# Directing Incoming CS Students to an Appropriate Introductory Computer Science Course

Leo C. Ureel II
*Michigan Technological University*
ureel@mtu.edu

James Heliotis
*Rochester Institute of Technology*
jeh@cs.rit.edu

Mohsen Dorodchi
*University of North Carolina, Charlotte*
Mohsen.Dorodchi@uncc.edu

Mireilla Bikanga Ada
*University of Glasgow*
Mireilla.BikangaAda@glasgow.ac.uk

Victoria Eisele
*Front Range Community College*
Victoria.eisele@frontrange.edu

Megan E. Lutz
*University of Georgia*
mlutz@uga.edu

Ethel Tshukudu
*University of Glasgow*
2371984T@student.gla.ac.uk

***Abstract*—Full Paper. Research. We discuss possible ways to direct students to right level of introductory programming. While many schools offer college preparatory or advanced placement courses in computing, there is still, unfortunately, a large part of the "college-ready" population that has no opportunity to learn computing at all before they arrive. Regulation of CS education at the state/province or national level is still rare (but growing). Thus incoming students possess a wide range of skills and knowledge. When coupled with increasing enrollments, this diversity of experience can result in courses having large numbers of both absolute beginners and seasoned coders. Such courses are difficult to teach, intimidate novice students, and bore those with more experience. This can result in low engagement and retention.**

**Unlike mathematics and language arts, introductory courses in CS vary widely from one institution to another in both conceptual material and programming language used. A standard point of entry to college mathematics is a calculus course, with some students instead starting earlier with pre-calculus or an algebra refresher, and others starting out in the second-term calculus course. There is rarely a concern about student skill being hidden by notational or other language differences, because the language of mathematics is close to universal. Similarly, freshman language arts courses in reading and/or writing assume a certain level of skill and maturity of comprehension and expressiveness in the target language; otherwise remedial courses are provided.**

**We investigate placement of incoming first year students into appropriate introductory computer science courses at higher education institutions where there is more than one choice of first course. The goal is to determine the best way to decide which first course would be the most helpful for each student.**

***Index Terms*—Computer Science Education, Placement, First Year Experience**

## I. Introduction

When Rochester Institute of Technology set out to create a new introductory CS course with a slower pace, the new course was designed to reach students with little or no previous computing experience. Immediately they were confronted by the question of how to properly place students into these courses. The goal of such placement is to minimize student frustration and decrease time-to-graduate.

We investigate the following main issues related to placing first year students into an appropriately paced introduction to computer science course:

- The ability to predict skill in the absence of prior experience,
- The value of programming language neutrality in an assessment instrument,
- Stigma and other self-efficacy issues associated with student performance, especially among groups underrepresented in computer science,
- The pace and difficulty of the target courses,
- Data regarding outcomes, satisfaction, and retention for students at schools where some kind of assessment and sorting is done, compared to those in one-size-fits-all introductory classes.

We conducted evidence-based research, through literature and available resources, on freshman college student readiness, utility of placement/entrance exams, and actual exams employed by universities.

The main assessment tools that are in use at the current time are (1) a standardized test such as the College Board AP exam, (2) a credit-by-exam (CBE) option, where students take something akin to an actual final exam, (3) a self-assessment-of-experience survey, or (4) personal interviews.

We present a summary of findings from literature as well as from our own prior experience. We show that placement strategies are often determined by the unique requirements of individual institutions. We describe the placement procedures implemented at a handful of schools. Finally, we make some recommendations for schools who wish to implement or revise their placement policies.

## II. A Selection of CS Placement Strategies

Through conversations, written communication, and published work, we list a variety of placement-by-examination approaches below.

### A. Michigan Technical University

Michigan Technological University (MTU) offers a three semester introductory programming sequence in Java that assumes no prior programming experience (CS1 Programming Fundamental, CS2 Object-Oriented Programming, CS3 Data Structures). Alongside this introductory sequence, MTU offers

an accelerated option for students with programming experience in any language [1]. The one-semester accelerated course covers the same material as the CS1+CS2 courses. The intent is to provide an introductory course in which novice programmers need not compete with experienced programmers, while students with programming experience learn at a pace that is both manageable and challenging.

All incoming majors are required to take a programming assessment during the summer before their fall enrollment in an introductory programming course. The assessment consists of twenty questions in two parts: the first portion asks about previous programming experience and the second portion asks questions specific to programming. Students who indicate they have had no previous programming experience are not required to take the second portion of the assessment [2].

The programming-specific portion of the assessment is language independent. Coding problems are presented in plain English (see Figure 1). Students may answer in any programming language, pseudo code, or even provide an English description of an algorithm.

```
Write a program that computes and prints
the sum of the even numbers from 17 up
to and including 329.
(ie., 18 + 20 + ... + 326 + 328).
You must use a loop.
```

Fig. 1. Example problem question from Michigan Tech assessment.

Prior to requiring the assessment, a 2015 survey showed 63% of students with no previous programming experience indicated that the standard introductory programming sequence was too difficult for novice programmers. At the same time, 66% of the students with programming experience indicated the sequence was too easy. In 2016, after all incoming students were required to take the assessment and were enrolled in the appropriate first CS course based on their score, a follow-up survey indicated that only approximately 25% of students with no prior experience said the course was too difficult. Meanwhile, enrollment for the accelerated introductory course rose to 74 students in the same semester, more than double the anticipated average enrollment [3]. An examination of successful completion of the introductory courses (completion of the course with a grade higher than D) before and after instituting the mandatory assessment and placement indicates a slight improvement in the percentage of students successfully completing CS1 as well as the overall successful completion rate if we total the completions of the standard CS1 and the accelerated CS1+CS2 courses together. The actual number of successful completions showed that a much larger number of entering students are capable of completing the accelerated CS1+CS2 course than were registering for that course prior to the change [2].

Surveys were conducted in Fall 2018 and 2019 to determine if students were satisfied with their placement upon completion. CS1121 (CS1) had 167 students in Fall 2018 and 136 in 2019. CS1131 (CS1+CS2) had 77 students in Fall 2018 and 70 in 2019. Figures 2 and 3 show that students were overwhelmingly satisfied with their courses.

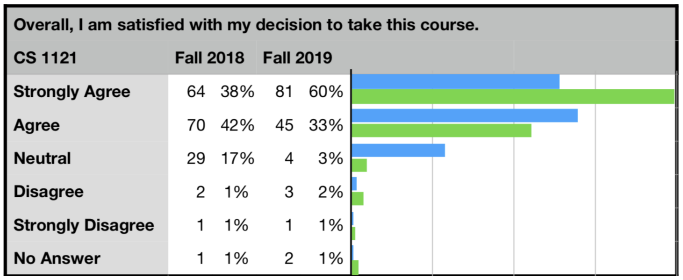


Fig. 2. Student satisfaction after completing CS1121 (CS1).

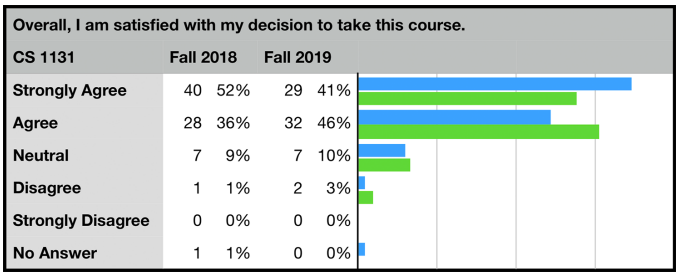


Fig. 3. Student satisfaction after completing CS1131 (CS1+CS2).

### B. *Rochester Institute of Technology*

Rochester Institute of Technology (RIT) also has the two paths described in §II-A, except that the exam used is the College Board AP exam, which admittedly excludes some students who could potentially qualify for the accelerated course. In rare cases, a student may petition for entry without having taken the AP exam.

A new study is just starting in RIT's Computer Science Department. The department is responsible for providing a unified introductory CS course sequence for majors in computer science, software engineering, computer engineering, computational mathematics, and several other disciplines. The 2-course sequence is fairly intense, covering standard CS1 and CS2 items as well as some graph algorithms, networked communication, and threading. No prior knowledge of computing is required, but the hypothesis is that more students could succeed, perhaps with reduced stress, if an alternative path through these courses were offered at a slightly slower pace, possibly extending the time in the sequence to three semesters in many cases. The faculty was loathe to follow the approach described in [4] where students are evaluated at the end of the first course because of the possibility of a student's loss of self-efficacy or interest after doing badly in the first course. The current proposal is to follow the Novice-Level CS0 model described earlier.

A pilot assessment exam was administered in the fall of 2019 to all students taking CS1. In questions with code, a custom language was used that was very similar to Python, the

language used in CS1. The exam grade did not count, except that students got points for taking it as part of their first week's homework assignment. Participation was therefore extremely high, at 95%.

The plan is to analyze exam scores in their relationship to performance in the actual course, and then either make appropriate adjustments to the exam for another pilot run, or if the exam seemed to discriminate well, assign a threshold grade for entry into CS1 versus the slower-paced curriculum. The new introductory course will not be offered until the exam appears to be an effective measure of anticipated success.

### C. University of Oregon

At the University of Oregon, CIS 210 is the introductory course for CS majors. It is officially advertised as having a "prior programming experience" prerequisite [5]. The reasons for the prerequisite, though deliberately ambiguous and in reality unenforceable, are as follows.

1) Make an attempt to at least somewhat flatten the terrain of prior student experience in CIS 210.
2) Distinguish the first course in the major from intro programming courses, which are offered at the 100 level as general education courses. The target audience for those courses is all students; no prior programming experience is necessary.
3) Do *not discourage* strong students with *no prior experience* who want to leap in and get started.

Students may therefore choose whether to start with a 100-level step-by-step introduction to programming course or with the 210 introduction to CS course. The latter course is still about programming to a very large extent, though with more depth and breadth, and taught using classic computer science problems.

To determine "prior experience", the department gives an eight-question, zero-points quiz on the first or second day. The instructors also show and discuss a slide containing a small amount of Python code that, in ten lines, illustrates about 16 fundamental concepts of computing. Although it is Python code, it is treated at that point as if it is pseudocode.

The quiz itself, which uses pseudocode, is not a validated instrument, but the questions on it are all taken from research papers. Thresholds have been established, but in the end students decide in which course to enroll.

Since the prior experience students bring to class still varies a lot, even for students who score well on the quiz, the faculty still find it a challenge to calibrate the course to meet the variety of student needs.

### D. University of Waterloo

The University of Waterloo (Ontario, Canada) [6] has three levels of what ends up being the same first-year introductory course. Within this explanation the courses will be called C, B and A

C The course for non-majors, or for majors that are struggling with the content

B The typical course for majors, and

A The advanced-level course for students that want to be challenged.

Students cannot automatically sign up for A – they initially self-select, but due to demand they are screened based on an application that is somewhat informal.

All three courses cover the same core concepts, but the courses increase in breadth and depth and difficulty.

The department enrolled approximately 1,750 students across the three courses in the fall of 2019. Each section has capacity for 90 students if necessary so that students can move down from B to C or from A to B up to about halfway through the course (typically after the midterm). If they switch down, they start with a "clean slate" with no grades carried forward, meaning that the remainder of their assignments are worth more and their final exam is weighted more.

## III. Survey of SIGCSE Members

An informal survey was sent out to the membership of SIGCSE, the ACM Special Interest Group for Computer Science Education. The questions were designed to find out what various faculties were doing with regard to the placement issue. The survey has no scientific validity. A summary of the answers received from 34 responses from distinct institutions follows.

The respondents tended to come from non-research-focused institutions.

- 38% Teaching institutions
- 47% Blended institutions
- 15% Research institutions

32% of the institutions claimed to have an internally-designed exam for CS course placement.

- Only 18% of those schools stated that the exam was required of all students wishing to enroll in CS courses.
- 64% of the schools giving exams stated that they were not geared toward any specific course content but instead toward general CS concepts.
- 45% of those schools claimed to have established numerical thresholds for being allowed into the more demanding class or classes, but only a couple of schools claimed that the rule was truly mandatory.

Figure 4 is an indication of confidence in the exam among the faculty who administer one. The top bar signifies the choice "extremely satisfied". The bottom bar (empty) signifies "extremely dissatisfied".

Then there was the question of who, in absence of an absolute grade threshold mandate on a test, makes the placement decision for new students. The answers here were quite varied, from faculty member to advisor to chair. In 6% of the cases, it was up to the students themselves. (15% of the respondents marked "other".)

## IV. Content Design of a Placement Exam

This section describes, at a high level, the various building blocks for a placement exam. More detail on how to make good questions, and how to measure the appropriateness of

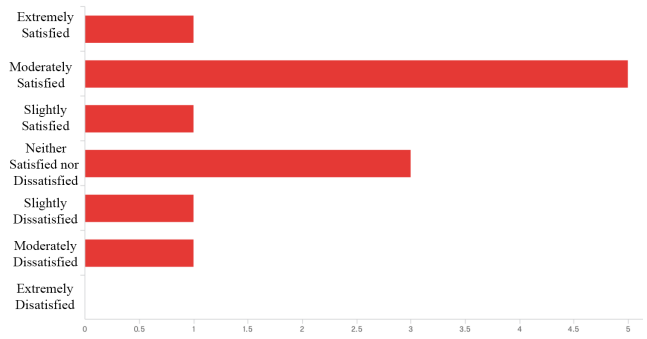


Fig. 4. *"How satisfied are you that your placement exam and placement policy are appropriate?"*

the chosen questions, is given later. The issue of pseudo code versus a synthetic language versus a real language is discussed in §IV-C.

### A. Methodology

There are some CS assessments that have been demonstrated reasonable success, such as in Román-Gonzáles et al [7], the First Computer Science 1 [FCS1, 8], and the Second Computer Science 1 [SCS1, 9]. The SCS1 was shortened and that reduced version was validated [10]. The approaches toward item and instrument development in those papers should be considered as near exemplars for schools and programs interested in developing their own placement exams, but with content, format, design and delivery selected keeping the purpose of the specific placement exam being developed in mind.

From the scant placement exams found where the questions or some sample questions are available, we have been able to classify them into two main groups: comprehension and problem-solving.

*1) Assessing Comprehension:* Most exam questions focus on the familiarity with, or the ability to grasp, computing concepts. Those concepts can be expressed through code in text form or in a graphical block language. The list below shows the popular ways that computing conceptual knowledge can be assessed.

- determining the result of executing code
- choosing the correct item for something missing in a snippet of code
- identifying an error in a snippet of code
- rearranging provided statements/expressions so that they accomplish the desired result [11]

These approaches can manifest themselves as multiple choice, true/false, or fill-in-the-answer questions.

Any serious writing of code by a student taking the exam, of course, makes it less likely that the exam could identify students without experience who would nevertheless have the potential to grasp concepts quickly.

*2) Assessing Problem-Solving:* The other form of question often described in written exams, and in fact in oral interviews as well, is the problem-solving question. Here some kind of real-world problem is described, and the student is asked to suggest a solution, possibly in choice of data structures, procedure, high-level design, or some combination of these.

### B. Format

It is appropriate to include a focus on general, broadly asked questions such as writing small blocks of pseudo code, asking what a provided block would do, choosing which of several blocks of code would solve a given problem, or rearranging the statements in a block of code so that they perform the required functionality [11]. The publication by Román-Gonzáles et al. showed such an assessment exam for a middle school audience [7].

Design of a placement exam is critical. In Tew [8], the author describes one process of creating a well designed assessment. The exam can be of any format, multiple choice, free response, include partial-credit Parsons problems [11], or may include a combination of item formats. The format will be determined by the purpose and goals of the assessment. In most cases, creating multiple choice questions is more difficult and time consuming than creating free response questions, but the multiple choice questions are quicker to score and facilitate automatic scoring. Another important aspect of exam design is that a variety of difficulty levels among the items is represented so accurate and precise estimates of student abilities can be obtained.

### C. Language(s)

There is a growing belief that computational thinking is something that we can define, discover, and assess without having to specify a computing paradigm or certainly a programming language. There appears to be evidence of this belief in the design of the SCS1 language, and in the fact that some schools choose to use graphical block languages. Mark Guzdial wrote a blog entry that is a good summary on the FCS1 and SCS1 languages [12].

However one must keep in mind that there are often assumptions made, conscious or otherwise, about what makes up the set of fundamental concepts or ways of thinking that will predict student success in CS. For example, consider the choice of language construct to represent assignment. Three test designers, whom we will label A, B, and C, are working on an exam question in the category of comprehension. A and B choose a single equals sign for variable assignment without giving it another thought, because the currently successful languages that do assignment all follow FORTRAN's lead and use that punctuation. C thinks a bit more carefully and states her observation that, in the far more universal written language of mathematics, the equals sign is either an assertion or a question regarding the equivalence of the two sides. She suggests instead use of the "`:=`" or "`<-`" of some earlier languages.

Although C's choice could definitely be argued as superior, it is still missing a larger point: semantics. Consider Figure 5 containing the question they eventually constructed.

```
What are the values of a, b, and c after
the following instructions have been
executed?

a <- 3
b <- 5
c <- a + ( 2 * b )
a <- 19
```

Fig. 5. Exam Question with Assignment

Many would state without hesitation that `c` is equal to 13 because of the values of `a` and `b` at the time the assignment was made. What may not be clear is that an assumption has already been made, that the code in the question is to be interpreted in a purely imperative fashion. It is in the personal experience of one of the authors that occasionally a student will look at the third line of code as *declarative*, that it states a rule that will be followed from then on, thus giving the answer that `c`=29.

If one wants to be really fair in one's exam design, one comes to the conclusion that the syntactic and semantic rules of the language, even if it is called pseudo code, must be provided to the test-taker. A new choice then appears.

1) Provide a "language manual" to everyone before the exam so they can be familiar with the notation used when they start the exam.
2) Explain the language, possibly incrementally, within the exam itself.

There are serious issues with either choice. In the first plan with a published manual, a new variable is introduced: how much time each student took to prep for the exam. It also makes it less likely that the exam assesses "innate ability". The second plan greatly lengthens the test, making it less practical to be statistically thorough, and potentially, although not formally investigated yet, increasing anxiety and the consequences that go with it. Finally, the more explanation one must add to an exam increases the likelihood of unconscious bias creeping in.

It is therefore incumbent that a more practical, but imperfect, alternative be included in this report. One could replicate an exam design using each of the programming languages seen as most-used by students of secondary school age. This approach would then be a fair test for the students who have been exposed to computing in those languages. For the minority that remains, the pseudo code approach with explanations could be attempted, or perhaps a language that is widely viewed as being close to pseudo code, e.g., Python could be the language of choice, but still with more syntax/semantics explanation attached, and more time allowed to digest it all. At this point one is left with separate vehicles whose results could never be combined for study, continuous improvement, or placement.

An alternative approach, that is not for everyone, is to present problems in natural language and allow the students to respond with solutions in any language, including natural language. This is the approach described in §II-A. The advantage of this approach is that it is language independent and students are tested on their ability to solve problems using a language they know. The disadvantage is that scoring the solutions is a laborious task that cannot be automated, except that it can be assigned to graduate students.

## V. Considerations on the Impact of Placement

### A. Utility of Self-Reporting

Many schools have employed surveys, rather than technical exams, wherein students evaluate their own abilities. Examples of survey questions are as follows.

- Have you taken a programming course before?
- What programming languages have you learned?
- How big (lines of code) is the largest program you have written?
- How confident are you in your ability to program?
- How confident are you in your ability to succeed in this course?

Demographic classification questions are also often included, depending on the purposes of the survey.

In an early work by Evans, et al. [13], a 100-question survey was conducted for students enrolled in the required first programming course with the goal of finding the best set of predictors amongst demographic, behavioral, cognitive, and problem-solving classifications. They found that no single classification category dominated the set of variables over the others.

Smith et al. [14] conducted a study that quantified the measurement approaches of student's prior knowledge in programming. In comparing aptitude tests and surveys, their results showed that the latter are limited in that they mainly reveal the different topics to which students have been exposed and do not measure the amount of knowledge students have gained.

Another study [15] developed a survey that helped them understand the prior programming experience of the the students. Among other questions, the survey included the language learned, the level of fluency, the source of each student experience, as well as the student's reflections on the usefulness of the experience. The authors reported that students showed bias in answering some questions because of the guided responses given by the survey. The reported bias presents itself as a limitation in assessing the prior experience.

### B. Impact on Underrepresented Groups

There are differences in CS participation across the social and ethnic groups. For instance, it is reported that students belonging to low socio-economic groups and some ethnic minority groups are less likely to study CS. The issues of recruitment and retention in CS are global and well documented in the literature [1]. These include students' misconception of CS, the lack of clear understanding of the potential career

path in CS, and students' low senses of self-belonging and self-efficacy.

There is an increasing drive to recruit underrepresented groups. Many issues affect the uptake of CS in higher education despite the influence of computing in all aspects of our lives and a growing demand in CS related jobs. Different analyses of K-12 students in the US suggest that there are many cognitive and social factors that impact students' willingness to engage in computer science problem solving. In particular, such factors cause gender disparities in computer science achievement as young as ten years of age [16].

Care must be taken to avoid exam questions with latent bias that perpetuates discrimination against underrepresented groups. As some examples, one cannot assume prior access to computer-related tools and applications, and that questions cannot include references that make sense only in relation to certain cultures. Bias in items may not be readily apparent, so pilot testing all items and analyzing them for bias is imperative before using them in practice.

### C. Stigma - Student and Faculty

Many students with no previous programming experience find the introductory programming course difficult. In [17], a potential link was demonstrated between prior experience and students' expectations, work habits, attitude and confidence, and perceptions of self and peers. Nevertheless, while prior knowledge of programming may influence students' decisions to enter CS and their progression in the first programming course, this is not necessarily the case in subsequent courses [4]. Students' self-perception of their programming ability has been found to have the largest association with their decision to undertake CS and their subsequent progression [15]. Even when such a student decides to take the course, not being in a class with students with similar programming background or skills level may have a negative impact on whether he or she completes the course [3]. In-depth interviews have revealed that gaps in prior experience can be a source of stress for those with less experience in a pair-programming environment, also contributing to lower self-confidence, and feelings of intimidation [17]. Conversely, students with prior knowledge are more confident and believe in their ability to succeed. Nevertheless, those with no prior knowledge are still able to succeed if they put in effort and time [17].

Female students with no programming experience tend to drop out and tend to underrate their self-efficacy more than males [3]. In [18], students with and without prior experience took a slower-paced introductory course, despite the majority of respondents agreeing to have a better understanding of computer science than before, just under half of the respondents would have preferred to skip directly to the first computer science course. Furthermore, those without prior experience did a full letter grade better on average in the subsequent course than those with the prior experience.

Of further concern may be the attitude of the instructors. Care must be taken not to give students the impression that they are deficient in some way and need extra help to bring them *up* to the standards of a so-called "normal" student, and they should not be treated differently in subsequent courses because of the path they took to get there.

## VI. Recommendations

When designing assessments or new instruments, it all comes back to the research question or questions. The purpose dictates the design, the data that are collected, the appropriate analyses, and then the interpretation, limitations, and next steps. With the collaboration or consultation of experts, common missteps or pitfalls can be avoided, improving the rigor and validity of our work. Our recommendations focus on ways to §VI-A assess the placement exam, §VI-B continuously improve the assessment, and §VI-C avoid common pitfalls in reporting results.

### A. Assessing the Assessment

Often discussed is the approach used to specify the threshold to placing students into different introductory CS courses. This depends on the goals and content of the assessment. For example, we may want to assess that the student possesses enough prerequisite knowledge to succeed in the course vs the student being placed into a downstream course. Institutions should rely on the intuition of the course instructors to place the initial threshold. From there, the institution should adjust the threshold based on an analysis of the assessment results. Here we present a discussion of how to assess the assessment.

Whatever the form of the placement exam that a program chooses to use, the purpose needs to be understood, so that its performance can be properly assessed. Different CS programs may have different goals for their placement exam, with different student populations who would take the test. Placing incoming students with no prior experience into CS0 or CS1 (one specific population and one specific goal) would require an instrument with different psychometric properties than one for placing incoming students into CS1 or an accelerated CS1/CS2 course (a different, more varied target population with a different goal). If the same placement test were given for both programs, neither purpose would be well served and most students would be poorly assessed.

There are several accepted ways to assess that an instrument is performing as it should; this report will focus on the item response theory (IRT) approach to test and item assessment. While the test development itself is an iterative process, as described in §IV, once it has been piloted to a representative sample from the target population, follow-up analysis on the item properties must be conducted. These analyses are conducted to assess the ongoing validity of the instrument and to refine the items that are included; some items may need to be rewritten if bias against subgroups is found or if phrasing is revealed to be too confusing; items may need to be removed from the instrument completely if a rewrite can't fix them or if there are other issues. Alternatively, an analysis of the items can reveal miskeyed items, which can be easily resolved by correcting the answer key.

Construct validity is assessed in several ways: there is *convergent validity*, in which scores on the instrument have a positive association with scores on a previously validated instrument that measure the same outcome. Similarly, an instrument should have a weak or no association with instruments that do not purport to measure the same thing, i.e., *divergent validity*. Taken together, convergent and divergent validity give evidence that the test is measuring what it should and not what it shouldn't. However, as these are correlations, we must be mindful of our interpretations of them as well as factors that influence correlation, so that too much is not made from them (see §VI-C). Programs that use placement exams may be especially interested in *criterion validity*, after placement has occurred, to see how well the placement of students based on the test has worked [19].

Many of the validated assessments in the literature we reviewed are multiple choice (MC) assessments, which means that guessing the correct answer is a possible outcome even for students with no knowledge of the subject matter. For MC tests, the 3-parameter logistic model (3PL) [20] is the most appropriate model for capturing that probability [21]. The three parameters of the 3PL are the lower asymptote (a pseudo "guessing" parameter), the item discrimination (roughly the "slope" of the item), and the item difficulty (the "location" of the item on the ability scale).

An MC item with four alternatives (one correct option and 3 distractors) has a theoretical probability $p = 0.25$ of being answered correctly by random chance. With that context in mind, an inspection of the estimates for the lower asymptotes for the items is a good place to start when doing a distractor analysis. If an item's lower asymptote is much higher than 0.25, that is a good indication that one of the distractors is not functioning and examinees are really only using three of the four alternatives available and the extra distractor can be removed from the item. Similarly, if for a four-alternative item, the lower asymptote estimate is found to be much lower than 0.25, that is an indication that the item is harder than anticipated and that examinees struggle to distinguish between the correct answer and the distractors. A careful reading of the distractors or think-aloud sessions with examinees may be needed to understand the issue.

Item discrimination describes how much information an item contains about examinees with very similar abilities. Highly discriminating items indicate a large amount of information between similar students: for small changes in student ability, there is a very large change in the probability of answering the item correctly. On the other hand, items with low discrimination do a poor job of telling similar students apart: there is very little change in the probability of a correct answer for a small change in student ability [21]. Moderate to highly discriminating items provide more precise estimates of student abilities near those item locations. On a placement exam of the sort we propose, more precise estimates of student abilities would be associated with a reduced risk of placing students into an incorrect course.

The last item property one should look at is the difficulty, which informs the overall difficulty of the test. The test difficulty should approximately equal the ability of target population to maximize the information gained about the target population. As this population differs for different institutions, the average difficulty on each program's placement test will naturally vary: we caution against borrowing another university's placement exam without a comparison of the incoming classes on a variety of measures. The most precise student estimates will be those whose abilities match the average difficulty of the test; if the test is too hard (or too easy) for the test group, a large portion of items will be answered incorrectly (or correctly) by all or most test takers and that precision will be lost. Tests must be matched to the ability level of the target population. Items that are too difficult or too easy for the target population should be replaced.

A major concern is the institutionalization of implicit bias, item bias should be investigated. Differential item functioning (DIF) is an IRT approach to looking for subgroup bias [22], i.e., how item properties may change for different subgroups of the population. Sometimes the cause for bias may be readily apparent, such as the context in a question being more familiar to domestic examinees than international examinees. At other times, the cause for DIF may not be readily apparent. In either circumstance, use of quantitative techniques to measure the magnitude of the bias on each item and investigation into whether such bias is present is imperative. If severe DIF is present, such that subgroups tend to receive much lower exam scores because of item properties and not their own abilities, those items should be inspected and, if necessary, rewritten or removed, to eliminate measurement bias that would adversely impact underrepresented groups in the placement process.

Regardless of the scoring method and item design used, items that have low discrimination, questionable lower asymptotes based on the theoretical guessing probability, or difficulties that are a gross mismatch for the target population and purpose, should be inspected and reworked or removed from the assessment.

### *B. Continuous Improvement*

We have identified several dimensions in maintaining an effective college-level placement exam, among them: content, cognitive domain, and language. For any assessment to be effective, it must not be stagnant, but instead be continuously developed and improved along each dimension for it to remain a valid tool for the purpose of placing students in different courses. Content improvement includes following the desired knowledge and skill level for students entering the next level, and responding to the changes of computing education at the school level, recognizing that students are getting more (hopefully not less) access to computers at earlier ages, and slowly gaining more prior experience [23]. Similarly, as computer science departments update the language they use, or as new or different languages or paradigms (e.g., block languages) become more prevalent at the school level, the assessment designers may wish to revisit language or language independence employed.

At the same time, the cognitive domain of questions needs to be monitored and updated when necessary. One possible cognitive domain model of many to classify the levels of complexity and specificity is Bloom's Taxonomy, whose use is described in [24]. The authors provide a rubric to assess the question's two-dimensional cognitive level based on Modified Bloom's Taxonomy, which can be fine-tuned based on the desired level for each item on an assessment; ideally the instrument will span the entire domain used, to assess all areas. This is different than psychometric difficulty, discussed in §VI-A.

It must be emphasized that placement exams do not change with course content or outcomes the way course exams do. Placement exams change because changes in student demographics and/or course syllabi mean that the existing placement exam no longer accurately sorts students to the course that maximizes their potential for success.

### C. Pitfalls in Scientific Studies

One dimension along which we sought to classify the results that we compiled was the strength of the evidence therein. For each study, we considered if the results were convincing based on the quality of the study's design, the appropriateness of the statistical or psychometric analysis to the research question(s) posed, the completeness and transparency of the reporting, and the accuracy of the communication and interpretation. Throughout this report, we strove to reference only those studies that satisfied this criterion, or we have otherwise specified what aspect of a study we felt was exemplary. As a field of research, however, we found there is urgent need for improving our rigor when conducting quantitative studies. This section briefly outlines some suggestions for best practices toward that goal.

Page limit requirements imposed on contributors may lead to critical decision making regarding what content must remain in a final submission, however a fundamental principle of open science demands that a full accounting of methods and data be made available for the purpose of reproducibility; when that runs counter to page limits, we must make that available via a permanent online repository, clearly and freely linked from our reports [25].

We must be mindful of the inherent variability that exists when sampling, and include the standard error or other estimates of uncertainty with our point estimates, and include language of uncertainty in our interpretations as well. We must also remember the nuance in our interpretations of effect sizes, and what is of practical importance to us (e.g., what is a meaningful or large effect) and decouple that from statistical significance, which can be achieved simply be obtaining a large sample, recalling that one does not imply the other [26]. The context in which we conduct our analyses is critical to understanding these effects and our interpretations of what is meaningful [25]–[27]. It is easy to obtain a small $p$-value through large samples; this must be interpreted with the total context of the study (sample size, effect size, scales and units of measurement) so the actual magnitude of the effect can be understood. If correlations are reported, we must recall Cohen's warning [27] that a large effect is only large in the context of the research area, and using the heuristic boundary lines that have become so entrenched preclude critical thinking and understanding of the data and relationships.

## VII. Conclusion

Many academics in computer science wish they had the same infrastructure as other sciences that would help them decide which first course would be the most helpful for each student starting in their major. This is a laudable goal, but challenging to achieve.

It is important to not begin by sitting down to design an exam. You must have the choices of courses in hand with well-described, at least in terms of requirements, or *preconditions*, and outcomes, if not also syllabi. Then the academic unit must make sure it has the resources to offer sufficient sections of the courses. The institution's registrar might even have to be involved.

The issues that come up in exam design include choice of

- programming language, or lack thereof
- question types (concepts,problem-solving)
- exam delivery
- grading procedure or mechanism
- student enrollment process

In addition, it must be decided if factors outside of the exam score will be taken into account when deciding placement.

The exam, once written and agreed upon, still probably needs one, two, or three terms of pilot study, during which you do not split the students but instead put them all in the same course (your best choice as to what serves the students best). During this time, the other courses are not used; if they are used by students having taken the pilot exam, you will have a much harder time making any overall conclusions about the exam's ability to discriminate appropriately.

Even after the test is in use, a process for continuous improvement is necessary, just like it is for any course.

## Acknowledgment

This work is partly supported by NSF under Grant 1820862, and by NCWIT's Extension Services for Undergraduate Programs which is generously supported by NSF (Awards HRD- 1203148/1203198/1203174/1203179) and Google.org. We also thank two PhD students: David Williams of the Dublin Buisiness School, and Liat Nakar of the Weizmann Institute.

## References

[1] G. Archer, L. Bohmann, A. Carter, C. Cischke, L. M. Ott, and L. Ureel, "Understanding similarities and differences in students across first-year computing majors," in *Frontiers in Education Conference (FIE), 2016 IEEE*, IEEE, Erie, PA, USA: ACM, 2016, pp. 1–8.

[2] G. Archer, B. Bettin, L. Bohmann, A. Carter, C. Cischke, L. M. Ott, and L. Ureel, "The impact of placement strategies on the success of students in introductory computer science," in *Frontiers in Education Conference (FIE)*, IEEE, Cincinnati, OH: ACM, 2017, pp. 1–9.

[3] L. Ott, B. Bettin, and L. Ureel, "The impact of placement in introductory computer science courses on student persistence in a computing major," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ACM, Larnaca, Cyprus: ACM, 2018, pp. 296–301.

[4] E. Holden and E. Weeden, "The experience factor in early programming education," in *Proceedings of the 5th conference on Information technology education*, ACM, Salt Lake City, UT, USA: ACM New York, NY, USA, 2004, pp. 211–218.

[5] K. Freeman, *Placement exams for introductory cs courses*, Personal correspondence, Aug. 2019.

[6] C. D. o. Computer Science, *1a cs course rule*, https://cs.uwaterloo.ca/1acourses, Accessed: 2019-09-14, 2019.

[7] M. Román-González, J.-C. Pérez-González, J. Moreno-León, and G. Robles, "Can computational talent be detected? predictive validity of the computational thinking test," *International Journal of Child-Computer Interaction*, vol. 18, pp. 47–58, 2018.

[8] A. E. Tew, "Assessing fundamental introductory computing concept knowledge in a language independent manner," PhD thesis, Georgia Institute of Technology, 2010.

[9] M. C. Parker, M. Guzdial, and S. Engleman, "Replication, validation, and use of a language independent cs1 knowledge assessment," in *Proceedings of the 2016 ACM conference on international computing education research*, ACM, Melbourne, VIC, Australia: ACM New York, NY, USA, 2016, pp. 93–101.

[10] R. Bockmon, S. Cooper, J. Gratch, and M. Dorodchi, "(re) validating cognitive introductory computing instruments," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ACM, Minneapolis, MN, USA: ACM New York, NY, USA, 2019, pp. 552–557.

[11] D. Parsons and P. Haden, "Parson's programming puzzles: A fun and effective learning tool for first programming courses," in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, Australian Computer Society, Inc., Hobart, Australia: Australian Computer Society, Inc., 2006, pp. 157–163.

[12] M. Guzdial, *Blog: We should stop saying 'language independent.' we don't know how to do that*, https://cacm.acm.org/blogs/blog-cacm/238782-we-should-stop-saying-language-independent-we-dont-know-how-to\-do-that/fulltext, Accessed: 2019-09-14, 2019.

[13] G. E. Evans and M. G. Simkin, "What best predicts computer proficiency?" *Communications of the ACM*, vol. 32, no. 11, pp. 1322–1328, 1989.

[14] D. H. Smith IV, Q. Hao, F. Jagodzinski, Y. Liu, and V. Gupta, "Quantifying the effects of prior knowledge in entry-level programming courses," in *Proceedings of the ACM Conference on Global Computing Education*, ACM, Chengdu,Sichuan, China: ACM New York, NY, USA, 2019, pp. 30–36.

[15] G. Strong, C. Higgins, N. Bresnihan, and R. Millwood, "A survey of the prior programming experience of undergraduate computing and engineering students in ireland," in *IFIP World Conference on Computers in Education*, Springer, Dublin, Ireland: Springer, Cham, 2017, pp. 473–483.

[16] J. Tsan, K. E. Boyer, and C. F. Lynch, "How early does the cs gender gap emerge?: A study of collaborative problem solving in 5th grade computer science," in *Proceedings of the 47th ACM technical symposium on computing science education*, ACM, Memphis, Tennessee, USA: ACM, 2016, pp. 388–393.

[17] A. Tafliovich, J. Campbell, and A. Petersen, "A student perspective on prior experience in cs1," in *Proceeding of the 44th ACM technical symposium on Computer science education*, ACM, Denver, Colorado, USA: ACM New York, NY, USA, 2013, pp. 239–244.

[18] C. Marling and D. Juedes, "Cs0 for computer science majors at ohio university," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, ACM, Memphis, Tennessee, USA: ACM, 2016, pp. 138–143.

[19] P. Kline, *A handbook of test construction (psychology revivals): introduction to psychometric design*. London: Routledge, 2015.

[20] A. Birnbaum, "Statistical theories of mental test scores," in. Boston, MA: Addison-Wesley, 1968, ch. Some latent trait models and their use in inferring an examinee's ability, pp. 17–20, edited by Lord, Frederic M and Novick, Melvin R.

[21] S. E. Embretson and S. P. Reise, *Item response theory*. New York: Psychology Press, 2013.

[22] D. Gamerman, F. B. Gonçalves, and T. M. Soares, "Differential item functioning," *Handbook of Item Response Theory, Volume Three: Applications*, vol. 3, pp. 67–86, 2018.

[23] B. Ericson, *Ap cs report for 2018*, Accessed: 2019-09-14, 2019. [Online]. Available: https://cs4all.home.blog/2019/02/28/ap-cs-report-for-2018/.

[24] M. Dorodchi, N. Dehbozorgi, and T. Frevert, "'i wish i could rank my exam's challenge level!": An algorithm of bloom's taxonomy in teaching cs1," in *2017 IEEE Frontiers in Education Conference (FIE)*, IEEE, Indianapolis, IN, USA: IEEE, 2017, pp. 1–5.

[25] R. E. Kass, B. S. Caffo, M. Davidian, X.-L. Meng, B. Yu, and N. Reid, *Ten simple rules for effective statistical practice*, 2016.

[26] S. Greenland, S. J. Senn, K. J. Rothman, J. B. Carlin, C. Poole, S. N. Goodman, and D. G. Altman, "Statistical tests, p values, confidence intervals, and power: A guide to misinterpretations," *European journal of epidemiology*, vol. 31, no. 4, pp. 337–350, 2016.

[27] J. Cohen, *Statistical power analysis for the behavioral sciences*. New York: Routledge, 2013.