Uses, Revisions, and the Future of Validated Assessments in Computing Education: A Case Study of the FCS1 and SCS1

Miranda C. Parker
University of California, Irvine
Irvine, CA, USA
miranda.parker@uci.edu

Mark Guzdial
University of Michigan
Ann Arbor, Michigan, USA
mjguz@umich.edu

Allison Elliott Tew
AET Consulting
Seattle, Washington, USA
allisonetew@gmail.com

ABSTRACT

Validated instruments of knowledge are important for creating an accepted and shared measurement for the most important part of education – student learning. The Force Concept of Inventory from Physics Education Research has had a significant impact across STEM education. The Foundational CS1 (FCS1) and Second CS1 (SCS1) assessments were created and validated to further computing education research. Now, ten years after the publication of the FCS1 assessment and five years after the release of the SCS1 assessment, we can trace the use and the impact that these validated instruments have had on the needs and knowledge of the computing education community. In this paper, we examine how the FCS1 and SCS1 assessments have been used and adapted. We use this discussion to guide a comparison between our field and physics education to give a sense of direction to future research. In looking back on the use of these validated instruments, we can better understand our needs in computing education research and about our future needs in assessment.

CCS CONCEPTS

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

KEYWORDS

assessments, CS1, validity

ACM Reference Format:
Miranda C. Parker, Mark Guzdial, and Allison Elliott Tew. 2021. Uses, Revisions, and the Future of Validated Assessments in Computing Education: A Case Study of the FCS1 and SCS1. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021), August 16–19, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3446871.3469744

1 INTRODUCTION

When the argument for the validity of the Foundational CS1 (FCS1) assessment was first published in 2011 [65], the computing education community had few validated instruments for computer science (CS) knowledge. Concept inventories had been created for a handful of introductory computing topics [22, 26, 62]. The first validated instrument for CS1 knowledge in one language (Java) is likely Decker's in 2007 [12]. However, the FCS1 was noteworthy for providing a validated measure of knowledge across multiple programming languages for a first-semester undergraduate computer science course (CS1), which has always been a focus in computing education research [49]. We distinguish here between validated measures of knowledge and validated measures of other important variables in education, such as attitudes [14].

The FCS1 was built using pseudocode for the programming problems, which gave it a wide range of applications. The FCS1 assessment was created with the initial intent to compare learning across CS1 courses taught in different languages. The pseudocode-based FCS1 assessment was then validated with students learning Java, Python, and MATLAB. However, the mere existence of a valid CS1 assessment did not inherently make it useful to the community. Access to the FCS1 assessment was limited to prevent saturation of the assessment, which would weaken the argument for the validity of the FCS1. In response, the FCS1 assessment was replicated and validated to create the Second CS1 (SCS1) assessment in 2016 [43] which could be distributed freely without danger of weakening the FCS1 validation. The similarly multi-lingual pseudocode-based SCS1 was released to the computing education community upon request (as opposed to publicly available) and has been used by the community in research studies accordingly.

We wrote this paper to understand the impact and use of a validated instrument within the computing education research community. We trace the impact of the FCS1 and SCS1 assessments through published works citing the original papers, highlighting how the assessments have been used and adapted. Although the FCS1 and SCS1 are only two validated assessments, they serve as exemplars to consider more broadly the role of assessments in CS education and how assessments can influence other work. One research question we hoped to answer was whether use of the FCS1 and SCS1 stayed within the argument for validity that we originally constructed, or if either assessment was used for purposes beyond the initial intent. We also connect back to the original vision of the FCS1 and SCS1 by comparing where CS education research is to where Physics education research was, in terms of validated instruments of knowledge. Explicitly, the SCS1 was originally developed and distributed with the hopes of conducting a larger scale study of methods for teaching computing, as Hake did with the Force Concept Inventory (FCI) in Physics education research [25]. We end this paper by drawing on the discussions to suggest where the future of assessment in computing education may go, based on current community needs.

RQ1: Did uses of FCS1 and SCS1 stay within the original arguments for validity?

ICER 2021, August 16–19, 2021, Virtual Event, USA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8326-4/21/08.
https://doi.org/10.1145/3446871.3469744

RQ2: Can we do a large-scale study of CS methods based on use of FCS1 and SCS1?

2 DEVELOPMENT AND USE OF FCS1

The FCS1 was originally developed in the context of the trio of CS1 courses at Georgia Institute of Technology (Georgia Tech) [19]. Georgia Tech required computer science for all of its students, and found that a single course did not meet all students' needs [23]. There were eventually three different courses offered: one in MATLAB for Engineering students, one in Python for computer science and science students, and a different Python course for liberal arts, architecture, and business students. There was a question of how to compare these courses, given they are all taught in different programming languages but all serve the same introductory computing requirement. Elliott Tew developed the FCS1 in pseudocode so that it might be used to compare learning across different CS1 implementations in any of Java, MATLAB, or Python. She chose MATLAB and Python because of the courses at Georgia Tech, and she included Java because of its popularity as a CS1 language. She validated the test with students at several institutions. Thus, from the outset, the FCS1 was meant to serve a wide audience, beyond a single programming language and a single institution.

The first challenge facing Elliott Tew was defining the content for the exam. There is not a single common syllabus or even set of topics for the introductory CS course across the United States or around the world. The most common approach to defining the topic list for a concept inventory is to survey experts [61]. A Delphi approach was being used to identify the content for a concept inventory for CS1, but the range of topics was large [21]. Unlike the FCI in Physics education, a concept inventory for CS1 could not rely on decades of research about students' alternative conceptualizations of the real world. CS is a science of the artificial [56]. CS1 topics are invented, and some are relatively new with little empirical work describing students' conceptualizations. For example, Elliott Tew had little about object-oriented programming in the FCS1 because of a large variance across classes for what constituted object-oriented programming and because of a corresponding lack of literature exploring student conceptualizations of different kinds of object-oriented programming [63]. The developers of the FCS1 needed a core set of concepts for which a reasonable set of distractors could be defined. Elliott Tew used a textbook analysis technique to define a common set of concepts, which were then filtered through the use of recommended curriculum standards and then expert review [64].

One of the challenges to writing a multiple-choice question assessment is determining the distractors. The right answer is obvious, but the wrong answers have to be close enough to the correct answer to serve to discriminate different conceptual understandings. Elliott Tew gave the FCS1 problems to students studying MATLAB, Java, and Python as open-ended questions, without answer choices. She then took the most popular wrong answers as her distractors. Elliott Tew found the most common wrong answers were mostly the same across all three languages [63]. Some of the wrong answers were easily explained by language-specific features, e.g., Elliott Tew's pseudocode used zero-based array indexing (as in Python and Java) which was a source of error for the students who learned MATLAB in their CS1 (since MATLAB indexes arrays starting at one). The commonality of errors across languages is also an interesting finding deserving of further investigation – that there is a commonality in alternative conceptualizations in different programming language contexts.

Table 1: Average Scores on the FCS1 Assessment by Quartile. Reproduced from [63] with permission.

Quartile	n	Mean Δ	σ
4	203	2.31	3.649
3	231	3.76	3.313
2	226	4.81	3.495
1	190	6.20	3.671

The argument for the validity of the FCS1 had three parts [64]. First, a panel of experts reviewed the content to reach an agreement that the concepts addressed were part of a CS1 course. Second, she gave all CS1 student participants two exams: one in whatever language they studied (Java, MATLAB, or Python), and one in pseudocode. The problems were written to be isomorphic, and the exams were given some time apart and counter-balanced to control for ordering effects. She looked for a correlation between the pseudocode assessment and the "native" language assessment, to validate that the pseudocode measured the same concepts as in the studied language. Finally, she collected the final grades from the CS1 students to validate that the pseudocode assessment was testing what the CS1 instructors valued.

2.1 The Use of FCS1

The main use of FCS1 was in Elliott Tew's dissertation [63]. She found that performance on the FCS1 did differ between the three CS1 courses at Georgia Tech. She also found that there was a cost to the use of pseudocode. The students who scored the best on the pseudocode assessment had the closest match in scores between the pseudocode assessment and the language-specific assessment — averaging a difference in 2.31 answers out of the 27 questions on the exams (see Table 1). But the average difference increases dramatically. For the bottom two quartiles, the difference is 17% (4.8 questions out of 27) and 22% (6.2 questions out of 27). It's not too difficult for students in the best-performing quartile to transfer their knowledge to pseudocode, but it's a significant challenge for the lowest-performing two quartiles.

The FCS1 was also used in the 2013 ITiCSE Working Group [69] that revisited the influential 2001 McCracken Working Group report [41]. The 2001 McCracken Working Group report was among the first studies to show that problems with learning programming in CS1 were a generalized phenomenon, seen in multiple institutions in multiple countries [41]. The 2013 study replicated the original finding and used the FCS1 as a measure of conceptual knowledge. Scores on the FCS1 were strongly positively correlated with the ability to program successfully [69].

There were other unpublished uses of the FCS1 examining the effectiveness of learning interventions. Studies comparing teaching strategies at the University of British Columbia in the context of the Carl Wieman Science Education Initiative (CWSEI) used FCS1 to establish a baseline and make comparisons. The goal of the CWSEI project was to improve instruction and student learning in the specific, local context. Since these findings would not generalize to a broader context, the results remain unpublished.

Due to concerns about the FCS1 becoming invalidated through broad dissemination (e.g., by being posted publicly on the Internet), there were few direct uses of the FCS1 by students. However, the FCS1 was successfully used as a model for creating other assessments. For example, Lee and Ko [38] created a pseudocode assessment to measure learning in different online learning contexts, from games to tutorials. They used the FCS1's examples, descriptions, and two-page pseudo-code guide in generating their exam. They validated their replicated instrument using Amazon Mechanical Turk workers and then used their instrument to compare several different settings.

We do not include a further analysis of the literature since many of the citations to any of the FCS1 papers [63–65] reference it as an example of a validated measure of knowledge for CS1. Rather than a systematic search of the literature, we have included here all legal uses of the FCS1. These are all the studies of the FCS1 whose authors requested access through the creators of the FCS1.

3 DEVELOPMENT AND USE OF SCS1

To protect the FCS1 and still have an instrument available to the community, the SCS1 was created as a replication of the FCS1 which could be distributed without potentially compromising the FCS1 [43]. The SCS1 had a simpler replication and validation process. Rather than compare against isomorphic tests in each of Java, MATLAB, and Python, the SCS1 is a pseudocode-based test that was only validated against the FCS1. Near the end of the introductory computing courses at one institution, 183 students took the SCS1 and the FCS1, one week apart and counter-balanced for order effects. The correlation between the SCS1 and FCS1 was high and sufficient to establish an argument for validity. However, because the SCS1 was created by writing isomorphic questions to the FCS1 to maintain FCS1's validity argument, that meant little was changed to alter or improve on the FCS1. As a result, the previous difficulty of the FCS1 assessment, as presented in [63], was transferred to the SCS1. The difficulty and discrimination of each SCS1 item can be seen in Table 2, where 22 items were identified as "hard" and no items were identified as "easy."

3.1 The Use of SCS1

In the original paper detailing the replication and validation of the SCS1 assessment, Parker et al. encouraged free and open use of their assessment. Since then, the SCS1 assessment has been used in many research studies in a myriad of ways. To understand the use and evolution of the SCS1 assessment, we conducted a search on Google Scholar in early January 2021 of papers that cite the SCS1 publication [43]. Our search returned 70 papers, which we then read and recorded how they used the SCS1 or cited the original publication. Our approach does bias our understanding of how the SCS1 is being used since we only examined published work or dissertations. Publication bias is a persistent problem in any review of literature [18]. We also recognize the constraints of using a research database at a given point in time. For example, because of the way Google Scholar indexes papers, some papers that were not technically published yet (e.g. SIGCSE 2021) were included in our analysis if pre-prints of the articles were posted on authors' website and were indexed by Google Scholar. Papers were removed from our analysis if they only briefly mention the SCS1. This includes 13 papers that cite SCS1 as an example of an assessment in CS with an argument for validity; six papers that reference SCS1 in their related works section, generally as an example of work being done in CS education; two papers that use the SCS1 paper to support arguments for reliability and validity of assessments; and two papers that use parts of the SCS1 paper, but not the assessment itself (e.g., the scatterplot was reproduced in [24], and the think-aloud methods were used in [29]). Additionally, three papers used the SCS1 assessment to create a new instrument, but the process was not described in enough detail to include in our discussion. Below, we detail some of the different ways that the SCS1 assessment has been used in the computing education community. A summary of these findings can be found in Table 3.

3.1.1 To Learn About Learning. The creators of the SCS1 assessment wrote that they welcomed the research community to use the SCS1 assessment to measure performance or learning gains [43]. Many studies have been published since then using the SCS1 assessment in exactly this manner [5, 31–33, 42, 75, 76]. Here we detail these studies and the findings that the SCS1 assessment has been used to support.

One area of study that often uses the SCS1 assessment is the topic of program tracing. In one study, the SCS1 assessment was used to show support for a comprehension-first and theoretically-informed pedagogy for program tracing via an interactive tutorial called PLTutor [42]. In this study, undergraduate students were recruited as they began a CS1 course in Java. Additionally, some questions from the SCS1 assessment were used in a pre- and post-test in a study that showed that teaching students a program tracing strategy can lead to higher student scores on program tracing problems and the course midterm [76]. 24 students were recruited who had completed one or fewer CS courses, all of whom were undergraduate students except for one who had a Master's degree.

The SCS1 assessment has also been used to investigate online learning environments and tools [31–33, 75]. The SCS1 assessment was used to show that online undergraduate students can attain comparable learning outcomes to students in traditional, in-person learning settings [32]. In fact, of students who reported putting forth their best effort on taking the SCS1 assessment, students in the online CS1 course scored statistically significantly higher on the SCS1 as a post-test than students who were in the traditional section [32]. These findings were replicated in a later study with new students [33]. Furthermore, these findings continue to hold when the online CS1 course is supported by an AI for feedback and evaluation [31]. Questions from the SCS1 assessment were also used to investigate a self-directed online Python learning tool. In a study of 79 novice programmers (defined as having had at most non-Python programming course and never learned or used Python), findings supported the claim that more agency and information can lead to higher student motivation, but not necessarily improved student learning [75].

Table 2: Item response theory classifications of SCS1 questions. Reproduced from [43] with permission.

		Difficulty (0-100%)			
		Hard (0-50%)	Moderate (50-85%)	Easy (85-100%)	Total
Discrimination	Poor (<0.1)	5, 8, 15, 18, 20, 24, 27	–	–	7 items
	Fair (0.1-0.3)	4,6,7,9,10,11,12,13,16, 17,21,22,25,26	23	–	15 items
	Good (>0.3)	14	1, 2, 3, 19	–	5 items
	Total	22 items	5 items	0 items	27 items

Table 3: A summary of the ways the FCS1 and SCS1 have been used and the accompanying papers

To Learn about Learning	FCS1	Programming language differences [63] 2013 ITiCSE Working Group [69]
	SCS1	Program Tracing: [42, 76] Online Learning: [31–33, 75] Comparing Block-based with Text-based: [5]
To Build More Valid Instruments	FCS1	Gidget [38] SCS1 [43]
	SCS1	Self Evaluation Instrument (SEI): [15] Basic Data Structures Inventory (BDSI): [48, 61, 77]
Improving	SCS1	Item Response Theory: [74]
Adapting	SCS1	12 question subset: [45] 9 question subset: [6] (SCS1R)
Translating	SCS1	Java: [17] Python: [37] (mSCS1), [58] MATLAB: [2, 3] (MCS1) German: [66, 67] Finnish: [15]

With the rise of block-based programming environments, questions have been raised about how block-based programming and block-based programming with text-based programming (dual-modality programming) compare to text-based programming. For a study comparing dual-modality programming environments and instruction with text-based approaches to instruction, participant learning was measured via the SCS1 assessment and course exams for 673 undergraduates in a semester-long CS1 course [5]. The study found that students performed better on course exams when they were taught using dual-modality representations and were provided dual-modality tools. However, the scores were not statistically significantly different on the SCS1 assessment, possibly due to the hard questions and lack of free-response options.

3.1.2 *To Build More Computing Learning Instruments.*

Many of the published works that cite the seminal SCS1 paper use it to support their arguments for more assessments [4, 13, 16, 35, 36, 39, 46], citing the line "there are not many valid instruments for CS1" [43]. However, as Margulieux et al. point out, there are multiple computing standardized measurement assessments [40]. Not all of these assessments seek to measure computing knowledge, as some measure self-efficacy, programming attitudes, or cognitive load, to name a few. The claim that there can always be more standardized assessment instruments in computing is perhaps a stronger statement than saying there are "not many" of such instruments. Moving beyond statements of the community needing more standardized assessments, a few groups have embarked on the endeavor to create entirely new assessment instruments with questions from the SCS1 assessment, and the process laid out in the SCS1 publication [43], as inspiration.

The SCS1 assessment is recommended to take an hour to complete and has difficult questions that were identified in the initial validation process. These factors can make the assessment inaccessible to many students and researchers. In an effort to create a more lightweight assessment that is less obtrusive, Duran et al. created the SEI (self-evaluation instrument) to rapidly assess student understanding of basic programming knowledge [15]. Duran et al. used the SCS1 assessment, after translating it into Finnish, to create an argument for validity for the SEI. Recruited from a basic CS course taught in Java and offered as a MOOC, 440 students completed the Finnish SCS1. Students were more likely to volunteer to take the SEI than the SCS1 assessment, and students provided negative feedback for the SCS1 assessment more often than the SEI. However, both the scores on the SEI and the SCS1 assessment correlated with the course exam. The researchers also found that some questions that were difficult for students in other studies and contexts (e.g. [74]) were easy for their students, and other questions that were easy for others were difficult for them, suggesting an avenue for future research on assessments in computing. The prototype SEI can be found at https://goo.gl/nGR9Th.

The Basic Data Structures Inventory (BDSI) was created as a concept inventory to cover topics typically included in a CS2 course, or the course that teaches basic data structures [77]. The SCS1 assessment was not directly used for the creation of the BDSI since the SCS1 was intended for CS1 courses and thus a different set of concepts. However, the pseudocode that the SCS1 assessment is based on was used for the BDSI [77]. Additionally, the BDSI was created following similar steps as the FCS1 and SCS1, including expert panels, short response questions that were then converted to multiple-choice questions, validation interviews, and statistical validation, including classical test theory and reliability [48, 61]. The BDSI is available at https://groups.google.com/forum/#!forum/cs2-bdsi-concept-inventory.

4 ADAPTING AND REVISING A VALIDATED ASSESSMENT

When the SCS1 assessment was released to the community, Parker et al. concluded their paper with a call for more assessments, including revising the SCS1 assessment [43]. Over the past five years, the SCS1 assessment has been investigated, revised, and adapted for broader use. Part of the need for revision is due to the difficulty of the SCS1 assessment, as previously discussed and presented in Table 2. This difficulty level is noted by some researchers in the community as a reason for not using the assessment [28]. As a result, many research teams have sought to improve the SCS1 assessment for wider use by the community.

4.1 Improving the SCS1

Since the SCS1 assessment's release to the community, a research group at the University of Washington has conducted further analysis on the SCS1 assessment to identify specific questions that could be improved. Where the original SCS1 paper included Classical Test Theory, Xie et al. used Item Response Theory to identify problematic questions [74]. Xie et al. used SCS1 responses from 425 pre-CS1 students and 64 pre-CS2 students. Their work can help identify which questions to focus on to improve the SCS1 assessment as a whole. Xie et al. also conclude that the SCS1 assessment may be best used as a post-test and inappropriate to use as a pre-test due to its difficulty and potential floor-effects. This has led some studies to refrain from using the SCS1 assessment due to concerns of its appropriateness as a pre-test administered before a CS learning experience [57, 70]. This is not an unfair categorization of the SCS1, especially given that it wasn't explicitly designed for use as a pre-test. Rather, it was designed as the FCS1 was: to be used to compare different CS1 classrooms with different programming languages. Although the SCS1 is convenient to use as a pre- and post- test because the scoring is on the same scale, making it easy to perceive growth or improvement in CS1 concepts, more work should be done to either alter the SCS1 or create an accompanying pre-test to an SCS1 post-test.

4.2 Shortening the SCS1

The original SCS1 assessment was administered within a 60 minute time period [43]. However, this can be too long for a class period, which might typically be 50 minutes. This, combined with the acknowledgment that it is a difficult assessment, can reduce the viability of the SCS1 assessment as a research tool. As such, there have been various efforts to shorten the SCS1 assessment [6, 17, 45].

Parker and Guzdial, the original creators of the SCS1 assessment, modified the SCS1 themselves, parsing it down to a 12 question test from the original 27 questions [45]. The questions were chosen based on their original difficulty and discrimination values published [43]. The original format of the assessment was maintained in terms of having three types of questions (definitional, tracing, and code completion), which were for each of four content areas (if statement, for loops, while loops, and logical operators). This effort was undertaken due to the adapted SCS1 assessment only being one piece of their study, where they asked undergraduate students near the end of their first undergraduate computing course to also take an access survey, spatial reasoning test, and socioeconomic status questionnaire [45]. Findings from their study showed a connection between socioeconomic status and CS achievement (as measured by the adapted SCS1 assessment), which was mediated by spatial ability and not by access to computing.

A research group out of the University of Nebraska, Lincoln has also published work that used a shortened version of the SCS1 assessment [6–8]. The revised SCS1 assessment (termed 'SCS1R') was created by keeping the original content of the assessment rather than the format of question types [6]. One question was selected for each of the nine concepts to try to keep the test to a maximum of 20 minutes. To choose which question to represent a content area, the authors say they 'selected the one question we judged to be the best fit for our study' [6]. Bockmon et al. recruited 635 undergraduate students in introductory computing courses to pilot the SCS1R. The research team used SCS1R to further link spatial skills and CS, such as how training in spatial skills can improve CS performance and enjoyment (n=197 undergraduate CS1 students) [8] and connecting spatial skills with programming ability (274 control-group participants, 71 experimental-group participants, all from an undergraduate CS1 course) [7].

4.3 Translating the SCS1

One critical design element of the FCS1 and SCS1 assessments was the pseudocode language they were written in. Because the assessments were designed to be used across courses using different programming languages, the assessements could not be written in one particular programming language [63], as discussed earlier. Despite the goal of creating an assessment that could be used by a multitude of students learning different languages, many research teams have chosen to replicate the SCS1 assessment into specific programming languages [2, 3, 17, 37, 58]. In one instance, the motivation for moving away from the pseudocode-based assessment was that the pseudocode was non-intuitive and concerns were raised about burdening students with interpreting a pseudolanguage, which was discussed as a concern with the results of the FCS1 validation studies [37]. In this case, the SCS1 assessment was translated into Python, in addition to being shortened for timing concerns and content alignment, and administered to 44 undergraduate students at the start of a Python-based CS2 course. Layman et al. found a connection between their version of the SCS1 (mSCS1; modified SCS1) and CS2 performance [37]. Other research teams have taken a similar approach of translating a subset of the SCS1 assessment into Python, though also adding 127 short, deliberate practice questions, for use in a pre-CS2 assessment [58]. Another case chose to translate some questions of the SCS1 questions into Java for use with 42 students in a CS1 course taught in Java [17]. Duvall et al. cited that the pseudocode language was too similar to Python and their students had difficulty understanding the questions. In another instance, the SCS1 assessment was translated into MATLAB to provide a MATLAB-specific concept inventory (MCS1; MATLAB Computer Science 1 Assessment) [2, 3]. The MCS1 was piloted during think-aloud interviews with six first-year engineering students.

The SCS1 assessment is a reasonable starting point for developing language-specific tools given the arguments for validity that support it. The creation of language-specific assessments should also include language-specific bugs and conceptions [60]. Weintrop and Wilensky found with their commutative assessment that blocks and text programming led to different bugs and misconceptions [73]. Elliott Tew found some of these in her validation of the FCS1 but explicitly designed the FCS1 around them, focusing on the commonalities rather than the language-specific issues [64]. A language-specific tool could start from SCS1 for validation but would have to develop different distractors that highlight the language-specific learning challenges.

Beyond translating the SCS1 assessment into different programming languages, there have also been efforts to translate the SCS1 into written languages other than English. A research team in Germany detail the steps they took to translate the SCS1 to German [66, 67]. Timmerman et al. then used the German translation of the SCS1 to monitor student learning in a CS1 course as the course was revised. A team in Finland followed Timmerman et al.'s steps and translated the SCS1 into Finnish [15]. This was then used to create the SEI, described in Section 3.1.2. There may be even more written language translations of the SCS1 out there given the use of SCS1 across the globe, but these are the only published instances of such an occurrence as of the time we conducted our Google Scholar search.

5 COMPARING ASSESSMENT USE IN COMPUTING EDUCATION TO PHYSICS EDUCATION

The Force Concept Inventory (FCI) had a dramatic impact on physics education and far beyond. Here, we discuss key papers in the development and use of the FCI in order to compare and contrast to the development and use of the FCS1 and SCS1. We did not conduct a systemic literature review of the FCI but instead consider one of the original goals for the SCS1, which was to conduct a similar analysis as Hake did.

The FCI was a validated instrument that focused solely on concepts of force from an introductory physics perspective [27]. It was noteworthy for building on the knowledge of students' alternative conceptions of force. Elliott Tew had to develop a process for identifying good distractors for her FCS1 instrument. Hestenes and colleagues were able to draw on decades of research that told them how students explained force to themselves incorrectly. The wrong answers on the FCI were known to be common among students.

Directly, the FCI led to a significant finding in the development of the argument for evidence-based research methods. Hake's 1998 paper drew on over 6,000 test-takers to argue that active-learning techniques ("interactive learning" was the term used at the time) led to better learning outcomes [25]. Hake had student responses to the FCI collected both before and after courses (pre and post) in a central repository, along with data about the courses in which the test-takers were enrolled[1]. Using this central repository, he was able to make a broad statement that influenced physics education, and also make a broader argument about active learning techniques in STEM classes. Hake had data on each use of the FCI, and he reported separately results from high schools, colleges, and universities.

Indirectly, the FCI influenced the development of novel active-learning techniques such as *peer instruction.* Eric Mazur famously gave the FCI to his students at Harvard, expecting them to do very well on it. When his students performed worse than he expected, he realized that he needed to improve his own teaching – a validated instrument had demonstrated what his students were actually learning [9]. Mazur invented peer instruction as a way to improve his student's performance on the FCI [10]. Peer instruction has now grown far beyond physics and is one of the most successful active learning techniques in computer science as well [47].

When the validation of the SCS1 assessment was initially published, the authors wrote a vision on "Following Hake's Lead." Parker et al. proposed that their assessment could be used for a large-scale study to determine the effectiveness of learning interventions. However, though the SCS1 assessment has been administered many times, a 'Hake' study is unlikely to develop any time soon, which is our answer to RQ2. Part of this is due to the context in which SCS1 is given. The SCS1 has been used in a variety of learning environments to answer a myriad of research questions. Though all these data points could be collapsed to analyze the assessment at a large scale, the large range of variables that are confounded with each data point would make any claims drawn from such an analysis suspect.

In the end, the SCS1 creators chose *not* to follow Hake's lead, and that decision may say something deep and interesting about the difference between computing education research and physics education research. The work by Hestenes, Hake, Mazur, and others around the FCI took place in higher-education introductory physics courses in the United States. When considering the range of people between their teens and twenties just in the United States, those in higher education Physics classes is a narrow subset. Further, FCI (as the name suggests) is only about *force.* It doesn't aim to cover all of an introductory physics course. It's about one specific topic in introductory physics, and it's a topic that is widely considered fundamental. In contrast, SCS1 has already been used in a much broader range of classes and contexts than was the FCI as reflected in Hake's study. The variation in SCS1 is too large to ignore, and there is too little use of SCS1 in directly comparable contexts to make a Hake-like analysis possible.

[1]From request for FCI access "Instructor's name and institution, number of students enrolled, pre/post test scores, standard deviations where available, and normalized gains" and then via follow-up survey, "pre/post testing method; statistical results; institution; type of students; activities of the students; and the instructor's educational experience, outlook, beliefs, orientation, resources, and teaching methods" [25]

6 DISCUSSION: COMMUNITY NEEDS AND THE FUTURE OF ASSESSMENT IN COMPUTING EDUCATION

The arguments for computing education vary around the world. States in the United States (often associated with the "CS for All" movement) tend to make a vocational argument [71]. There is a shortage of labor in computing, computer science, and software development. The argument is to teach everyone about computer science in order to fill those jobs. The "Computing at School" movement (CAS) in the United Kingdom makes an argument about computing as a fundamental field, like physics and mathematics [30]. In the same way that everyone should have some understanding of the science of the physical world (e.g., physics, chemistry, and biology), the CAS movement argues that students should also learn about the science of the digital world.

If the goals vary, then we might expect the instruction to do so as well. A course that prepares students for the workforce may well be different from a course that teaches introductory CS as a science. In the United States, this distinction might be seen comparing the Advanced Placement exam and course for "CS A" (APCSA) which aims to prepare students for CS1 in a CS major undergraduate curriculum and the exam and course for "CS Principles" (APCSP) which aims to be a broad introduction to the field of computer science [1]. Teaching for different learning objectives will likely vary.

If the courses vary significantly, then so might we expect the student audience and who succeeds at the course. We see this in the distinction between APCSA and APCSP. APCSP attracts more diverse students than APCSA, and APCSP students have different outcomes [54]. Different courses with different learning objectives will likely attract different students and will certainly require different assessments. Not only are different assessments required because of the course material, but to promote equity among student populations. Differential Item Functioning methods have not been used with either the FCS1 or SCS1, but should be to understand how different student groups perform on the assessments [11]. Constructing courses and curricula to promote equity will only work if the assessments for measuring learning and achievement have also been updated to support equity goals.

To answer our first research question (RQ1), about how use of FCS1 and SCS1 matched the arguments for validity used in their development, we find that FCS1 and SCS1 have been used to assess introductory computer science in general and, more specifically, to assess introduction to programming courses. Those assessment goals are broader than the validation arguments for FCS1 and SCS1. Both instruments only ask questions about programming concepts, because that was the common denominator that Elliott Tew found when she looked across CS1 textbooks and standards [64]. FCS1 and SCS1 were only validated with undergraduate CS students taking in-person (face to face) classes in residential college settings. While several institutions and courses participated in the validation studies, all of the courses addressed a narrow segment of the population in a rarefied context.

In most published studies, the FCS1 and SCS1 are being used in residential collegiate settings around a CS1 course taught in Python or Java – thus, our answer to RQ1 is "mostly." However, the SCS1 is also being used with audiences and contexts for which there is no argument for validity [34]. This was true even when the SCS1 was published and use cases were presented, including using the SCS1 with teachers [43]. We also noticed the SCS1 being used in the published works in online settings or in CS2, settings which are different than the one in which SCS1 was validated. We only review published papers. The SCS1 and FCS1 might be used with pre-service teachers, with K-12 students, with languages other than those three, with introductory courses whose goals (and instruction) are different than CS1, or in modality contexts other than in-person classes in residential colleges. The FCS1 and SCS1 were developed as multi-lingual assessments, but the languages chosen were relatively similar compared to languages for functional programming or declarative logic programming. We cannot make any claim of validity beyond these imperative languages. While researchers might use FCS1 and SCS1 somewhere other than CS1 in a residential college setting with these three programming languages, the validity argument has to be made anew. The whole point of a validated assessment is to establish a commonly accepted measure. If there is no argument that it is reasonable to use this measure for a given research context, the community of computing education researchers has no reason to accept the results.

Researchers who use the FCS1 or SCS1 in novel research contexts may be treating "*programming*" as a solitary conceptual topic or unit, like "*force*" in the FCI. It is unlikely that programming is a unitary concept, that "programming" is a single set of concepts or skills that we can assess. We have ample evidence that transfer between programming languages is difficult [55, 68]. How much more complicated might that transfer be if the classes in which we might teach programming have different goals, student audiences, levels, and modalities. Programming means different things in different contexts. Programming in APCSA is different from programming in APCSP.

The FCS1 and SCS1 are still useful to computing education researchers when used in contexts for which they have been validated. They can also be useful as models for other assessments, e.g., performance on the new assessments might be compared to performance on FCS1 and SCS1 as we see in several of the published papers. SCS1 might also be validated for new contexts, as in new languages. But it's pretty clear that computing education researchers might want to assess learning in other areas related to the big concept of programming, and assess other important aspects of their educational experience, from sense of belonging [14, 72] to self-efficacy [50, 59].

A final lesson from the FCI is that FCS1 and SCS1 are at a dramatically different granularity. Force is one part of a physics course. CS1 is an entire course with many parts in it. The computing education research community might be well-served by finer-grained concept inventories. We might focus on programming statements, like `if` and `for`, though past research suggests that those do not work as knowledge units [53]. We might instead focus on conceptual units, such as conditional testing, Boolean expressions, repetition (combining iteration and recursion), mutability, or even just sequential execution, which we do know is a challenge for many learners [52]. We might combine these conceptual units to focus on learning trajectories [20, 44, 51, 52]. Or we might focus on skills, like the ability to trace execution or to debug. Or we might focus on deeper concepts, like the idea that the computer transforms representations (from code to HTML) into execution or Web pages.

Computing education research should aim to reach agreement on validated assessments, and also on what we're assessing. We need to find some agreement on what the core concepts are – whether they are variables, state, or mutability – if we want to be talking about the same things across papers. If we want to have a research dialog about student understanding of a computational concept or skill, we need a common definition, and common validated assessments can form a stronger foundation for our discipline.

ACKNOWLEDGMENTS

We would like to thank the computing education research community for using the SCS1 and sharing their feedback and results. The assessment could not have gone nearly so far without your work.

REFERENCES

[1] Owen Astrachan and Amy Briggs. 2012. The CS Principles Project. *ACM Inroads* 3, 2 (2012), 38–42.

[2] Ada Barach. 2020. *A MATLAB Programming Concept Inventory for Assessing First-Year Engineering Courses: A Replication of the Second Computer Science 1 Assessment.* Ph.D. Dissertation. The Ohio State University.

[3] Ada E. Barach, Connor Jenkins, Serendipity S. Gunawardena, and Krista M. Kecskemety. 2020. MCS1: A MATLAB Programming Concept Inventory for Assessing First-year Engineering Courses. In *2020 ASEE Virtual Annual Conference Content Access*.

[4] Brett A. Becker and Thomas Fitzpatrick. 2019. What Do CS1 Syllabi Reveal About Our Expectations of Introductory Programming Students?. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* 1011–1017.

[5] Jeremiah Blanchard, Christina Gardner-McCune, and Lisa Anthony. 2020. Dual-Modality Instruction and Learning: A Case Study in CS1. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education.* 818–824.

[6] Ryan Bockmon, Stephen Cooper, Jonathan Gratch, and Mohsen Dorodchi. 2019. (Re) Validating Cognitive Introductory Computing Instruments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education.* 552–557.

[7] Ryan Bockmon, Stephen Cooper, Jonathan Gratch, Jian Zhang, and Mohsen Dorodchi. 2020. Can Students' Spatial Skills Predict Their Programming Abilities?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.* 446–451.

[8] Ryan Bockmon, Stephen Cooper, William Koperski, Jonathan Gratch, Sheryl Sorby, and Mohsen Dorodchi. 2020. A CS1 Spatial Skills Intervention and the Impact on Introductory Programming Abilities. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education.* 766–772.

[9] Catherine H. Crouch and Eric Mazur. 2001. Peer Instruction: Ten Years of Experience and Results. *American Journal of Physics* 69, 9 (2001), 970–977. https://doi.org/10.1119/1.1374249 arXiv:https://doi.org/10.1119/1.1374249

[10] Catherine H. Crouch, Jessica Watkins, Adam P. Fagen, and Eric Mazur. 2007. Peer Instruction: Engaging Students One-on-One, All at Once. *Research-based reform of university physics* 1, 1 (2007), 40–95.

[11] Matt J. Davidson, Brett Wortzman, Amy J. Ko, and Min Li. 2021. Investigating Item Bias in a CS1 Exam With Differential Item Functioning. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education.* 1142–1148.

[12] Adrienne Decker. 2007. *How Students Measure Up: An Assessment Instrument for Introductory Computer Science.* Ph.D. Dissertation. State University of New York at Buffalo.

[13] Ömer Demir and Süleyman Sadi Seferoğlu. 2019. Developing a Scratch-Based Coding Achievement Test. *Information and Learning Sciences* (2019).

[14] Brian Dorn and Allison Elliott Tew. 2015. Empirical Validation and Application of the Computing Attitudes Survey. *Computer Science Education* 25, 1 (2015), 1–36.

[15] Rodrigo Duran, Jan-Mikael Rybicki, Juha Sorva, and Arto Hellas. 2019. Exploring the Value of Student Self-Evaluation in Introductory Programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research.* 121–130.

[16] Rodrigo Silva Duran, Jan-Mikael Rybicki, Arto Hellas, and Sanna Suoranta. 2019. Towards a Common Instrument for Measuring Prior Programming Knowledge. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education.* 443–449.

[17] Shannon Duvall, Scott Spurlock, Dugald Hutchings, and Robert Duvall. 2021. Improving Content Learning and Student Perceptions in CS1 with Scrumage. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education.*

[18] Phillipa J. Easterbrook, Ramana Gopalan, J.A. Berlin, and David R. Matthews. 1991. Publication Bias in Clinical Research. *The Lancet* 337, 8746 (1991), 867–872.

[19] Andrea Forte and Mark Guzdial. 2005. Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences for Introductory Courses. *IEEE Transactions on Education* 48, 2 (2005), 248–253.

[20] Brian D. Gane, Maya Israel, Noor Elagha, Wei Yan, Feiya Luo, and James W Pellegrino. 2021. Design and Validation of Learning Trajectory-Based Assessments for Computational Thinking in Upper Elementary Grades. *Computer Science Education* (2021), 1–28.

[21] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2008. Identifying Important and Difficult Concepts in Introductory Computing Courses Using a Delphi Process. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education.* 256–260.

[22] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2010. Setting the Scope of Concept Inventories for Introductory Computing Subjects. *ACM Transactions on Computing Education (TOCE)* 10, 2 (2010), 1–29.

[23] Mark Guzdial. 2009. Teaching Computing to Everyone. *Commun. ACM* 52, 5 (2009), 31–33.

[24] Patricia Haden. 2019. Quantitative Methods. In *The Cambridge Handbook of Computing Education Research*, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, 102–132.

[25] Richard R. Hake. [n.d.]. Interactive-Engagement Versus Traditional Methods: A Six-Thousand-Student Survey of Mechanics Test Data for Introductory Physics Courses. *American journal of Physics* 66, 1 ([n. d.]).

[26] Geoffrey L. Herman, Michael C. Loui, and Craig Zilles. 2010. Creating the Digital Logic Concept Inventory. In *Proceedings of the 41st ACM technical symposium on Computer science education.* 102–106.

[27] David Hestenes, Malcolm Wells, and Gregg Swackhamer. 1992. Force Concept Inventory. *The physics teacher* 30, 3 (1992), 141–158.

[28] Franc Jakoš and Domen Verber. 2017. Learning Basic Programing Skills With Educational Games: A Case of Primary Schools in Slovenia. *Journal of Educational Computing Research* 55, 5 (2017), 673–698.

[29] Brittany Itelia Johnson. 2017. *A Tool (Mis) Communication Theory and Adaptive Approach for Supporting Developer Tool Use.* Ph.D. Dissertation. North Carolina State University.

[30] Simon Peyton Jones, Tim Bell, Quintin Cutts, Sridhar Iyer, Carsten Schulte, Jan Vahrenhold, and ByoungRae Han. 2011. Computing at School. *International comparisons. Retrieved May* 7 (2011), 2013.

[31] David Joyner. 2018. Intelligent Evaluation and Feedback in Support of a Credit-Bearing MOOC. In *International Conference on Artificial Intelligence in Education.* Springer, 166–170.

[32] David Joyner. 2018. Toward CS1 at Scale: Building and Testing a MOOC-for-Credit Candidate. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale.* 1–10.

[33] David Joyner and Melinda McDaniel. 2019. Replicating and Unraveling Performance and Behavioral Differences between an Online and a Traditional CS Course. In *Proceedings of the ACM Conference on Global Computing Education.* 157–163.

[34] Michael T. Kane. 1992. An Argument-Based Approach to Validity. *Psychological bulletin* 112, 3 (1992), 527.

[35] Amy J. Ko and Sally A. Fincher. 2019. A Study Design Process. In *The Cambridge Handbook of Computing Education Research*, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, 81–101.

[36] Thomas Lancaster, Anthony Robins, and Sally Fincher. 2019. Assessment and Plagiarism. In *The Cambridge Handbook of Computing Education Research*, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press.

[37] Lucas Layman, Yang Song, and Curry Guinn. 2020. Toward Predicting Success and Failure in CS2: A Mixed-Method Analysis. In *Proceedings of the 2020 ACM Southeast Conference.* 218–225.

[38] Michael J. Lee and Amy J. Ko. 2015. Comparing the Effectiveness of Online Learning Approaches on CS1 Learning Outcomes. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) *(ICER '15)*. Association for Computing Machinery, New York, NY, USA, 237–246. https://doi.org/10.1145/2787622.2787709

[39] Linda Mannila, Fredrik Heintz, Susanne Kjällander, and Anna Åkerfeldt. 2020. Programming in Primary Education: Towards a Research Based Assessment Framework. In *Proceedings of the 15th Workshop on Primary and Secondary Computing Education.* 1–10.

[40] Lauren Margulieux, Tuba Ayer Ketenci, and Adrienne Decker. 2019. Review of Measurements Used in Computing Education Research and Suggestions for Increasing Standardization. *Computer Science Education* 29, 1 (2019), 49–78.

[41] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 2001. A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (Canterbury,

UK) *(ITiCSE-WGR '01)*. Association for Computing Machinery, New York, NY, USA, 125–180. https://doi.org/10.1145/572133.572137

[42] Greg L. Nelson, Benjamin Xie, and Amy J. Ko. 2017. Comprehension First: Evaluating a Novel Pedagogy and Tutoring System for Program Tracing in CS1. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 2–11.

[43] Miranda C. Parker, Mark Guzdial, and Shelly Engleman. 2016. Replication, Validation, and Use of a Language Independent CS1 Knowledge Assessment. In *Proceedings of the 2016 ACM conference on international computing education research*. 93–101.

[44] Miranda C. Parker, Yvonne S. Kao, Dana Saito-Stehberger, Diana Franklin, Susan Krause, Debra Richardson, and Mark Warschauer. 2021. Development and Preliminary Validation of the Assessment of Computing for Elementary Students (ACES). In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 10–16.

[45] Miranda C. Parker, Amber Solomon, Brianna Pritchett, David A. Illingworth, Lauren E. Marguilieux, and Mark Guzdial. 2018. Socioeconomic Status and Computer Science Achievement: Spatial Ability as a Mediating Variable in a Novel Model of Understanding. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. 97–105.

[46] Markeya S. Peteranetz, Patrick M Morrow, and Leen-Kiat Soh. 2020. Development and Validation of the Computational Thinking Concepts and Skills Test. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 926–932.

[47] Leo Porter, Dennis Bouvier, Quintin Cutts, Scott Grissom, Cynthia Lee, Robert McCartney, Daniel Zingaro, and Beth Simon. 2016. A Multi-Institutional Study of Peer Instruction in Introductory Computing. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 358–363.

[48] Leo Porter, Daniel Zingaro, Soohyun Nam Liao, Cynthia Taylor, Kevin C. Webb, Cynthia Lee, and Michael Clancy. 2019. BDSI: A Validated Concept Inventory for Basic Data Structures. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*. 111–119.

[49] Keith Quille and Susan Bergin. 2019. CS1: How Will They Do? How Can We Help? A Decade of Research and Practice. *Computer Science Education* 29, 2-3 (2019), 254–282.

[50] Vennila Ramalingam and Susan Wiedenbeck. 1998. Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy. *Journal of Educational Computing Research* 19, 4 (1998), 367–381.

[51] Kathryn M. Rich, T. Andrew Binkowski, Carla Strickland, and Diana Franklin. 2018. Decomposition: A K-8 Computational Thinking Learning Trajectory. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. 124–132.

[52] Kathryn M. Rich, Carla Strickland, T. Andrew Binkowski, Cheryl Moran, and Diana Franklin. 2017. K-8 Learning Trajectories Derived from Research Literature: Sequence, Repetition, Conditionals. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. ACM, New York, NY, USA, 182–190. https://doi.org/10.1145/3105726.3106166

[53] Kelly Rivers, Erik Harpstead, and Kenneth R Koedinger. 2016. Learning Curve Analysis for Programming: Which Concepts Do Students Struggle With?. In *ICER*, Vol. 16. 143–151.

[54] Linda J. Sax, Kaitlin N.S. Newhouse, Joanna Goode, Max Skorodinsky, Tomoko M. Nakajima, and Michelle Sendowski. 2020. Does AP CS Principles Broaden Participation in Computing? An Analysis of APCSA and APCSP Participants. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 542–548.

[55] Jean Scholtz and Susan Wiedenbeck. 1990. Learning Second and Subsequent Programming Languages: A Problem of Transfer. *International Journal of Human-Computer Interaction* 2, 1 (1990), 51–72.

[56] Herbert A. Simon. 2019. *The Sciences of the Artificial*. MIT press.

[57] David H. Smith IV, Qiang Hao, Filip Jagodzinski, Yan Liu, and Vishal Gupta. 2019. Quantifying the Effects of Prior Knowledge in Entry-Level Programming Courses. In *Proceedings of the ACM Conference on Global Computing Education*. 30–36.

[58] Yang Song, Yunkai Xiao, Jonathan Stephens, Emma Ruesch, Sean Roginski, and Lucas Layman. 2020. Suitability of SCS1 as a Pre-CS2 Assessment Instrument: A Comparison with Short Deliberate-practice Questions. In *Proceedings of the 2020 ACM Southeast Conference*. 313–314.

[59] Phil Steinhorst, Andrew Petersen, and Jan Vahrenhold. 2020. Revisiting Self-Efficacy in Introductory Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. 158–169.

[60] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. 2018. Programming Misconceptions for School Students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. 151–159.

[61] 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 *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 372–378.

[62] Cynthia Taylor, Daniel Zingaro, Leo Porter, Kevin C. Webb, Cynthia Bailey Lee, and Michael Clancy. 2014. Computer Science Concept Inventories: Past and Future. *Computer Science Education* 24, 4 (2014), 253–276.

[63] Allison Elliott Tew. 2010. *Assessing Fundamental Introductory Computing Concept Knowledge in a Language Independent Manner*. Ph.D. Dissertation. College of Computing, Georgia Institute of Technology.

[64] Allison Elliott Tew and Mark Guzdial. 2010. Developing a Validated Assessment of Fundamental CS1 Concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 97–101. https://doi.org/10.1145/1734263.1734297

[65] Allison Elliott Tew and Mark Guzdial. 2011. The FCS1: A Language Independent Assessment of CS1 Knowledge. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. 111–116.

[66] Dion Timmermann and Christian Kautz. 2016. Design of Open Educational Resources for a Programming Course With a Focus on Conceptual Understanding. In *Proceedings of the 44th SEFI Annual Conference*.

[67] Dion Timmermann, Christian Kautz, and Volker Skwarek. 2016. Evidence-Based Re-Design of an Introductory Course "Programming in C". In *2016 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–5.

[68] Ethel Tshukudu and Quintin Cutts. 2020. Understanding Conceptual Transfer for Students Learning New Programming Languages. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*. 227–237.

[69] Ian Utting, Allison Elliott Tew, Mike McCracken, Lynda Thomas, Dennis Bouvier, Roger Frye, James Paterson, Michael Caspersen, Yifat Ben-David Kolikant, Juha Sorva, et al. 2013. A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations. In *Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports*. 15–32.

[70] Kristina Von Hausswolff and Anna Eckerdal. 2018. Measuring Programming Knowledge in a Research Context. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.

[71] Jennifer Wang. 2017. Is the US Education System Ready for CS for All? *Commun. ACM* 60, 8 (2017), 26–28.

[72] Alicia N. Washington, Shaefny Grays, and Sudhipta Dasmohapatra. 2016. The Computer Science Attitude and Identity Survey (CSAIS): A Novel Tool for Measuring the Impact of Ethnic Identity in Underrepresented Computer Science Students. In *American Society of Engineering Education 123rd Annual Conference and Exposition*.

[73] David Weintrop and Uri Wilensky. 2015. Using Commutative Assessments To Compare Conceptual Understanding in Blocks-Based and Text-Based Programs. In *ICER*, Vol. 15. 101–110.

[74] Benjamin Xie, Matthew J. Davidson, Min Li, and Amy J. Ko. 2019. An Item Response Theory Evaluation of a Language-Independent CS1 Knowledge Assessment. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 699–705.

[75] Benjamin Xie, Greg L. Nelson, Harshitha Akkaraju, William Kwok, and Amy J. Ko. 2020. The Effect of Informing Agency in Self-Directed Online Learning Environments. In *Proceedings of the Seventh ACM Conference on Learning@ Scale*. 77–89.

[76] Benjamin Xie, Greg L. Nelson, and Amy J. Ko. 2018. An Explicit Strategy to Scaffold Novice Program Tracing. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. 344–349.

[77] Daniel Zingaro, Cynthia Taylor, Leo Porter, Michael Clancy, Cynthia Lee, Soohyun Nam Liao, and Kevin C. Webb. 2018. Identifying Student Difficulties With Basic Data Structures. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*. 169–177.