert is given: a collection of questions; for each question, a collection of answers; for each answer, a collection of rationales for that answer. The expert then

(1) experts set the scope of the assessment (using a Delphi Process to choose important and difficult topics)
(2) students articulate mental models and experts identify misconceptions (through interviews to find interpretations for student responses)
(3) experts develop questions (to distinguish mental models)
(4) experts validate the assessment (using statistical and qualitative analysis across several trial administrations to iterate towards robust and validated questions)

Figure 1: A traditional process for creating new CIs. Steps are taken from [5] and are based on recent practice in several disciplines [4].

(1) experts write a prompt setting the scope for contributions
(2) students contribute questions
(3) students contribute answers and rationales (by answering other students' questions and providing a brief justification)
(4) a machine learning algorithm prioritizes responses to informative questions (in order to find robust questions)
(5) experts select questions for relevance (thus validating the questions)
(6) experts identify misconceptions (by reviewing student rationales for each answer and interpreting the responses)

Figure 2: The novel ATCG process.

selects the questions that appear most interesting to assemble into an instrument. Examining the rationales gives insight into the mental models responsible for the student responses. In practice, we find that most answers arise for the same reason, but a careful reading finds a few other "outlier" reasons. If ATCG is being used to help build a CI, such questions would have to be modified so that different (mis)conceptions do not result in the same answer.

The ATCG protocol has a bootstrapping problem: What questions do the first students answer? Quizius permits multiple options: let them simply finish early, or use pre-populated questions. The latter may be various sources: instructor-provided, from an existing CI, or even from previous iterations of using Quizius (e.g., if the process is re-run across years—which is what we did in our study in section 4).

4 A COMPARATIVE STUDY

We now compare an ATCG-generated instrument for CS2 Java arrays (the *generated* instrument), against Kaczmarczyk et al.'s CI (the *expert* instrument). G1-G6 in Figure 6 show the student-generated questions. At a high level, we find:

- The generated instrument contains most of the same topics and distractors as the expert instrument.
- The generated instrument misses some of the content of the expert instrument; we conjecture why this is so.
- The generated instrument contains array use topics—and distractors for them, along with explanations of misconceptions—*not* included in the expert instrument.

While performing this comparison, we also identify possible points of confusion in the expert instrument.

4.1 The Study Population and Iteration

We generated this instrument from two iterations of a CS2 class at a mid-size competitive private research US university. Although the course covered algorithms and data structures typical of CS2, it was also a first course using Java and a first course with object-orientation (the students had previously studied functional programming). Each time, the study was run midway through the course, after students had completed a lab assignment on each of objects and arrays. Just over 120 students participated in each iteration of the study, and we note (based on results from an entry survey) that less than half of these students had high school programming experience in Java.

The study was structured as an online homework assignment, and students were given a week to complete it. The first time the study was run, students were required to write one question (consisting of a short program in Java whose printed output demonstrated something they considered interesting about arrays; Figure 4 shows an example prompt), then to answer 15 questions written by other students in free-response style (to predict the printed output of the written programs without running them). They were also asked to answer the 4 array-related questions from the expert instrument. Figure 3 shows their counts for each answer in brackets; note that many answers were chosen by no students.

In the second iteration, we seeded the system with contributions from the previous year. We note that we observed qualitatively similar distributions of mental models between the two years. To reduce student effort, we made question submission optional (23 out of 128 students submitted one voluntarily anyway). Because of improvements to the question allocation algorithm and a tighter semester schedule, we only asked them to answer 10 questions. Section 4.6 gives more results on engagement and student time. *All* questions and responses in Figure 6 are student-generated; the responses are all from the second iteration of the study.

4.2 Similar Questions — Indexing and Length

To begin, we note that both the generated and the expert questions contain questions involving indexing (is Java 1-indexed or 0-indexed) and array length (E1 and E2 in Figure 3 (expert instrument) and G1, G2, G4, and G7 in Figure 6 (generated instrument)). We discuss 1-indexed misconceptions quantitatively in Section 4.6.

4.3 Missing Questions — Memory and Type

Although there is a diversity of questions in the generated instrument, we note that there are no questions about memory allocation or type of an array object similar to E3 and E4 (Figure 3). We conjecture that this was precluded by the prompt used in this study, which required students to write a short Java program whose output would be printed. We consider it likely that most CS2 students do not have access to the reflection capabilities necessary to phrase such questions as Java programs. Perhaps if given a different prompt, students might write questions regarding type or memory allocation, but it is also possible that a less-structured activity would be harder for students. Thus, an open-ended CI interview offers possibilities that a closed, computer-based process may make more difficult.

E1. What should the values of x and y be, in order to fill all elements of the following array with values of -1? [Distractors are based on thinking that arrays start at index 1 and/or a for loop stepping through an array will need to go until < the array's length - 1]

```
int[] myArray = new int[10];
int i;
int x;
int y;
for (i = x; i < y; i++) {
    myArray[i] = -1;
}
```

(a) x = 1, y = the length of myArray [1x]
(b) **x = 0, y = the length of myArray** [109x]
(c) x = 1, y = the length of myArray - 1 [0x]
(d) x = 0, y = the length of myArray - 1 [22x]

E2. How many elements are in myArray? [Distractors are based on thinking that an array's length is off by 1 or equal to the number of letters in the array's name]

(a) 7 [0x]
(b) 9 [1x]
(c) **10** [124x]
(d) 11 [0x]

E3. Which of the following answers most accurately describes the parts of the declaration of myArray? [Distractors are based on thinking the type of an array is just an array or a general object and/or declaring the length of an array instead sets the first element or all elements in the array to the value of the length] [remove answer e for non-Java languages]

```
int[] myArray = new int[5];
```

(a) **type is an integer array, length is 5** [124x]
(b) type is an array, each element has a value of 5 [0x]
(c) type is an integer array, each element has a value of 5 [0x]
(d) type is an integer array, the value of the first element is 5 [0x]
(e) type is an object array, the constructor is passed a value of 5 [2x]

E4. Which of the following answers most accurately describes the memory allocation of myArray in E3? [Distractors are based on thinking memory is not allocated for the elements in the array or only the 1st element or plus an extra element]

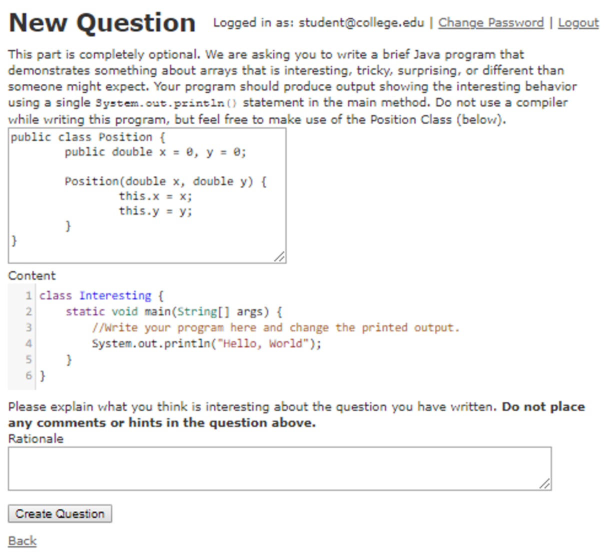
```
int[] myArray = new int[5];
```

(a) memory is allocated for 6 spaces [7x]
(b) **memory is allocated for 5 spaces** [120x]
(c) no memory is allocated [0x]
(d) memory is only allocated for the first element [0x]

Figure 3: Four questions from the expert instrument. Out of the 16 total questions, these related somehow to arrays. Questions, answers, and interpretations (mental models) are from Kaczmarczyk and collaborators (private correspondence, extending [9]). Answers are followed by the number of *our* respondents who selected that answer. Note that not all respondents answered all questions.

4.4 Novel Questions — Assignment, Aliasing, and Equality

Finally, we note that there are a number of concepts tested by the generated instrument that are not present in the expert instrument. For example, G1 (Figure 6) involves nested arrays, G3 tests aliasing after assignment of one array to another, G4 involves mutation of an array passed to a method, and G5 and G6 test array equality. These are all concepts that are not addressed by the expert instrument. The authors of this paper were surprised at some of the things students thought to write questions about, and we conjecture that student-authored questions may help to get beyond expert blind spots. In particular, G2 (which reveals a mutation and aliasing misconception—this is discussed in Section 4.6) is not a question that occurred to us.



New Question

Logged in as: student@college.edu | Change Password | Logout

This part is completely optional. We are asking you to write a brief Java program that demonstrates something about arrays that is interesting, tricky, surprising, or different than someone might expect. Your program should produce output showing the interesting behavior using a single `System.out.println()` statement in the main method. Do not use a compiler while writing this program, but feel free to make use of the Position Class (below).

```
public class Position {
        public double x = 0, y = 0;

        Position(double x, double y) {
                this.x = x;
                this.y = y;
        }
}
```

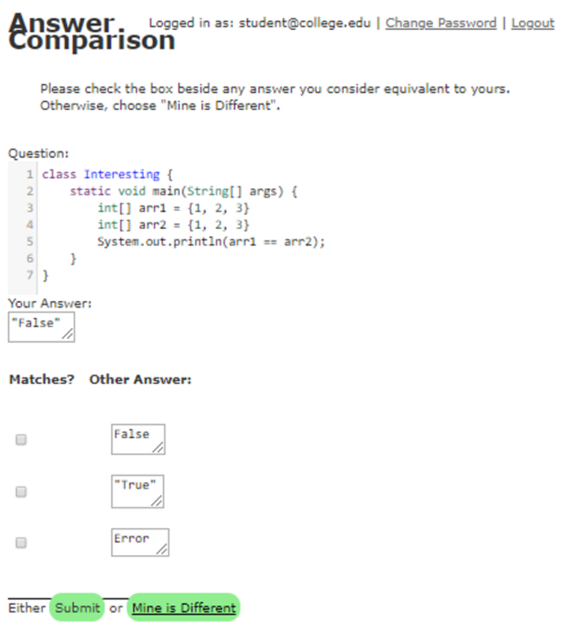
Content

```
class Interesting {
    static void main(String[] args) {
        //Write your program here and change the printed output.
        System.out.println("Hello, World");
    }
}
```

Please explain what you think is interesting about the question you have written. **Do not place any comments or hints in the question above.**

Rationale

Figure 4: The Question Writing Interface for students.



Answer Comparison

Logged in as: student@college.edu | Change Password | Logout

Please check the box beside any answer you consider equivalent to yours. Otherwise, choose "Mine is Different".

Question:

```
class Interesting {
    static void main(String[] args) {
        int[] arr1 = {1, 2, 3}
        int[] arr2 = {1, 2, 3}
        System.out.println(arr1 == arr2);
    }
}
```

Your Answer: "False"

Matches?	Other Answer:
☐	False
☐	"True"
☐	Error

Either Submit or Mine is Different

Figure 5: The Answer Grouping Interface for students. After giving a free response answer and a brief rationale, students are taken to this screen where they either combine their answer with an existing group, or indicate that their answer is different from the others.

4.5 Novel Mental Models

This study allowed us to identify new mental models related to multiple indexing, equality comparison, and initialization of arrays. Additionally, by analyzing over a hundred student responses to expert instrument question E4 (Figure 3), we identified a new mental model relative to those previously identified: one student believed that 6 spaces were allocated for a `new int [5]`, because they were aware that Java caches the length of an array as an integer, and

The provided Position class could be used freely:

```
public class Position {
    public double x = 0, y = 0;
    Position(double x, double y) {
        this.x = x;
        this.y = y;
    }
}
```

G1.

```
public class Interesting {
    public static void main(String[] args) {
        int[][][][][] arr = {{{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}},
                              {{{9, 10}, {11, 12}}, {{13, 14}, {15, 16}}}},
                            {{{{17, 18}, {19, 20}}, {{21, 22}, {23, 24}}},
                              {{{25, 26}, {27, 28}}, {{29, 30}, {31, 32}}}}};
        System.out.println(arr[1][1][1][1][1]);
    }

}
```

(a) 32 [24x]
(b) 1 [9x]
(c) 16 [4x]
(d) {3,4} {11,12}, {19,20}, {27, 28} [2x]
(e) {{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}
{{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}
{{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}
{{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}
{{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}
[1x]
(f) illegible answer in console [3x]
(g) other [10x, including 30, 28, 12, 14, and 31]

G2.

```
public class Interesting {
    public static void main(String[] args) {
        int x = 3;
        int[] test = {x,1,1,1};
        x = 1;
        System.out.println(test[0]);
    }
}
```

(a) 3 [42x]
(b) 1 [8x]

G3.

```
public class Interesting {
    public static String posString(Position p) {
        return "(" + p.x + ", " + p.y + ")";
    }
    public static void main(String[] args) {
        Position[] whatIsJava = new Position[5];
        Position[] otherArray = whatIsJava;
        otherArray[4] = new Position(5.5, 2.2);
        System.out.println(posString(whatIsJava[4]));
    }
}
```

(a) (5.5, 2.2) [23x]
(b) null [14x]
(c) error [10x]
(d) (0, 0) [7x]
(e) other [2x]

G4.

```
public class Interesting {
    public static void reverse(int[] input) {
        int lastPlace = input.length - 1;
        int middlePlace = input.length / 2;
        for (int x = 0; x <= middlePlace; x++) {
            int temp = input[x];
            input[x] = input[lastPlace - x];
            input[lastPlace - x] = temp;
        }
    }
    public static void main(String[] args) {
        int[] arr = { 2, 16, 8, 42, 89 };
        reverse(arr);
        System.out.println(arr);
    }

}
```

(a) [89, 42, 8, 16, 2] [24x]
(b) Some memory address [19x]
(c) {89, 42, 8, 16, 2} [7x]
(d) other [4x]

G5.

```
public class Interesting {
    public static void main(String[] args) {
        int[] first = {1, 2, 3, 4};
        int[] second = {1, 2, 3, 4};
        System.out.println(first==second);
    }
}
```

(a) false [31x]
(b) true [11x]
(c) error [10x]

G6.

```
public class ArraysExample {
  public static void main(String[] args) {

    Position[] array1 = new Position[2];
    Position[] array2 = new Position[2];

    Position posn1 = new Position(2.5, 3.5);
    Position posn2 = new Position(2.5, 3.5);

    array1[0] = posn1;
    array1[1] = posn2;

    array2[0] = posn2;
    array2[1] = posn1;

    System.out.println(array1.equals(array2));
  }
}
```

(a) False [39x]
(b) true [10x]
(c) Error [2x]
(d) other [3x]

G7. [Note that Quizius ranked this question as having low information (limited diversity in answers). We include it here for comparison with answers to G1.]

```
public class Interesting {
    public static void main(String[] args) {
      try {
        Integer[] intArray = new Integer[] {1, 2, 3, 4, 5};
        System.out.println(intArray[5]);
         // print out the 5th element of the array
      } catch (ArrayIndexOutOfBoundsException e) {
     System.out.println("intArray does not contain an element at index 5");
      }
    }
}
```

(a) “intArray does not contain an element at index 5” [53x]
(b) Error [2x]

Figure 6: A selection from the most informative questions generated using ATCG. Each question is a program, whose output students were asked to predict. Each answer is followed by the number of respondents who agreed with it. Some answers are combined as “other” for brevity.

reasoned that that was allocated in addition to the 5 spaces for the 5 elements of the array. Although it is a rare mental model in our population, this suggests a refinement to that question. This mental model may have been lost in the much smaller sample size for which it is feasible to conduct extensive interviews.

4.6 Quantitative Analysis

Space precludes a detailed analysis of the questions in the two instruments, and student responses to them (and the mental models they reveal). However, we enumerate some salient points:

- The expert instrument (Figure 3) is able to detect up to 12 misconceptions (either or both of 2 for E1, 3 for E2, 4 for E3, and 3 for E4) related to arrays. Our population exhibited a ceiling effect on the expert instrument, however, as only 5 of those misconceptions appeared in our population, and the majority of students got all the CI questions correct. (Figure 3 contains detailed incidence counts.) The generated instrument (considering only programs 1-6, as presented in Figure 6) detected 13 misconceptions in our student population.
- The top-ranking generated questions are ones that produce a spread of student responses. Although this spread is not always due to a difference in mental models (G4 in Figure 6), other questions capture misconceptions in a nuanced way. In particular, on G1 on the generated instrument, 9 out of the 53 students exhibited a 1-indexing misconception (answer b). Interestingly, this 1-indexing misconception did not occur on a student-generated question involving a 1-dimensional array (G7), and on E1 from the expert instrument (Figure 3), only one student chose an answer consistent with a 1-indexing misconception. This is likely an instance of the well-known phenomenon of fragile student knowledge [12].
- For our student population, the generated questions detected misconceptions more frequently than the CI questions. There are many possible reasons for this, which we do not have space to discuss further. We do find interesting mental models in answers that claim a program is erroneous. We speculate this may not happen as frequently if students believe the programs were written by an expert.
- We measured the amount of time each student spent on the question authoring page and on the answer writing page, which is a reasonable proxy for student time, although some large outliers (one answer took 8.8 hours) suggest that students may have left the page open while doing other things. The median time to write a question was just over 11 minutes, and the median time to answer a question was 1 minute and 17 seconds. Furthermore, just over 30% of our students voluntarily answered more questions than required to complete the homework assignment—some by a significant margin, which suggests some of them may even have found this process educational or fun.

  Across all students, about 90 hours of student time were used in total. About 8 expert hours over the course of a week were used to design and refine the protocols for the study, and about 15 expert hours over the course of two weeks were spent on selecting the relevant questions and interpreting student rationales. In total wall clock time, the second iteration of this study took 4 weeks to run end-to-end, which is substantially faster than any CI-creation process.

5 DISCUSSION

It is not clear whether it is better to ask students to write questions before, during, or after answering the questions of other students. Having them write questions before answering any occasionally results in questions that are nearly isomorphic, although the written questions are unbiased by the work of other students. Having them write questions during or after allows students to benefit from the examples they have seen, and deliberately write different questions, but it may artificially constrain their responses based on what they have seen; more studies will need to be done to evaluate the relative strength of these effects quantitatively.

We found that some students submitted questions unrelated to arrays in Java, although they did produce a broad spread of answers. (In particular, a question about object equality comparison left even the authors searching carefully through the Java language specification to understand the correct answer.) Although the tool could be altered to allow students to flag off-topic questions, the instructor may wish to keep those questions that reveal a spread of mental models, even if they are off-topic.

Though we did not encounter instances of abusive use, it is also important for users to be able to flag offensive content (as they can in Quizius). Malicious users might write an abusive answer that others would have to read; more subtly, they might submit a program whose output is offensive, which everyone else would then be forced to type. Working in a graded course, rather than on the open Internet, probably helped us avoid such incidents.

6 CONCLUSION

This paper presents ATCG as an intriguing mid-point, and potential sweet spot, between the heavy-weight CI process and entirely instructor-generated evaluation instruments. Augmenting instructor work with that of students can reveal a rich set of models, working around expert blind-spots. In a comparative study, we find that *purely* student-generated questions and answers, with relatively little effort (and with some of the auxiliary benefits of CSP), can compare very favorably to ones generated with considerable effort by a team of experts. Our tool, Quizius, helps run the ATCG process, presenting a Web interface and embodying a machine learning algorithm. We have already begun to apply ATCG in two other computing contexts, and we feel this process should be applied to all of computer science (indeed, Quizius is not even computing-specific), especially in areas where it may take a very long time and considerable expense to cover with a CI.

ACKNOWLEDGMENTS

We are grateful to Lisa Kaczmarczyk for contributing questions from an expert instrument. We thank Leo Porter and Daniel Zingaro, as well as Mike Clancy and Cynthia Taylor, for their elucidating conversations on Concept Inventories and for pointing us to action research. This work was partially supported by the National Science Foundation.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* (2002).

[2] Mary Brydon-Miller, Davydd Greenwood, and Patricia Maguire. 2003. Why action research? *Action Research* (2003).

[3] Paul Denny, John Hamer, Andrew Luxton-Reilly, and Helen Purchase. 2008. PeerWise: students sharing their multiple choice questions. In *ICER*. ACM.

[4] DL Evans, Gary L Gray, Stephen Krause, Jay Martin, Clark Midkiff, Branislav M Notaros, Michael Pavelich, David Rancour, Teri Reed-Rhoads, Paul Steif, et al. 2003. Progress on concept inventory assessment tools. In *Frontiers in Education*. IEEE.

[5] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C Loui, and Craig Zilles. 2008. Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bulletin* (2008).

[6] John Hamer, Quintin Cutts, Jana Jackova, Andrew Luxton-Reilly, Robert McCartney, Helen Purchase, Charles Riedesel, Mara Saeli, Kate Sanders, and Judithe Sheard. 2008. Contributing Student Pedagogy. *SIGCSE Bull.* (2008).

[7] Charles Henderson. 2002. Common concerns about the force concept inventory. *The Physics Teacher* (2002).

[8] David Hestenes, Malcolm Wells, Gregg Swackhamer, et al. 1992. Force concept inventory. *The Physics Teacher* (1992).

[9] Lisa C Kaczmarczyk, Elizabeth R Petrick, J Philip East, and Geoffrey L Herman. 2010. Identifying student misconceptions of programming. In *SIGCSE*. ACM.

[10] Frederic M Lord. 2012. *Applications of item response theory to practical testing problems*. Routledge.

[11] Mitchell J Nathan, Kenneth R Koedinger, Martha W Alibali, et al. 2001. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. *ICCS*.

[12] DN Perkins and Fay Martin. 1986. Fragile knowledge and neglected strategies in novice programmers. In *Empirical Studies of Programmers*.

[13] Beth Simon, Michael Kohanfars, Jeff Lee, Karen Tamayo, and Quintin Cutts. 2010. Experience report: peer instruction in introductory computing. In *SIGCSE*. ACM.

[14] C. Taylor, D. Zingaro, L. Porter, K.C. Webb, C.B. Lee, and M. Clancy. 2014. Computer science concept inventories: past and future. *Computer Science Education* (2014).