The Practical Details of Building a CS Concept Inventory

Cynthia Taylor
Oberlin College

Michael Clancy
University of California, Berkeley

Kevin C. Webb
Swarthmore College

Daniel Zingaro
University of Toronto Mississauga

Cynthia Lee
Stanford University

Leo Porter
University of California, San Diego

ABSTRACT

Concept inventories (CIs) allow researchers and practitioners to measure student conceptual learning within a course or topic area. While they have enabled meaningful pedagogical change in other disciplines, there are relatively few CIs in computer science. In this paper, we report on our experiences as recent developers of a CI for basic data structures. We discuss each step along the route to a CI and offer tips based on what we have learned. We encourage others to create CIs, and we hope that this paper will serve as a practical guide through the process.

CCS CONCEPTS

• **Social and professional topics → Computing Education**.

KEYWORDS

concept inventory, assessment, data structures

ACM Reference Format:
Cynthia Taylor, Michael Clancy, Kevin C. Webb, Daniel Zingaro, Cynthia Lee, and Leo Porter. 2020. The Practical Details of Building a CS Concept Inventory. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366903

1 INTRODUCTION

As computer science (CS) matures as a discipline and technology becomes increasingly pervasive in society, the expectations for CS graduates have never been higher. As CS instructors, we make every effort to maximize student learning — but how can we reliably measure the effectiveness of our learning interventions?

A *concept inventory* (CI) is an assessment designed to measure student conceptual understanding of a course. Unlike a traditional metric of student learning, such as an exam or course grade, a CI is not designed to assess individual students. Rather, a CI enables comparisons of student learning across instructors, institutions, curricula, and instructional techniques [1]. CIs empower education researchers to reliably measure the effects of interventions on student learning, and they allow practitioners to identify areas in which their students are struggling.

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366903

In other disciplines such as physics, CIs have inspired widespread change in pedagogy and facilitated large-scale studies with thousands of students [7, 13]. However, within computer science, only two validated CIs are widely available [15, 21]. We recently developed a third: a concept inventory for basic data structures (a subset of CS 2) [24][1]. This paper describes our work building that instrument. At each step we explore the CI design space in the context of the lessons we learned. We hope that our experiences will concretize the CI development process and spur future CI development in other areas of CS.

2 BACKGROUND AND RELATED WORK

A *conception* describes a belief, theory, or description previously developed to explain some behavior observed in the world. When these beliefs conflict with accepted scientific theories, they become *misconceptions*. One way to study student conceptions is to carefully devise a question that, answered correctly, provides a high degree of confidence in the student's mastery. Conversely, answering incorrectly may indicate misconceptions, particularly if a student demonstrates a consistent pattern of incorrectness across a collection of related questions. Such a collection of questions and their corresponding answers is referred to as a concept inventory (CI).

Concept inventories emerged from the physics education community, where the Force Concept Inventory (FCI) was created in response to the observation that students could successfully solve mathematical physics problems without adjusting their basic (mis)conceptions of how the world worked [16]. The FCI sparked widespread change in physics education and inspired the development of Peer Instruction [7]. The success of the FCI motivated the development of CIs in many other STEM disciplines, including chemistry [19], engineering [31], genetics [27], statistics [28], calculus [10], geoscience [20], oceanography [3], nursing [25], astronomy [4, 26, 35], and biology [9].

In contrast, few CIs have been developed in computer science thus far [29]. Tew and Guzdial developed the FCS1, a CI for introductory programming [30]. However, the FCS1 is not generally available, which led to the development of the Second CS1 Assessment (SCS1), an isomorphic version of the FCS1 [21]. Additionally, Herman et al. [15] developed a CI for digital logic.

While other efforts explore preliminary work towards CI development for more advanced areas of CS, including algorithms [8, 11], recursion [14], advanced data structures [17], computer architecture [22], and operating systems [32], these efforts remain incomplete or lack rigorous validation.

Overall, the relative paucity of validated CIs for computer science, especially in advanced topics, represents both an area of great potential and a considerable need for the CS education community.

[1] Available at https://groups.google.com/forum/#!forum/cs2-bdsi-concept-inventory

3 CI DEVELOPMENT OVERVIEW

Typically, a team looking to build a CI has a particular topic or course in mind for which they wish to measure student learning. Our team focused on data structures content often found in "CS 2" courses. CS 2 covers core concepts that CS students use throughout their education (and their careers), typically including data structures, complexity analysis, object-oriented programming, and recursion. CS 2 is taught in nearly every CS program at least once a year. It's typically required for every CS major and minor, and it often serves many other non-CS students. Moreover, while many courses in CS continue to fluctuate as technology changes and new programming languages and platforms are developed, CS 2's core concepts have remained relatively static, ensuring applicability of a CI well into the future.

Before starting CI development, we worked as a team to determine the likely users of the CI and their needs. We decided to target three likely audiences: computing education researchers studying the impact of pedagogical, curricular, or tool changes; curriculum designers evaluating end-of-course knowledge relative to later course prerequisites; and CS 2 instructors measuring student learning in their courses. We also worked to determine measures of validity for the CI, specifically focusing on the claims we hoped to make to convince each audience that the CI would work for their purposes. The ultimate claims of validity and evidence gathered in support of those claims for this CI are provided in prior work [24].

Having established our goals, we looked to work by Adams and Wieman to guide us in our CI development [1]. Their work distills the development of numerous earlier CIs in a variety of science fields and lays out a systematic procedure for applying their design principles in other contexts. In particular, it describes six steps for building a CI. We collapsed two steps of their model (their step 2 and 3 into our step 2) by developing open-ended questions based on instructor knowledge of how students struggle with topics. We then interviewed students using these initial open-ended questions. This change ultimately yielded the following development process:

(1) Establish topics that are important to teachers.
(2) Develop a set of open-ended questions on these topics.
(3) Interview students and offer open-ended practice exams to discover misconceptions in order to convert questions into a forced-choice or select-all format.
(4) Carry out validation interviews at a variety of institutions.
(5) Statistically validate the CI.

Subsequent sections describe each of these steps in detail. We also discuss administration for question development in Section 7 and statistical validation in Section 9, and we offer recommendations for administering the complete CI in Section 10.

4 ASSEMBLING A TEAM

Our primary project team included five people designated as "principal investigators" (PIs) for an NSF grant, two consultants (one is the grant's external evaluator), and one graduate student. Three of the PIs, both of the consultants, and the graduate student work at large research-oriented public universities; the other two PIs come from highly selective private colleges with small student populations. Team background thus skewed toward research institutions.

Our external evaluator on the grant also served in a significant advisory role. We solicited her support as she previously designed the Colorado Upper-Division Electrostatics diagnostic [6] and has a strong background in assessment design and the learning sciences. Where appropriate, she brought in outside advisors to help us with specific project goals (e.g., to aid in statistical validation). Overall, this type of guidance proved invaluable as she was able to help us through difficult questions (e.g., how much of a course needs to be covered by a CI, how can the team tell when the questions are polished enough to move to validation, how do we help foster future adoption of the instrument, etc.). We strongly recommend that any team without direct experience in CI construction employ someone with this kind of expertise to help ensure project success.

Because a CI serves as a standard assessment instrument for a course, it must be representative of a diverse set of concerns regarding the course material and presentation. To go beyond what the PIs represent, we assembled a separate "expert panel" of eight CS instructors with a variety of backgrounds and skills, drawn from our professional connections. We tried to select experts with a variety of institutional demographics (small vs. large enrollments, public vs. private, community college vs. four-year teaching institution vs. four-year research institution) and pedagogy-related characteristics (experience teaching CS 2, experience teaching follow-on courses, topics covered and activities used in CS 2). Although we recruited panelists only from North America, developers may want to include more global representation if the course is consistent world-wide.

We offered each of our expert panelists an honorarium for their participation in the project. As we will discuss throughout the paper, these experts helped us determine the core concepts for the CI, provided guidance as to which learning goals are most critical, and contributed feedback regarding the suitability of our questions for testing expert-level thinking.

4.1 Lessons Learned: Expert Panel

We learned two important lessons working with the expert panel. First, we strongly benefited from selecting panelists who regularly attend the same annual conference as the PIs (in our case, the SIGCSE symposium). The conference represented a de facto meeting venue where everyone was prepared to discuss CI development topics. Second, it's critically important to respect the expert panelists' time. They should only be approached for activities that the primary project team can't perform on their own (e.g., to collect a broader perspective on a topic). We characterize the requests we made to our expert panel at each step in the sections that follow.

While we are indebted to our expert panel for their essential contributions, we can't claim to provide a specific formula for selecting panelists. We recognize many unresolved and nuanced considerations when choosing panelists. For example, how should PIs factor in a prospective panelist's likelihood of long-term collaboration, and should they be chosen based on past experiences working with them? Is it better for the PIs to optimize a panel for diversity, topic expertise and experience, or prior assessment authorship?

5 STEP 1: ESTABLISHING TOPICS

In preparation for the project, we spent several months collecting and reviewing course syllabi to discern which topics are commonly

taught in CS 2. Our analysis ultimately found that variations of CS 2 typically teach content in six topics: sorting, recursion, basic data structures, advanced data structures, object-oriented programming, and reasoning about code. Once the project started in earnest, we did a more careful analysis based on materials collected from PIs and our expert panel, including course descriptions, syllabi, exams, quizzes, solutions, and grading standards. Information about prerequisite and postrequisite courses was also useful. We used the subtopics covered in these documents to perform a more fine-grained evaluation of topics taught in CS 2.

Reading through the provided materials, we created a list of 49 subtopics covered in CS 2. To ensure our high-level groupings were accurate, we then sought to assign each of these 49 subtopics to one of our six topics. We found that each subtopic fell cleanly into one topic. To determine how important each of these topics was, we then asked our expert panel the following survey questions:

- How critical are the following topics for students' success in your CS 2 class?
- How important is it for students to know the content of the following topics as prerequisites for courses that follow your CS 2 class?
- How difficult is it for students to learn the following topics in your CS 2 class?

After our discussions with the panel, it became clear that while all CS 2 courses covered some subset of these topic groups, no CS 2 course covered all of them. In fact, there seemed to be two flavors of CS 2: one with a heavy emphasis on data structures, the other on object-oriented programming. Additionally, the entire range of CS 2 topics was far too broad for a single CI. This was a moment where our external evaluator was particularly helpful. She advised us that CI designers often think they need full course coverage when, in practice, the goals of a CI can be met by instead focusing on a portion (e.g. 60%) of course content.

We were encouraged to find that all of the courses we looked at covered the topics we classified as basic data structures, that basic data structures were often a significant portion of the course, and that the expert panel felt basic data structures were important for student success in CS 2 and follow-on courses. As a result, we narrowed our focus to a CI just for the topics within that group (10 of our original 49 subtopics; see [23]).

We note the existence of an established alternative for isolating course topics, specifically the Delphi process [12]. Delphi involves experts iteratively gathering and merging opinions. We elected to use our process instead because we felt confident that we could gain consensus among our team of experts and that each expert would make their voice heard without the need of a formalized process for eliciting contributions.

5.1 Learning Goals

Next, we converted topics to learning goals, statements of the form "after studying this topic, students will be able to ...". Based on expert-panel responses and short-answer exercises drawn from old exams, we devised a list of eleven learning goals. Panel members rated the difficulty and importance of each goal in the list.

We met with the expert panel in person at SIGCSE 2016 to discuss the eleven goals. Discussion yielded a number of changes, and we consequently consolidated our goals to a list of six. We were advised by our external evaluator that 6 goals was appropriate given the focus and depth of our assessment; many other CI projects have between 4 and 10 course-level learning goals that they seek to address [2, 4, 5, 16, 27, 31]. More details about our topics and learning goals are available in prior work [23].

5.2 Lessons Learned: Finding Topics and Goals

Many of our original goals were of a form similar to "Implement insertion into, location in, and deletion from a binary search tree." This style led to many similar, repetitive goals, where the only difference between goals was the data structure used. We realized that our misstep in designing the learning goals was coupling general data structures concepts to specific data structures. To solve this problem, we decoupled the learning goals themselves from any specific data structures, which allowed us to consolidate learning goals for different data structures into a single goal. For example, one such goal is now "Design and modify data structures capable of insertion, selection, search, and related operations." Ultimately, we produced a set of these broad learning goals together with a separate list of the particular data structure interfaces and implementations covered in the CI.

There was some contention among our experts about whether students should be able to select the appropriate interface, select the appropriate implementation, or implement the data structures themselves. Some instructors argued strongly that their students only needed to be able to choose an appropriate data structure to use in a program, while others focused entirely on data structure implementation. The group eventually concluded that students should be able to do all three to some degree, and these became our first three learning goals.

6 STEP 2: OPEN-ENDED QUESTIONS

During the prior step, we collected a number of short-answer exercises from old exams, both from our team and the expert panel. Collectively, we also have a substantial body of expertise in teaching CS 2; one of us had taught it more than 20 times (in traditional, self-paced, and lab-centric formats), producing a considerable collection of course material. We also used two exercises created by Karpierz and Wolfman [17]. We selected the subset of exercises appropriate for basic data structures and interviewed with those.

To prepare us for the interview process, our external evaluator ran a two-day workshop on conducting cognitive interviews [34] for assessment design. In this workshop, she discussed how to conduct interviews and brought in students from her campus to help us practice interview techniques. Three PIs and one consultant participated.

The logistics of interviewing students was more onerous than anticipated. We began by recruiting interviewees via announcements in the CS 2 courses at our universities. We interviewed fifty students across three institutions for steps 2 and 3. Students were offered an Amazon gift card for participating. While we tried to interview a diverse set of students, self-selection may have biased us towards students who were more confident in their abilities.

As we needed students to have been exposed to all of the material covered in the CI, we faced a relatively short window of approximately the last two weeks of the term in which to do interviews. This meant that in some cases, we would make changes based on student interviews only to then have to wait until the following term to do more interviews to get feedback on these changes.

Scheduling interviews proved challenging, as 10–20% of students who initially expressed interest failed to show up for their appointments. Asking the students to email us to confirm their interest after an initial signup, as well as emailing them a reminder the day before the interview, helped to increase participation.

Each 1-hour interview included reviewing the Human Subjects consent form, an ice-breaker discussion about courses to help make the student comfortable, talking through the nature of think-aloud interviews, and then having the student work through our draft questions. At the beginning, we warned participants that we would intentionally not be engaging them with verbal or non-verbal feedback as they worked through the problems. Not providing feedback to students was difficult for us because, as instructors, we found ourselves instinctively wanting to encourage students as they thought through the problem correctly or wanting to probe incorrect thinking. However, it was important not to provide this kind of feedback if we wanted to ascertain student thinking.

As students worked through questions, we did not intervene (except to ask them to think aloud if they were quiet). But when they finished a question, we often asked them what they were thinking about while solving the question. These dialogues were a particularly rich source of information for the team when students encountered unexpected difficulties (e.g., attempting to perform binary search on a doubly-linked list). These new difficulties informed our creation of new distractors for that question and sometimes led us to develop an altogether new question targeting a difficulty. We also made a number of grammatical and word changes to increase each exercise's clarity.

Interviews typically involved two interviewers, one for note taking and the other for interviewing. Notes were taken during the interview on each question. The project team or a subset met daily during interviews to make changes to questions addressing what was seen.

For example, as we interviewed, we discovered that some questions were too easy, and others were too difficult, and so we eliminated those. We also developed new questions based on student struggles and misconceptions we observed, and revised existing questions for proper interpretation. Once we had a sufficient number of working questions, we coded them for the learning goals and data structures they primarily covered and any additional learning goals they touched on. Each question was independently coded by two different team members, and then the two coders discussed and came to consensus on which learning goals it covered. We then developed new questions for any learning goals and data structures that did not have coverage.

6.1 Lessons Learned: Question Development

We quickly recognized a tension between the coverage of the CI and the time it takes to administer it. We felt that it was important for instructors to be able to give the CI in a standard 50-minute class session, which limited our coverage to some extent. For example, we do not have questions for every learning goal for every data structure, but we do cover all learning goals and all data structures. This compromise fit the goals of the CI — unlike a final exam, a CI does not need to cover all of the material of the course, but rather the most important conceptual core of the course.

Organizing and keeping track of different versions of questions with six people developing and interviewing was extremely difficult. We used a shared Google drive with a folder for each question and a Google document for each version of the question within a folder. However, it was still challenging for us to sync up, especially as the collection of questions we were using at any time were non-consecutively numbered. (For example, our notes from this time record that we had two different questions referred to by the number 33.) This became even more complicated as we started having students take different versions of the CI – the set of questions (and question versions) we were currently using became non-trivial to keep track of, and resulted in interviews that sometimes used the wrong versions of questions. We recommend that changes made to questions should be tracked so that the team can recall why particular wording changes were made. In sum, we strongly recommend having a process for organizing materials and handling these versioning issues (e.g., a versioning control system of some form).

We discarded far more questions than ended up on the CI. We developed 43 questions, with many questions having multiple versions. Ultimately, 13 working questions made the cut for our final CI. This points both to the importance of our student interviews, which weeded out 30 problematic questions, and to the need for a large set of potential questions.

7 STEP 3: CONVERTING TO MULTIPLE CHOICE QUESTIONS

For the next task, we converted the existing short-answer exercises into multiple-choice exercises where distractor answers represented common misconceptions. To discover these misconceptions and better understand student interpretation of the questions, we combined student interview data with a broader set of student responses from large-scale "study sessions". We held these study sessions at 2 institutions, for a total of 408 student responses.

These optional sessions were usually held in the evening or as part of a discussion or lab section led by Teaching Assistants (TAs). Students took the test, TAs collected it, and then the TAs went over the correct answers with the students. These tests included open-ended questions and multiple-choice questions with an "explain your answer" section added. This "explain your answer" section both let us clarify distractor answers and reassured us that students were choosing answers for the correct reasons.

After we collected the tests, we used Gradescope[2] to code the student answers. We adapted common incorrect answers into distractor answers for the multiple-choice versions of our questions. We also replaced distractor answers that were infrequently chosen and revised questions that were too easy or too difficult.

During this period, we met twice with our expert panel to solicit feedback about our questions. Specifically, we wanted to make sure

[2]From www.gradescope.com: "Gradescope is an assessment platform that reduces the time associated with grading in college courses via an optimized online workflow and clever application of artificial intelligence."

that they felt that our questions were both asking about concepts they felt were important and were of a difficulty level they felt was appropriate. Our first meeting yielded large changes to our questions, while the second meeting, which occurred after most of our interviewing, resulted in general approval of our questions.

7.1 Lessons Learned: Multiple Choice

It is worth noting that although we present the development of open-ended questions, the development of distractor answers, and the finalization of question wording as separate processes, they overlapped considerably. At many points in time, we had a collection of questions, some of which were still open-ended, some of which were multiple-choice but without a finalized wording and set of distractors, and some of which were finalized except for being polished. In one interview or test session we might be collecting new distractor answers for some questions, while checking the wording on more developed questions. Additionally, at some points questions that we thought were almost done were discovered to have some fatal flaw, which would require a complete redesign and restarting the interview process for that question.

The process of converting questions from open-ended to multiple-choice yielded surprising consequences as it effectively converted an application, analysis, synthesis, or evaluation question into a recognition question. This was somewhat incompatible with our learning goals, as questions with the goal "Design and modify data structures capable of insertion, selection, search, and related operations," when implemented in a multiple-choice format where students were choosing the correct code, became closer to those with the goal "Trace through and predict the behavior of algorithms (including code) designed to implement data structure operations."

We also found that in some cases, students frequently made mistake X on an open-ended version of a question, but rarely selected mistake X as a multiple-choice distractor. A good example appeared when students performed recursion on a binary tree: on the open-ended question, students often checked if the right child of a node was not null, and if so, returned the result of a recursive call to the right; then checked if the left child was not equal to null, and if so, returned the result of a recursive call to the left. This results in recursion to only the right side of the tree when both children exist. However, when this was offered as a distractor answer, students rarely selected it as a correct answer. Another tricky problem was one where we asked students to deduce whether an unknown implementation of a linked list (for example, one in a Java library) was a singly- or doubly-linked list. In the open-ended version, students frequently answered with "You can just look at the source code and check if `node.prev` exists" despite the fact that they would not have access to the source code in the scenario we described. However, when we converted this to a multiple-choice question, during interviews it became clear that having a distractor answer that involved checking the source code made students believe that they did have access to the source code in the scenario. We eventually eliminated this distractor answer because we felt that we could not include it without it being misleading.

For more about the student misconceptions we uncovered during our interview process, see our prior work [36].

8 STEP 4: VALIDATION INTERVIEWS

Next, we needed to ensure that students were interpreting the questions correctly and that the CI was successfully differentiating between students who understood the covered concepts and those who did not.

Numerous sources of guidelines provide support for testing with multiple-choice questions. Our criteria are aptly summarized in this excerpt from Kelly [18]: "(a) the item should be interpreted by all students in the same way; (b) the item should target a single problem so that its answer would be completely right or completely wrong, and not partly right and partly wrong; and (c) the difficulty level of the item should not depend on either obscure words or unintentional cues in the stem."

In order to determine if students were correctly interpreting CI questions, we held two rounds of validation interviews with 49 students across 5 institutions. In these interviews, students worked their way through the entire CI while thinking aloud. Interviewers did not interact with the students other than to prompt them to continue narrating their thoughts if they fell silent. We were especially alert to signs that students might pick correct answers for a question using incorrect logic or select incorrect answers using logic that was conceptually sound.

Each round of our validation interviews prompted revisions. The first round produced substantive changes and hence we felt it necessary to conduct another round of interviews. The second round produced only minor changes to the questions, so we felt comfortable moving forward without additional interviews.

8.1 Lessons Learned: Validation Interviews

Validation interviews were instructive and helped us understand when questions needed substantial revision, or in some cases, removal. Such an example of a fatal flaw caught in validation is a question where we told students to imagine a data structure that stored items as key-value pairs, where the key was an integer value that could be negative. We asked students to choose the data structures that would provide the best performance for implementing a `get` method. The correct answers were binary trees, which offered $O(\log n)$ time, and arrays, which could be searched in $O(\log n)$ time assuming pairs were inserted in key order. When we did validation interviews on this question, we discovered that both students who did very poorly overall and students who performed very well over all were choosing the array as the only correct answer, and claiming it had an $O(1)$ runtime. Students who performed very poorly overall assumed that anything with an integer key could be inserted into an array and then indexed by that integer (i.e. they were assuming that `array[-2]` was a valid operation). High-performing students circumvented the original intent behind the question by organizing data into an array in a manner that was extremely space inefficient but provided constant-time access. Adding a space constraint to the question made it too complex, so we ultimately removed it.

9 STEP 5: STATISTICAL VALIDATION

We were guided in our selection of statistical methods by our external evaluator, another evaluator she referred us to, and by recent work in CS education on other CIs. We referred to Chasteen et al. [6] as a general guide for how to statistically validate a CI, and then

consulted recent work by Xie et al. [33] as an example of validating a computer science CI. Guided by both of these works, we applied both Classical Test Theory (CTT) and Item Response Theory (IRT). Classical Test Theory provides metrics such as item discrimination and item difficulty that can be used to discover poorly-performing questions. However, its results are dependent on the population taking the CI, so every run must be analyzed individually. Item Response Theory is more robust to population differences, but the data must meet more strict assumptions in order for it to be valid.

To statistically validate our CI, we initially administered it at 8 institutions, including liberal arts schools, community colleges, primarily undergraduate institutions, and research institutions. We used the data from all of them for our difficulty statistics. However, not every run had enough participants to be statistically valid for our other metrics. The smallest run that had a large enough sample to be statistically valid had 64 participants. To create a statistically valid sample that equally represented all four institutions with a sufficient sample size, we combined our sample run with 64 participants with 64 participants selected at random from each of the other 4 institutions and used this as our sample.

Unfortunately, there were two questions that required small changes after our large-scale run. We re-ran the CI at three institutions: two R1s and a small liberal arts school. Of these runs, only one had sufficient sample size to be statistically valid for most of our measures. These measures served to confirm our previous statistical validation, and demonstrate that our changes did not undermine the statistical validity of the CI.

One measure we evaluated was item correctness, as we want questions to vary in difficulty. This requirement was based on our discussions with our expert panel, who had expressed concerns about the exam being either too easy or too difficult. Based on our interviews and open-ended runs to construct the distractors, we already had a good deal of evidence to suggest that our questions ranged widely in difficulty. We confirmed this intuition when we ran the instrument at a wide variety of institution types and found that while performance on individual questions varied from school to school, in general some questions were more difficult and some were less difficult. Our average correctness for questions varied from 18% of students answering the hardest question correctly to 82% answering the easiest question correctly. For more on our statistical analysis, see our prior work [24].

9.1 Lessons Learned: Statistical Validation

It can be difficult to both represent a variety of institutions (community colleges, small liberal arts schools, research institutions) and have large sample sizes, as some institutions by nature will have small class sizes. We found that the literature is not uniform on which statistics to run, what cutoffs to use, why validating CIs is different than validating other tests, why certain statistics do or do not have merit, and so on. We suggest documenting your decision-making process and offering your readers rationale and interpretation for each statistic that you present.

10 ADMINISTERING THE CI

Once a CI has been designed and validated, hopefully instructors and researchers will begin administering it. This raises the issue of how to properly administer a CI to a class. In this section, we describe what we learned from our administration of the CI throughout the development process.

Consider, for example, exam administration in a large-enrollment course, with multiple sessions. Given that there is no single session where all students are in the same room, should the exam be given in lecture, or in discussion, or in lab, or in some separately-scheduled room? Whatever the choice, there is the possibility of test-takers gaining access to the answers supplied in an earlier session.

An instructor must also decide how much the CI results should "count" in the course final grade, where options include:

- Volunteer (students get no credit for completing the CI)
- Extra credit (correct answers count toward the course grade, or participation counts for a set amount)
- Points count the same as any other points

Ideally, students are motivated to take the CI — say, by billing it as a review session for a class exam — but are not tempted to cheat. An optional review session may result in only highly-motivated students taking the CI, which will not produce an accurate assessment of students' knowledge. A review session given in a lecture or lab session has the advantage of encouraging students to take the CI without motivating them to cheat.

It may be difficult for researchers to find populations (outside of their own courses) in which to run the CI. We relied on our personal connections to find appropriate classes. We caution that this should be done sparingly, as the administration and bookkeeping time costs can add up. We are grateful to our colleagues for printing exams, supervising the students, offering value to the students in terms of reviewing the responses and misconceptions, and getting the data back to us.

11 CONCLUSION

Developing a concept inventory (CI) is an ambitious and time-consuming endeavor. However, a well-designed, validated CI can be invaluable to the research community in allowing a comparable measure of student learning across instructors, student populations, and pedagogies. Moreover, developing the CI will require establishing instructor consensus on course topics and learning goals, and student misconceptions on concepts. Both are valuable in their own right. While designing a CI may seem daunting, we hope this paper offers a road map and friendly advice to those interested in CI development. There are many CIs to be designed for computer science, and existing CIs to validate for new contexts and purposes. We hope you'll join in.

12 ACKNOWLEDGEMENTS

The authors gratefully acknowledge the contributions of Stephanie Chasteen for serving as our external evaluator; Soohyun Liao for help conducting interviews and validating the instrument; and Darci Burdge for help conducting interviews. The authors also thank our many collaborators on the project: Meghan Allen, Owen Astrachan, Maureen Doyle, John Glick, Paul Hilfinger, Kate Sanders, Paramsothy Thananjeyan, and Steve Wolfman. This work was supported in part by NSF award IUSE 1505001.

REFERENCES

[1] W. Adams and C. Wieman. Development and validation of instruments to measure learning of expert-like thinking. *International Journal of Science Education*, 33(9):1289–1312, 2011.

[2] D. L. Anderson, K. M. Fisher, and G. J. Norman. Development and evaluation of the conceptual inventory of natural selection. *Journal of research in science teaching*, 39(10):952–978, 2002.

[3] L. Arthurs, J. F. Hsia, and W. Schweinle. The oceanography concept inventory: A semicustomizable assessment for measuring student understanding of oceanography. *Journal of Geoscience Education*, 63(4):310–322, 2015.

[4] J. M. Bailey. *Development of a concept inventory to assess students' understanding and reasoning difficulties about the properties and formation of stars*. PhD thesis, University of Arizona, 2006.

[5] E. M. Bardar, E. E. Prather, K. Brecher, and T. F. Slater. Development and validation of the light and spectroscopy concept inventory. *Astronomy Education Review*, 5(2):103–113, 2007.

[6] S. V. Chasteen, R. E. Pepper, M. D. Caballero, S. J. Pollock, and K. K. Perkins. Colorado upper-division electrostatics diagnostic: A conceptual assessment for the junior level. *Physical Review Special Topics - Physics Education Research*, 8(2), 2012.

[7] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American journal of physics*, 69(9):970–977, 2001.

[8] H. Danielsiek, W. Paul, and J. Vahrenhold. Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pages 21–26, 2012.

[9] C. D'Avanzo. Biology concept inventories: overview, status, and next steps. *BioScience*, 58(11):1079–1085, 2008.

[10] J. Epstein. Development and validation of the calculus concept inventory. In *Proceedings of the ninth international conference on mathematics education in a global community*, volume 9, pages 165–170, 2007.

[11] M. F. Farghally, K. H. Koh, J. V. Ernst, and C. A. Shaffer. Towards a concept inventory for algorithm analysis topics. In *Proceedings of the 48th ACM Technical Symposium on Computer Science Education*, pages 207–212, 2017.

[12] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. Identifying important and difficult concepts in introductory computing courses using a delphi process. *SIGCSE Bulletin*, 40(1):256–260, 2008.

[13] R. R. Hake. Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American journal of Physics*, 66(1):64–74, 1998.

[14] S. Hamouda, S. H. Edwards, H. G. Elmongui, J. V. Ernst, and C. A. Shaffer. A basic recursion concept inventory. *Computer Science Education*, 27(2):121–148, 2017.

[15] G. L. Herman, C. Zilles, and M. C. Loui. A psychometric evaluation of the digital logic concept inventory. *Computer Science Education*, 24(4):277–303, 2014.

[16] D. Hestenes, M. Wells, and G. Swackhamer. Force concept inventory. *The Physics Teacher*, 30(1):141–158, 1992.

[17] K. Karpierz and S. A. Wolfman. Misconceptions and concept inventory questions for binary search trees and hash tables. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 109–114, 2014.

[18] F. J. Kelly. The kansas silent reading tests. *Journal of Educational Psychology*, 7(2):63, 1916.

[19] S. Krause, J. Birk, R. Bauer, B. Jenkins, and M. J. Pavelich. Development, testing, and application of a chemistry concept inventory. In *34th Annual Frontiers in Education Conference*, 2004.

[20] J. Libarkin, E. Ward, S. Anderson, G. Kortemeyer, and S. Raeburn. Revisiting the geoscience concept inventory: A call to the community. *GSA Today*, 21(8):26–28, 2011.

[21] M. C. Parker, M. Guzdial, and S. Engleman. Replication, validation, and use of a language independent CS1 knowledge assessment. In *Proceedings of the 12th ACM Conference on International Computing Education Research*, 2016.

[22] L. Porter, S. Garcia, H.-W. Tseng, and D. Zingaro. Evaluating student understanding of core concepts in computer architecture. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 279–284, 2013.

[23] L. Porter, D. Zingaro, C. Lee, C. Taylor, K. C. Webb, and M. Clancy. Developing course-level learning goals for basic data structures in CS2. In *Proceedings of the 49th ACM technical symposium on Computer Science Education*, pages 858–863, 2018.

[24] L. Porter, D. Zingaro, S. N. Liao, C. Taylor, K. C. Webb, C. Lee, and M. Clancy. BDSI: A validated concept inventory for basic data structures. In *Proceedings of the 15th ACM Conference on International Computing Education Research*, pages 111–119, 2019.

[25] C. Y. Read and L. D. Ward. Faculty performance on the genomic nursing concept inventory. *Journal of Nursing Scholarship*, 48(1):5–13, 2016.

[26] P. M. Sadler, H. Coyle, J. L. Miller, N. Cook-Smith, M. Dussault, and R. R. Gould. The astronomy and space science concept inventory: development and validation of assessment instruments aligned with the k–12 national science standards. *Astronomy Education Review*, 8(1), 2010.

[27] M. K. Smith, W. B. Wood, and J. K. Knight. The genetics concept assessment: a new concept inventory for gauging student understanding of genetics. *CBE—Life Sciences Education*, 7(4):422–430, 2008.

[28] A. Stone, K. Allen, T. R. Rhoads, T. J. Murphy, R. L. Shehab, and C. Saha. The statistics concept inventory: A pilot study. In *33rd Annual Frontiers in Education Conference*, 2003.

[29] C. Taylor, D. Zingaro, L. Porter, K. C. Webb, C. B. Lee, and M. Clancy. Computer science concept inventories: past and future. *Computer Science Education*, 24(4):253–276, 2014.

[30] A. E. Tew and M. Guzdial. Developing a validated assessment of fundamental CS1 concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, pages 97–101, 2010.

[31] K. E. Wage, J. R. Buck, C. H. Wright, and T. B. Welch. The signals and systems concept inventory. *IEEE Transactions on Education*, 48(3):448–461, 2005.

[32] K. C. Webb and C. Taylor. Developing a pre-and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 103–108, 2014.

[33] B. Xie, M. J. Davidson, M. Li, and A. J. Ko. An item response theory evaluation of a language-independent CS1 knowledge assessment. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 699–705, 2019.

[34] K. A. Young. Direct from the source: The value of "think-aloud" data in understanding learning. *Journal of Educational Enquiry*, 6(1):19–33, 2005.

[35] M. Zeilik. Birth of the astronomy diagnostic test: Prototest evolution. *Astronomy Education Review*, 1(2):46–52, 2002.

[36] D. Zingaro, C. Taylor, L. Porter, M. Clancy, C. Lee, S. N. Liao, and K. C. Webb. Identifying student difficulties with basic data structures. In *Proceedings of the 14th ACM Conference on International Computing Education Research*, pages 169–177, 2018.