Computer Science Education
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/ncse20

Computer science concept inventories: past and future

C. Taylor[a], D. Zingaro[b], L. Porter[c], K.C. Webb[d], C.B. Lee[e] & M. Clancy[f]

[a] Department of Computer Science, Oberlin College, Oberlin, OH, USA.

[b] Department of Math and Computer Sciences, University of Toronto, ON, Canada.

[c] Computer Science and Eng. Dept., University of California, San Diego, CA, USA.

[d] Department of Computer Science, Swarthmore College, Swarthmore, PA, USA.

[e] Department of Computer Science, Stanford University, Stanford, CA, USA.

[f] EECS Computer Science Division, University of California, Berkeley, CA, USA.

Published online: 20 Oct 2014.

Computer Science Education, 2014
http://dx.doi.org/10.1080/08993408.2014.970779

Computer science concept inventories: past and future

C. Taylor[a]*, D. Zingaro[b], L. Porter[c], K.C. Webb[d], C.B. Lee[e] and M. Clancy[f]

[a]*Department of Computer Science, Oberlin College, Oberlin, OH, USA;* [b]*Department of Math and Computer Sciences, University of Toronto, ON, Canada;* [c]*Computer Science and Eng. Dept., University of California, San Diego, CA, USA;* [d]*Department of Computer Science, Swarthmore College, Swarthmore, PA, USA;* [e]*Department of Computer Science, Stanford University, Stanford, CA, USA;* [f]*EECS Computer Science Division, University of California, Berkeley, CA, USA*

(*Received 9 May 2014; accepted 4 September 2014*)

Concept Inventories (CIs) are assessments designed to measure student learning of core concepts. CIs have become well known for their major impact on pedagogical techniques in other sciences, especially physics. Presently, there are no widely used, validated CIs for computer science. However, considerable groundwork has been performed in the form of identifying core concepts, analyzing student misconceptions, and developing CI assessment questions. Although much of the work has been focused on CS1 and a CI has been developed for digital logic, some preliminary work on CIs is underway for other courses. This literature review examines CI work in other STEM disciplines, discusses the preliminary development of CIs in computer science, and outlines related research in computer science education that contributes to CI development.

Keywords: concept inventory; assessment; misconceptions

1. Introduction

A *concept inventory* (CI) is a standardized assessment tool designed to measure student understanding of the core concepts of a topic (Goldman et al., 2010), or the extent to which instruction has helped students achieve expert-level thinking in a domain (Adams & Wieman, 2011; Herman, Loui, & Zilles, 2010a; Hestenes, Wells, & Swackhamer, 1992). Critically, CIs provide a mechanism for educators to compare student learning outcomes across instructors, institutions, curricula, and pedagogical practices.

The exclusive focus of CIs on core concepts differentiates them from other metrics of student learning such as final exams, which frequently ask students to perform detailed calculations or to regurgitate memorized details. By focusing on a topic's central concepts, CIs achieve broad applicability as standards for pedagogical comparison. Their purpose is

*Corresponding author. Email: ctaylor@oberlin.edu to improve pedagogy rather than summatively assess students (Adams & Wieman, 2011). CIs are sometimes administered as both pre- and post-course assessments, so as to control for initial knowledge and provide a clear before-and-after view of student learning.

A broadly applicable CI can catalyze innovation and improvement in teaching and learning by providing meaningful feedback to instructors and researchers on the effectiveness of their interventions, course design, pedagogy, and other teaching-related factors. Improvements brought by CIs typically benefit all students, sometimes with a further beneficial boost to subgroups such as women and minorities (Reay, Li, & Bao, 2008). CIs also encourage the raising of standards across campuses, as more direct comparisons between institutions are made possible and programs with lower performance can be identified.

Toward these objectives, CIs have been successfully developed and used in other STEM disciplines, such as physics, chemistry, and biology, to drive discipline-specific education research and pedagogical reformation. The ultimate result of the pedagogical changes, enabled by research made possible by CIs, has been lauded for improving student outcomes (Crouch & Mazur, 2001).

In computer science, CI development remains in its infancy. CIs have been developed for digital logic (Herman, Loui, & Zilles, 2011) and CS1 (Tew & Guzdial, 2011). Other course assessments for discrete mathematics, computer architecture, and operating systems have had preliminary work performed (Almstrum et al., 2006; Porter, Garcia, Tseng, & Zingaro, 2013; Webb & Taylor, 2014) but are either incomplete, unvalidated, or not publicly available.

Although computer science appears to be distinctly behind other STEM disciplines in the development, deployment, and widespread use of CIs, a great deal of complementary work has been accomplished in a number of areas, including the development of AP CS examinations, the identification of core course concepts and common misconceptions, and potential questions on topics that could be part of a larger full CI (Clancy, 2004; Goldman et al., 2010; Karpierz & Wolfman, 2014; Lister et al., 2004; McCracken et al., 2001; Paul & Vahrenhold, 2013; Soloway, Ehrlich, Bonar, & Greenspan, 1982). Given the groundwork already accomplished by the computer science education research community, our discipline is now poised for more widespread CI development and use.

In this work, we provide a literature review of the following topics:

- An overview of the work that has been done to construct CIs in other STEM disciplines, including a discussion of standard CI development best practices.
- A characterization of the challenges for CI development in computer science, along with a discussion of commonly used assessments and preliminary work toward building CIs for CS topics.

- A review of work in computer science education that contributes to CI development, including the development of standardized assessments (AP CS) and research that identifies core concepts and common student misconceptions.

2. Concept inventories in engineering and other sciences

This section primes the discussion for concept inventories in computer science by reporting the successes of CIs in other disciplines and describing their development best practices. Concept inventories have had a large impact on pedagogy in a variety of scientific fields. In physics, the Force Concept Inventory (Hestenes et al., 1992) tests students on their conceptions of force and motion as covered in the first post-secondary physics course. Deploying the FCI revealed a vast difference between student and expert understanding of core physics concepts (Adams & Wieman, 2011). Perhaps, more significantly, it exposed a vast difference between what instructors thought their students understood, based on conventional final exam performance, and the misconceptions that their students actually held (Mazur, 2009).

It is difficult to overstate the effect of the FCI on the physics education community. Repeated use of the FCI showed instructors that students were not obtaining expert-like conceptions of force, even after significant physics instruction (Crouch & Mazur, 2001). Learning outcomes were lackluster, regardless of instructor or institution, leading physics instructors to rethink their teaching practices.

FCI performance catalyzed the development of new teaching methods, such as peer instruction (PI) and other active-learning pedagogies, which have drastically improved student learning (Crouch & Mazur, 2001). A common metric to measure quantity of learning is normalized gain (NG), which is the ratio of amount of learning to maximum available learning. For example, if students score 40% on a pre-test and 80% on a post-test, then they have learned 40% out of a maximum 60%, so the NG is 0.67. In one study including some 6542 students in 62 different courses, it was found that all traditional courses had low learning gains (defined as $NG \leq 0.3$), compared to 85% of interactive-engagement courses that demonstrated medium gains ($0.3 < NG \leq 0.7$) (Hake, 1998).

The strength of the Hake study – including its influence on faculty and institutional change – is that it amassed a large sample of students from a broad range of institutions and instructors. Without the FCI to provide a common measure, a study of this magnitude would have been impossible. The FCI is a critical piece of infrastructure on which pedagogical experimentation, innovation, and reform have been built.

Today, the FCI is something of a gold standard to which other CIs are compared (Herman, 2011). While the FCI is the most striking example

Table 1. Validated concept inventories in other STEM fields; FC indicates members of the NSF-funded foundation coalition.

Subject area	Developers
Astronomy	Zeilik (2002)
Natural selection	Anderson, Fisher, and Norman (2002)
Dynamics	Gray et al. (2005) (FC)
Signals and systems	Wage, Buck, Wright, and Welch (2005) (FC)
Statics	Steif & Dantzler (2005)
Light and spectroscopy	Bardar, Prather, Brecher, and Slater (2006)
Calculus	Epstein (2007)
Genetics	Smith, Wood, and Knight (2008)
Properties of stars	Bailey (2008)
Astronomy and space science	Sadler et al. (2009)
Geoscience	Libarkin, Ward, Andersln, Kortemeyer, and Raeburn (2011)

of the benefits of CIs on learning, other disciplines have followed suit and have begun to similarly benefit from CI availability. For example, the Signals and Systems Concept Inventory (SSCI) has demonstrated that interactive engagement in signal-processing courses is beneficial over traditional teaching methods, as was similarly found by physics educators (Buck, Wage, Hjalmarson, & Nelson, 2007).

2.1. *Concept inventory topics*

A vast number of CIs have been developed for a variety of sciences. Table 1 lists several validated CIs that are widely used in their fields. There are many more CIs that are in the process of being developed and validated, or currently available but in less widespread use.

The National Science Foundation has supported the *Foundation Coalition* (www.foundationcoalition.org), which, among other activities, has pursued the development of numerous concept inventories in a variety of engineering fields. The fields include circuits, computer engineering, electromagnetics, electronics, signals and systems, waves, dynamics, fluid mechanics, heat transfer, strength of materials, thermodynamics, chemistry, and materials. The projects are in various states of completion.

Especially, relevant are existing CIs on topics related to computer science such as calculus (Epstein, 2007), statistics (Allen, 2006), signals and systems (Wage et al., 2005), and circuits (Helgeland & Rancour, 2003; Ogunfunmi & Rahman, 2010). These CIs could potentially be used as starting points for CIs on similar CS topics. In Section 3, we discuss some challenges that are inherent in developing CIs in CS; investigating the ways in which researchers from other disciplines have addressed related challenges may yield important insights.

2.2. *Concept inventory development process*

Creating and validating a concept inventory can be an arduous task. Adams and Wieman (2011) outline an established procedure to develop and validate a CI. This protocol has been used in the creation of at least nine different CIs in the sciences (Adams & Wieman, 2011). Their protocol is as follows:

(1) *Establish topics*. Determine the topics that are important to faculty members. Techniques include self-reflection and discussion with experienced faculty members or subject experts.
(2) *Identify student thinking*. Observe (e.g. in course help sessions) and interview students to understand how their thinking deviates from expert thinking. Consult experienced teachers and domain misconception literature.
(3) *Create open-ended survey questions*. Administer these questions to entire classes of students in order to further examine issues raised in the interviews.
(4) *Create forced-answer test*. Establish distractors based on actual student responses obtained in the above steps. The test should contain no more than 30 questions.
(5) *Validate test questions through interviews*. Reach consensus among experts that all responses are correct. Ensure that students interpret the questions consistently and that maladaptive student thinking results in incorrect responses.
(6) *Administer and statistically analyze*. Administer the inventory to several large classes, applying statistics to account for reliability (e.g. consistency when administering the test to two equivalent populations) and validity (e.g. correlating with course assessments).

The final three steps are iterative: when validity or reliability concerns are discovered, questions should be adjusted and students/experts again interviewed.

A core aspect of the development process is the generation of distractors. For each multiple-choice question, one response is correct (the key) and the remainder are incorrect (the distractors). Each distractor must be rooted in student misconceptions, so that when a distractor is chosen it says something about the particular misconception leading to that choice. In addition, distractors should have "drawing power": if students hold a particular misconception, they should be drawn toward the associated distractor (Almstrum et al., 2006). In this way, distractors are data to the research team: when a student chooses a distractor, that choice provides useful observational data related to student misconceptions. Distractors can stem from student interviews, responses to open-ended questions, domain experts, textbook materials, and other sources, but must ultimately be pilot-tested with students (Almstrum et al., 2006).

There are other approaches for overall CI development (Almstrum et al., 2006) and for individual steps of the above process. For example, Goldman et al. (2008) proposed using a Delphi Process to facilitate reaching a consensus among CI developers for key concepts. To begin this process, 10 to 30 subject area experts each list 10 to 15 key concepts. These concepts are then reconciled into a single master list. Second, the experts rate each concept on three axes: importance, difficulty, and the extent to which students are expected to master the concept. Third, the experts are given the average and interquartile range (IQR) for each concept and each axis from the previous step and perform the rating again. Experts are required to anonymously justify any ratings that fall outside of the IQR. Finally, experts vote one last time in light of the justifications given in the previous step.

While the development and validation of many CIs follow steps similar to these protocols, there is no clear consensus on the precise order or necessity of the steps. Lindell, Peak, and Foster (2007) assessed the methodology used to create 12 different physics and astronomy CIs and discovered vast differences in the protocols followed. These differences included the ways that topics were established, the process for developing distractor answers for questions, the number of students on which the CI was tested, and the statistical measures (if any) used to validate the test.

Details of CI development aside, the process is time-consuming and lengthy, requiring a committed research team and input from many types of stakeholders. For example, soliciting expert opinion (whether informally or through a Delphi process) requires the research team to assemble a suitable panel of researchers and teachers, after which these experts must delineate the broad conceptual focus of the CI and then reach consensus on the particular concepts to test. To incentivize participation, panelists are often compensated for their time. Similarly, interviewing students requires suitable interviewer training and is complicated by the availability and willing participation of students (Adams & Wieman, 2011). The expense of the CI development process, both in terms of time and funding, may contribute to the lack of available CIs for computer science. Recent work suggests that CI development and maintenance could be made more lightweight by following open-source principles (Porter, Taylor, & Webb, 2014).

3. Challenges for computer science

As a young field, computer science faces many challenges with respect to concept inventory development. Almstrum et al. (2006) discuss many of the difficulties of creating a computer science concept inventory in the context of their proposal for a discrete math CI. Some of the difficulties are shared with other disciplines: determining the breadth of the CI, isolating concepts from related concepts, keeping to a small number of questions, generating useful distractors, etc. Other challenges are more unique to CS, and we discuss such challenges in this section.

3.1. Pre-test limitations

Almstrum et al. (2006) cite issues of notation and vocabulary as especially difficult in computer science, because, as they state, "the computing field is notorious for its dependence on notations and conventions" and "using a specialized vocabulary or notation in writing CI items may mask misconceptions of a more fundamental nature, particularly if the CI is used as a pretest." Authors of two recent preliminary CIs, one for computer architecture (Porter et al., 2013) and one for operating systems (Webb & Taylor, 2014), have echoed these difficulties in offering CIs as pre-tests. Additionally, Tew (2010) states "it is likely that a majority of student misconceptions are a result of instruction in CS rather than based upon a set of common, naive understandings [that students] bring to the topic from their experience in the world," suggesting that pre-tests may not be as useful in CS as they are in other sciences where students arrive with existing conceptions of how the natural world works. In Statics, another field where students tend not to enter courses with preconceptions about how things work, pre-test scores have been close to random, and the pre-test has only been found useful for comparison in courses where the students possess initial conceptions (Steif & Hansen, 2007). See the article by Herman, Zilles, and Loui in this special issue for another example of pre-test difficulties in CS assessments.

3.2. Programming language dependence, technology change, and other threats to immortality

In many fields (e.g. Newtonian mechanics), the covered content rarely, if ever, changes. For these fields, CI validation would mark the end of the development process. In a young field like CS, curricular change is common due to changes in the underlying technology we use. For example, many CS instructors are re-evaluating their course content to include topics related to parallel processing given the recent availability of multiple CPU cores. Another example would be the rise of power concerns in hardware design. Just a few decades ago, performance was the only important metric when designing hardware. Today, power has become a principal interest and concepts related to power conservation now appear in hardware courses.

Programming languages present a special case of our core technologies changing. The main programming languages used to teach CS courses change every so often in response to pedagogical innovation, new programming paradigms, or industry concerns. For example, Pascal's popularity in CS1 in the 1980s and 1990s has given way to Java and Python today, and those languages will certainly give way to others in the future. To deal with these changes, there are two broad choices: create and validate language-independent CIs, or adapt CIs appropriately when languages change.

Validating a language-independent CI involves considerable additional effort, including verifying that students can learn and understand a chosen pseudocode to an extent commensurate with their understanding of a true programming language. This has been done in one assessment of CS1 conceptual understanding (Tew, 2010). Besides shielding the CI from some language changes, such language independence also necessarily focuses on language semantics rather than syntax. Since a CI by nature focuses on only the core concepts of a course, it should not be affected by mere syntactical concerns. However, a closer look suggests that language independence may not in itself be sufficient for long-lasting CIs. CS1 courses, for example, vary widely not only in programming language, but also in contextual approach and programming paradigm. Courses may use an object-oriented or functional approach, or use languages with vastly different models of memory management. It is difficult to imagine one pseudocode that could capture this variety.

Although many core concepts are likely to persist over time, our field is one that changes and, as such, the basic tenets of what we do and how we do it are still in flux. It is hard to imagine a CI not requiring adaptation over time. An important open question to our community is to decide how best to create CIs that can withstand present and future variety.

3.3. Difficulty assessing skills

CIs for many computer science topics should perhaps include questions that evaluate students' ability to engage in *processes* such as code analysis, program design, program modification, and testing. These aspects of learning are difficult to assess among students with varying institutional and curricular backgrounds, often necessitating additional context. (For example, the AP CS exam described in Section 5.2 has incorporated a case study to adequately assess these processes.)

Evidence suggests that some aspects of understanding are hard to evaluate. For example, Zingaro, Petersen, and Craig (2012) commented on the difficulty of designing and grading traditional code-writing exercises. Simon et al. (2010) examined a collection of 76 data structures exams from 14 institutions, given between 1973 and 2009. These authors noted the shift in many CS2 courses from implementation of data structure internals to application and use of abstract data types. However, in the 35 years covered by the exams in this research, there was no noticeable shift to application questions.

This may mean that assessing students' ability to apply data structures to solve problems is not straightforward. A unique challenge for developers of CIs in CS is to navigate the interplay between conceptual understanding and applying that understanding to problem-solving situations.

Of relevance here are three related questions:

(1) Are processes in fact concepts? For example, is debugging strictly process, or is debugging in fact conceptual?
(2) Do processes contain a conceptual component? For example, is debugging a process within which exist related concepts such as data flow and control flow?
(3) Are processes strictly separate from concepts? For example, is there clean separation between a process such as debugging and a concept such as a loop?

The answers to these questions directly impact whether process should be tested on a conceptual CI. We suggest that it may be fruitful to search for concepts that underlie processes, evaluate whether these concepts are core concepts, and then include any core concepts on a CI. Future work is certainly warranted in this area.

3.4. Use of concept inventories vs. comprehensive assessments

As a relatively short assessment that is designed to measure student understanding of the most crucial concepts in a course, a CI is quite different from a comprehensive assessment of everything that is covered in a course. This naturally leads to questions of the suitability of a CI when used to assess student learning in order to compare across teachers, departments, institutions and pedagogical techniques. However, a CI's focus on these fundamental concepts makes it relatively easy to administer, and means that all students should be capable of performing well on it.

Hake used the FCI and its precursor, the Mechanics Diagnostic Test (MD), in a 6000 student study across 62 courses (Hake, 1998). He discusses using concept inventories rather than a comprehensive assessment, and argues that the FCI and MD have two main advantages: "the multiple-choice format facilitates relatively easy administration of the tests to thousands of students [. . . and] the questions probe for a conceptual understanding of the basic concepts of Newtonian mechanics in a way that is understandable to the novice who has never taken a physics course, yet at the same time are rigorous enough for the initiate" (Hake, 2007). It is this combination of ease of administration and universality of concepts that allow a CI to be used to compare across widely different contexts (e.g. different instructors, institutions, etc.) in a meaningful way.

Given the success of using CIs to assess the effectiveness of new pedagogical techniques within the physics community, calls to use them in a similar way have sprung up within the computer science education literature (Almstrum et al., 2006; Clement, 2004; Goldman et al., 2010). However, some may feel that a comprehensive assessment is more suitably used for this purpose. Almstrum et al. (2006) address this, saying: "The underlying philosophy is that using a limited instrument that has been validated is better than using none. A CI should be concise, yet accurate, and should probe into potential student mismodelings of fundamental concepts."

Almstrum et al. (2006) point out that performance on a CI should be correlated with performance on more comprehensive instruments. Similarly, Steif and Hansen (2007) suggest comparing CI results with performance on other measures such as course examinations to ensure the generality of the CI results. Almstrum et al. (2006) suggest that multiple CIs, each focusing on some subset of course topics, may be a solution for a course that covers a large number of fundamental concepts. A similar idea, that of modular CIs, has been proposed by Porter, Taylor, and Webb (2014).

It is worth noting that now, eight years after the review of the state of concept inventories in computer science by Almstrum et al. (2006), there are very few CIs available for CS subjects. The challenges presented in this section have likely played a role in hindering progress.

4. Concept inventories in computer science

Given these challenges, the development and adoption of concept inventories within computer science remains slow. While CIs for digital logic (Herman et al., 2010a) and CS1 (Tew & Guzdial, 2010) have been developed, neither is presently in widespread use.

This leaves instructors who wish to measure student learning gains without a standardized assessment tool. Due to the absence of such common evaluative referents in CS, we have no anchor with which to validate the effectiveness of new pedagogical innovations. That is, while new pedagogies are being rapidly proposed and deployed in CS contexts, we lack a critical tool for understanding their impact on student learning. This disconnect between learning goals and evaluation slows the pace of curricular reform as researchers grapple with contradictory results caused, in part, by locally produced assessment mechanisms (Tew, 2010).

Thus far, student misunderstandings have been identified by both formal CIs and, more commonly, other assessment mechanisms. As in other fields, these assessments often demonstrate that students understand far less than instructors expect (Lister et al., 2004; McCracken et al., 2001; Tew & Guzdial, 2011; Tew, McCracken, & Guzdial, 2005). Tables 2 and 3 provide results from concept inventories and informal assessments. The majority of the informal assessments address introductory programming course (CS1) topics, and demonstrate the wide variance in student performance depending on the assessment used. As such, these results serve to underscore the

Table 2. Post-test results from various concept inventories in computer science.

Exam content	Correct (avg.)
Unvalidated or incomplete CIs	
Algs. and Data structures (Paul & Vahrenhold, 2013)	n/a
BSTs and Hash tables (Karpierz & Wolfman, 2014)	42–63%
Computer architecture (Porter et al., 2013)	56%
Operating systems (Webb & Taylor, 2014)	55%
Validated CIs	
Digital logic CI (Herman et al., 2010a)	55%
CS1 language independent CA (Tew & Guzdial, 2011)	34%

Table 3. Informal CS1 assessments.

Exam content	Correct (avg.) (%)
Looping (Rainfall) (Soloway, Bonar, & Ehrlich, 1983)	33
BASIC programming (Bayman & Mayer, 1983)	31
Write calculator program (McCracken et al., 2001)	21
Comprehension/tracing (Lister et al., 2004)	60
Fundamental intro. concepts (Tew et al., 2005)	42
Number and date sorting (Chen, Lewandowski, McCartney, Sanders, & Simon, 2007)	59
Code value/reference assignment (Ma, Ferguson, Roper, & Wood, 2007)	63/17

importance of having *established assessment mechanisms*, which facilitate consistent and meaningful comparisons between curricular and pedagogical computer science practices.

4.1. Digital logic

Authors of the Digital Logic Concept Inventory (DLCI) (Herman, 2011; Herman, Loui, Kaczmarczyk, & Zilles, 2012; Herman, Loui, & Zilles, 2010b, 2011; Herman, Zilles, & Loui, 2009; Longino, 2006; Zilles, Longino, & Loui, 2006) established topics through a Delphi process (Goldman et al., 2008) that ascertained important concepts through a panel of instructors. The authors examined student misconceptions through interviews, the results of which informed the number of items per concept and the distractors for each question. In an alpha version of the DLCI, the authors collected additional misconception data by allowing students to supply answers for questions whose choices were not represented by the provided options. The validation evidence included useful distractors, item response curves, expert feedback, and observation of students solving the problems (Herman, Loui et al., 2011). Further validation evidence for the DLCI using classical test theory and item-response theory can be found in the article by Herman, Zilles, and Loui in this special issue.

By enabling performance comparisons across courses, the DLCI has allowed instructors to speculate on pedagogical improvements for digital logic courses (Herman & Handzik, 2010). In a preliminary study using the DLCI, it was found that students had considerable difficulty with bitwise manipulation of binary numbers. Looking specifically at Boolean logic, the authors noted that students tend to use faulty logic to reduce hard concepts to easier concepts, did not understand implication, and had difficulty when dealing with complemented variables (Herman, Kaczmarczyk, Loui, & Zilles, 2008). These results showed that students had difficulty even with topics instructors might consider to be relatively easy.

4.2. CS1: introduction to computer science

Tew and Guzdial developed an inventory of Foundational CS1 (FCS1) knowledge (Tew, 2010; Tew & Guzdial, 2010; Tew & Guzdial, 2011). Beginning with the tables of contents from the most popular CS1 textbooks, Tew pruned topics based on the CC2001 curriculum (ACM/IEEE-CS Joint Task Force on Computing Curricula, 2001), the agreement of three of the most popular of those textbooks, and a thematic analysis of concepts. This process yielded a set of 10 core CS1 topics and served as the content goal of the FCS1. Tew notes that typical test validation procedures may not be helpful when designing the first test of its kind in a new domain. For example, typical validation arguments include correlating scores on a new test to those of an existing test that targets a similar domain of knowledge. In the case of the FCS1, however, there was no such existing test. Note that Tew refers to the FCS1 as a Concept Assessment, rather than a Concept Inventory. We consider it a CI for the purposes of this work, but it is worth noting that there are some differences between the FCS1 development process and classic CI development; for example, the FCS1 distractors do not come from student interviews or misconceptions.

To account for shifts in programming language popularity over time, Tew developed a language-independent conceptual test in which students first learn to reason with pseudocode and then respond to questions written in that pseudocode. Tew first verified that students' conceptual understandings and misunderstandings are not localized to syntactic peculiarities of individual programming languages. She found that students exhibit the same difficulties across programming languages, independent of whether questions are closed form or open form. Using a think-aloud study, students demonstrated that they could understand and reason with the pseudocode; only rarely did the pseudocode directly contribute to students' misunderstandings. Finally, students took both the pseudocode version of the test and a version in the programming language used in their CS1 (e.g. Python, Java). Students' scores were significantly correlated, such that students who did well when working with the pseudocode also did well when working in a programming language. This suggests that knowledge from the programming language of instruction was transferred to the pseudocode test. Students scored an average of 34% on the pseudocode version and 49% on the language version of the FCS1 which both indicate difficulty with core CS1 concepts after a CS1 course.

There have been other steps toward a CS1 concept inventory as well. Kaczmarczyk, Petrick, East, and Herman (2010) report preliminary work on a CS1 concept inventory, identifying core concepts (Goldman et al., 2010) and student misconceptions (Kaczmarczyk et al., 2010). They identified four main themes in student misconceptions: students fail to understand the relationship between language concepts and underlying memory, students misunderstand how while loops work, students lack a basic understanding of objects, and students cannot trace code linearly.

4.3. Informal CS1 assessments

A number of assessments have been developed for CS1, summarized in Table 3. Poor results across the board demonstrate the widespread nature of misconceptions in CS and their consequent impact on students' ability to trace, read, and write code. In many of these assessments, researchers sought to ask students questions that many educators expect to be readily answerable by the end of a first CS course. For example, Lister et al. (2004) focused on students' ability to trace and understand code, arguing that students who cannot perform these rudimentary tasks cannot be expected to successfully write code. The focus of the questions was on variables, loops, and iteration and, even on these core CS1 concepts, student performance was poor (60%). These informal assessments have demonstrated that students lack the basic skills that we expect, yet these assessments are not themselves validated. Valid assessment measures would increase trust in our measures and facilitate comparable comparisons of data across institutions and pedagogical approaches.

Other research teams have developed assessment questions for CS1 and CS2. Sanders et al. (2013) created a repository of 654 multiple-choice questions covering CS1 and CS2 topics, tagged with metadata including difficulty, language, and topic. These tagged questions may be of use to researchers starting development of a CI for either course. Additionally, they report on development issues similar to those of a CI: making questions available, licensing, and using online infrastructure to develop questions and save metadata. Finally, the authors discuss multiple-choice "patterns": abstract categorizations of the skills required to answer a particular question. For example, compare-and-contrast questions ask the student to compare two pieces of code in terms of a relevant factor (runtime, memory, style, and so on). Debugging questions present code with a syntactic or semantic bug and ask students to identify the problematic line or select why the code is buggy. In all, 12 patterns are explored, and we suggest that this variety may serve as inspiration when drafting CI questions.

4.4. CS2: algorithms and data structures

Researchers have begun preliminary work on developing a validated CI for algorithms and data structures. Danielsiek, Paul, and Vahrenhold (2012) developed lists of important topics for a first-year algorithms course in Germany, roughly representing the algorithms component of a North American CS2 course. The authors analyzed 400 student exams to identify common errors and difficult topics, and consulted local faculty to verify the topics of central importance. To develop pilot multiple-choice items, the authors began with "flash test items" based on misconceptions from the literature or their exam analysis. Think-alouds with students were then used to obtain further student misconceptions. Newly discovered misconceptions include: students conflate heaps and binary search trees, lack procedural and conceptual knowledge related to invariants, and confuse dynamic programming with recursive divide-and-conquer formulations.

In follow-up work, Paul and Vahrenhold (2013) sought to investigate the utility of easy-to-evaluate instruments as compared to other instruments such as interviews and concept maps. They developed multiple-choice, fill-in-the-blank, and free-form questions and administered them to an average of 80 students. The authors note that for some topics these low-cost instruments can be used to detect student misconceptions, but for other topics the more demanding instruments such as interviews appear to be necessary. For example, a free-form question asking for definitions of **class** and **object** failed to uncover misconceptions, as students simply regurgitated textbook definitions. New misconceptions discovered in this work include students believing that each object must be referenced by at most one variable and that Java uses call by reference for variable passing. The authors also confirmed the earlier findings that students struggle with invariants and conflate heaps and binary search trees (Danielsiek et al., 2012). Of note is that student misconceptions are context-dependent: they may appear in some situations but not other closely related situations, depending on cues in particular problems. See Herman, Zilles, and Loui in this special issue for more discussion on this point.

A comprehensive summary and extension of the work of Danielsiek et al. (2012) and Paul and Vahrenhold (2013) appears in the article by Vahrenhold and Paul in this special issue. There, the authors offer four case studies of developing, evaluating, and validating potential CI questions. Contributions include in-depth discussion of the development of CI questions – from identifying concepts to validating the results – and a discussion of the ways that CI questions can be used as part of classroom instruction.

Recently, Karpierz and Wolfman (2014) interviewed nine instructors to identify misconceptions, analyzed 126 exams, and conducted 25 student interviews, resulting in the creation and validation of two CI questions. Using a student population distinct from both Danielsiek et al. (2012) and Paul and Vahrenhold (2013), they attempted but could not replicate the Danielsiek et al. (2012) finding of students confusing heaps and binary search trees. However, Karpierz and Wolfman (2014) did identify a different student misconception about binary trees, namely that binary trees are balanced by default. They also questioned students about hash table resizing, and found two common misconceptions: students often believe that the hash table is extended in-place when resized and that the keys do not need to be rehashed when the hash table is resized.

4.5. *Computer architecture*

A preliminary test of conceptual understanding was created for computer architecture by Porter et al. (2013). The examination was created by two computer architects with a background in pedagogy research. They brought together the final exams of multiple faculty at a large research institution, identified common exam questions, and identified the key, fundamental concepts associated with these common exam questions. The concepts identified were basic concepts critical to the course. The authors then designed questions, aiming for them to be correctly answered by any student passing the class. This resulted in nine questions on topics including the performance implications of deeper pipelines, the roles of various cache components, and performance analysis of single-cycle, multi-cycle, and pipelined processors.

The exam was run in four classes, taught by four different instructors at four different institutions. Pre-tests were found to be difficult for students due to a lack of understanding of basic terms and concepts. Somewhat discouragingly, on almost half the questions, there was no statistically significant improvement between the pre-test and the post-test. Overall, for all four classes, the per-question average on the post-test was only 56% correct. Although this was preliminary work, the results suggest a disconnect between what instructors thought students were learning and what was actually being learned.

4.6. *Operating systems*

A preliminary concept inventory for operating systems is currently under development (Webb & Taylor, 2014). The authors created the test based on their experiences teaching OS courses. Their work is part of an effort to democratize CI development in a manner akin to the open-source software movement (Porter, Taylor, & Webb, 2014).

Deploying their exam in four courses at three institutions, the authors identified a large variation in overall correctness rates by question. They conclude that certain concepts (e.g. indirection in file systems and thread synchronization) are more challenging than others (e.g. LRU page replacement). While the fact that there is variation between concepts is not surprising, they note that the availability of CI questions allows us to quantify that disparity.

For every question, they allowed students to respond with an option that stated "I am not familiar with this terminology / I don't know." The results show that, for the pre-test, students chose this option regularly, whereas it was selected relatively rarely on the post-test. The authors conclude that student confidence increased, even when they were not responding correctly.

5. Groundwork for future CI development

Future CI developers, maintainers, and adopters can leverage the significant body of work performed by the CS education research community and those tasked with developing standardized student assessments. Core concepts and misconceptions in CS, particularly in introductory CS, have been well studied by the community and CI developers can utilize those efforts as a potential head start in CI development. Lessons related to assessment maintenance, primarily from the standardized assessment community, inform CI developers of the need to plan for our assessments to change as our field changes. We explore these efforts and their relevance for CI development below.

5.1. Concepts and misconceptions

Significant progress has been made especially in the first two steps of CI development: establishing important topics and identifying student thinking. This section characterizes the types of research into identifying student misconceptions and important topics, insofar as such prior research can assist those interested in developing a CI. Such work should not be "redone," but should instead be repurposed for use in the first few steps of CI development. For further classification of student misconceptions and barriers to programming, see Clancy (2004).

Most research into topic importance occurs in CS1 and CS2 courses, typically by surveying or interviewing instructors. Instructors are quite useful for determining the important topics, but interviewing students directly is preferable for determining difficulty and misconceptions related to these topics (Goldman et al., 2010).

For example, Schulte and Bennedsen (2006) administered a web survey to university, college, and high school CS teachers. Among other research questions, these authors were interested in the topics that the teachers thought were most important to teach in CS1. The most relevant topics were selection and iteration, simple data structures, and parameters. Interestingly, these teachers did not believe that these topics were among the most difficult topics, instead citing recursion, algorithm efficiency, data structures, and other design-based topics as most difficult.

From other research, it is clear that iteration, in particular, is extremely difficult for students. For example, Soloway et al. (1982) tested students at the end of a first programming course on three looping questions. Novices scored only 39–44% on these questions. Perhaps, more concerning, intermediate programmers' performance on one of these three questions did not improve at all over the novice scores. This question, referred to as the "rainfall problem", involves writing a program that repeatedly reads integers until 99999 is entered, then outputs the average of the non-negative integers (not including the 99999 terminator). Performance on this problem, in all known studies, is poor (Guzdial, 2011): in one case, students at the end of a CS1 scored 14% on the problem. One might hope that, in the intervening 32 years since Soloway et al. (1982) published this study, things would have changed and our students would perform better on the rainfall problem. Unfortunately, this is not the case: students struggle with this problem today as they did in 1982, even when adjusting the problem based on today's typical CS1 teaching contexts (Simon, 2013).

Some novice difficulties can be addressed by modifying aspects of programming language syntax to match natural preferences of novice programmers. For example, novices do better on the rainfall problem when permitted to exit a loop in the middle rather than being forced to use a loop that terminates at the top or bottom (Soloway et al., 1983). Other difficulties seem more resistant, in that they cannot be remedied through syntax or presentational changes. Pea (1986) offers some examples of these latter types of misconceptions. For example, students may believe that all lines of a program are simultaneously active, so that as soon as the condition of any if-statement becomes true, its associated code fires (even if that code is much earlier in the control flow).

Data produced in the context of teaching has also been used as a source of student difficulties and misconceptions. Though conveniently available, this data does not suggest reasons for the difficulties, only that these difficulties exist. For example, Robins, Haden, and Garner (2006) tracked the questions asked by students in laboratory sessions. The frequency of some types of questions decreased throughout the semester; for example, arrays proved quite problematic at the start, but such questions decreased substantially in later weeks. In contrast, some topics seemed not to become any easier, such as classes and instance variables. From this and other work (Soloway et al., 1982), it seems that many students do not acquire core conceptual knowledge of introductory topics such as assignment statements and loops. Students may believe that variables can hold multiple values at the same time, or may not be able to appropriately choose among loop constructs, or may struggle with simple variable updates (Soloway et al., 1982).

Other work is more closely tied to CI development in that the authors generated misconceptions for future use in a CI. For example, Kaczmarczyk et al. (2010) interviewed students to study misconceptions on 10 CS1 topics previously identified as important and difficult by experts (Goldman et al., 2008). Many misconceptions emerged, including students ascribing undue semantic meaning to variable declarations, believing that memory is allocated for uninstantiated objects, and not realizing that instance variables of primitive types have default values in Java. Similar work seeks to uncover mental models of programming held by students through an investigation of the types of errors that students make. Work shows, for example, that students hold a wide range of non-viable models of value and reference assignment (Ma et al., 2007). Some students assigned from left to right instead of right to left, interpreted the assignment operator as an equality comparison operator, or believed that reference assignment is only used to name an object. Similarly, students can hold a variety of non-viable models of recursion, including imagining a recursive procedure as a single entity and failing to pass values back up the recursion stack (Götschi, Sanders, & Galpin, 2003).

5.2. *The Advanced Placement Computer Science examinations*

The Advanced Placement Computer Science (AP CS) examinations have been the most widespread assessment vehicles in the USA for CS1 (tested by the AP CS A exam) and CS2 (tested by the AP CS AB exam).[1] They are significantly different from a CI in that they are comprehensive assessments, and they aim to evaluate student understanding to certify a student as proficient in the examination topic. A CI, on the other hand, focuses on misconceptions students may encounter along the way to that proficiency.

Despite these differences in assessment intent, it is likely that CI design in computer science can benefit from the extensive development efforts for the AP CS exams. In both, there is the need to identify core learning outcomes for students and to periodically verify that the assessment still meets course and curricular standards. Moreover, AP CS questions may suggest counterparts (in either content or format) among CI items, and AP CS results may suggest misconceptions that a CI author should address.

The AP CS exams may also point the way toward more significant evolution of a CI. For example, a 1989 survey of high school AP CS and college CS instructors revealed a serious discrepancy between AP CS AB and the collegiate CS2 courses, namely the length of programming projects. In 1995, this disparity prompted the inclusion of a *case study* as the basis for exam questions. A case study describes a programming problem, the narrative process used by an expert to solve the problem, and one or more solutions to the problem. Case studies emphasize the decisions encountered by the programmer during design and development and the criteria used to choose among alternatives. Case studies have many benefits in programming education; see Clancy and Linn (1992) and Linn and Clancy (1992) for more details. For the purposes of assessment, engagement with a case study approximates the writing of a long program and thus adds to the face validity of the exam. It also allows the evaluation of a student's ability to design, analyze, modify, and debug code in a way that is both educational and measurable.[2]

The language on which the AP CS exam is based has changed twice: from Pascal to C++ in 1999 and from C++ to Java in 2004. Prior to the switch to C++, an AP CS Test developer asserted that "the multiple-choice section of one recent APCS examination was examined to determine what changes would be necessary were the course to use C++ instead of Pascal. Over 30% of the questions could appear on a C++ exam with no change whatsoever, only one question would require more than trivial syntactic substitutions, and no question's underlying content would change" (Nevison et al., 1995). Nevertheless, each switch required substantial effort, mainly in the form of presentations and workshops. Also, the Pascal-based version of the case study was translated to C++ for the 1999 switch, and a similar rewrite was done to support the switch to Java.

Given the extensive history of development and revision of the AP CS exam, there are clear lessons for those undertaking the development of a CI. For example, CIs must be written to accommodate language and curricular change, and we must discover and experiment with novel types of assessments that may be unique to CS. Especially interesting is how the AP CS exam addressed broader learning goals, such as understanding the development process of relatively complex programs. The need for the inclusion of a case study in the AP CS exam leads to questions of what types of materials a CS concept inventory must include in order to test conceptual knowledge related to topics such as designing and developing programs, and what types of questions we can ask to test that knowledge.

6. Moving forward

In this literature review, we have described the impact of CIs in other sciences, characterized the landscape of CIs in computer science, and discussed how existing research toward identifying core concepts and common misconceptions can be leveraged to accelerate CS concept inventory development. Despite the slow development of concept inventories in computer science relative to other STEM disciplines, CIs have recently seen increased interest from the CS community. We believe that this trend will continue going forward, and we encourage broader participation in CI development.

To date, much of the work in CS has invested effort to improve the quality and coverage of CIs for introductory courses. Such courses are particularly important for the retention and addition of potential CS majors, and in some cases are the only CS courses that students might take. However, while an introductory course serves an important role as the first formal computer science experience for many students, it may be time to broaden the focus toward the development of concept inventories to cover the core CS curriculum. Leveraging an analogy from compiler principles, one might imagine the curriculum as a control flow graph where each course has incoming conceptual dependencies and each course produces understanding of core concepts. Developing concept inventories throughout the curriculum would enable us to measure and quantify this critical flow of knowledge, with the potential for several improvements:

(1) Better understanding of the relationships between courses and, hence, potential curricular improvements to ensure students are well-prepared when courses begin.
(2) Identification of student misconceptions at the end of a course to help instructors intervene before that student attempts courses with that concept as a prerequisite.
(3) Improved instructor understanding of student abilities at the start of a course, as instructors could obtain post-test results from prior courses.

Of course, designing CIs to cover the CS curriculum is not without challenges, including curricular variations, intuitive pre-test language, and higher level skill assessment. However, in spite of the challenges, the CS community has already laid a solid foundation for this type of work to continue. We believe that solutions to these challenges are within the reach of a coordinated effort from the CS community. Should the community embrace this challenge and develop CIs for a range of computer science courses, these CIs could usher in a new era of curricular and pedagogical innovation and evaluation.

Acknowledgements
We thank Steve Wolfman and the anonymous reviewers for their thorough reviews and helpful comments on our work.

Funding
This work was supported in part by NSF [grant number 1140731].

Notes

1. Other standardized tests such as the CS International Baccalaureate and the CS Major Field Examination share many features of the AP CS exam and are not treated separately here.
2. Case studies will no longer be a part of the AP CS A exam starting in 2014, partly because the elimination of the AP CS AB exam in 2009 decreased the emphasis on working with relatively large programs.

References

ACM/IEEE-CS Joint Task Force on Computing Curricula. (2001). *Computer Science Computing Curricula 2001*.

Adams, W. K., & Wieman, C. E. (2011). Development and validation of instruments to measure learning of expert-like thinking. *International Journal of Science Education, 33*, 1289–1312.

Allen, K. (2006). *The statistics concept inventory: Development and analysis of a cognitive assessment instrument in statistics* (Doctoral dissertation). Retrieved from http://pdf-release.net/external/1543023/pdf-release-dot-net-Kirk%20Allen%20dissertation.pdf

Almstrum, V. L., Henderson, P. B., Harvey, V., Heeren, C., Marion, W., Riedesel, C., & Tew, A. E. (2006). Concept inventories in computer science for the topic discrete mathematics. *ACM SIGCSE Bulletin, 38*, 132–145.

Anderson, D. L., Fisher, K. M., & Norman, G. J. (2002). Development and evaluation of the conceptual inventory of natural selection. *Journal of Research in Science Teaching, 39*, 952–978.

Bailey, J. M. (2008). Development of a concept inventory to assess students' understanding and reasoning difficulties about the properties and formation of stars. *Astronomy Education Review, 6*, 133–139.

Bardar, E. M., Prather, E. E., Brecher, K., & Slater, T. F. (2006). Development and validation of the light and spectroscopy concept inventory. *Astronomy Education Review, 5*, 103–113.

Bayman, P., & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM, 26*, 677–679.

Buck, J. R., Wage, K. E., Hjalmarson, M. A. & Nelson, J. K. (2007). Comparing student understanding of signals and systems using a concept inventory, a traditional exam, and interviews. In *IEEE frontiers in education (S1G-1)*. Milwaukee, WI.

Chen, T. Y., Lewandowski, G., McCartney, R., Sanders, K. & Simon, B. (2007). Commonsense computing: Using student sorting abilities to improve instruction. In *Proceedings of the 38th ACM technical symposium on computer science education (SIGCSE)* (pp. 276–280), Covington, KY.

Clancy, M. (2004). Misconceptions and attitudes that interfere with learning to program, Chap. 1. In M. Petre & S. Fincher (Eds.), *Computer science education research*. London: Routledge Falmer.

Clement, J. M. (2004). A call for action (research): Applying science education research to computer science instruction. *Computer Science Education, 14*, 343–364.

Clancy, M. & Linn, M. C. (1992). Case studies in the classroom. In *Proceedings of the 23rd ACM technical symposium on computer science education (SIGCSE)* (pp. 220–224), Kansas City, MO.

Crouch, C. H., & Mazur, E. (2001). Peer instruction: Ten years of experience and results. *American Journal of Physics, 69*, 970–977.

Danielsiek, H., Paul, W. & Vahrenhold, J. (2012). Detecting and understanding students' misconceptions related to algorithms and data structures. In *Proceedings of the 43rd ACM technical symposium on computer science education (SIGCSE)* (pp. 21–26), Raleigh, NC.

Epstein, J. (2007). Development and validation of the calculus concept inventory. In *Proceedings of the ninth international conference on mathematics education in a global community* (Vol. 9, pp. 165–170), Charlotte, NC.

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a Delphi process. In *Proceedings of the 38th ACM technical symposium on computer science education (SIGCSE)* (pp. 256–260), Portland, OR.

Goldman, K., Gross, P., Heeren, C., Herman, G. L., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2010). Setting the scope of concept inventories for introductory computing subjects. *ACM Transactions on Computing Education (TOCE), 10*(5), 1–29.

Götschi, T., Sanders, I. & Galpin, V. (2003). Mental models of recursion. In *Proceedings of the 34th ACM technical symposium on computer science education (SIGCSE)* (pp. 346–350), Reno, NV.

Gray, G. L., Costanzo, F., Evans, D., Cornwell, P., Self, B. & Lane, J. L. (2005). The dynamics concept inventory assessment test: A progress report and some results. In *Proceedings of the 2005 American Society for Engineering Education annual conference and exposition*, Portland, OR.

Guzdial, M. (2011). From science to engineering. *Communications of the ACM, 54*, 37–39.

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

Hake, R. (2007). Six lessons from the physics education reform effort. *Latin American Journal of Physics Education, 1*, 24–31 .

Helgeland, R. & Rancour, D. (2003). Circuits concept inventory. Retrieved from http://www.foundationcoalition.org/home/keycomponents/concept/circuits.html

Herman, G. L. (2011). *The development of a digital logic concept inventory* (Doctoral dissertation), University of Illinois. Retrieved from https://www.ideals.illinois.edu/bitstream/handle/2142/24134/Herman_Geoffrey.pdf?sequence=1

Herman, G. L. & Handzik, J. (2010). A preliminary pedagogical comparison study using the digital logic concept inventory. In *IEEE frontiers in education* (pp. F1G–1), Washington, DC.

Herman, G. L., Kaczmarczyk, L., Loui, M. C. & Zilles, C. (2008). Proof by incomplete enumeration and other logical misconceptions. In *Proceedings of the fourth international workshop on computing education research (ICER)* (pp. 59–70), Sydney.

Herman, G. L., Loui, M. C., Kaczmarczyk, L., & Zilles, C. (2012). Describing the what and why of students' difficulties in boolean logic. *ACM Transactions on Computing Education (TOCE), 12*(3), 1–28 .

Herman, G. L., Loui, M. C. & Zilles, C. (2010a). *Creating the digital logic concept inventory*. In *Proceedings of the 41st ACM technical symposium on computer science education (SIGCSE)* (pp. 102–106), Milwaukee, WI.

Herman, G. L., Loui, M. C. & Zilles, C. (2010b). Work in progress-how do engineering students misunderstand number representations? In *IEEE frontiers in education* (pp. T3G–1), Washington, DC.

Herman, G. L., Loui, M. C., & Zilles, C. (2011). Students' misconceptions about medium-scale integrated circuits. *IEEE Transactions on Education, 54*, 637–645.

Herman, G. L., Loui, M. & Zilles, C. (2011). Administering a digital logic concept inventory at multiple institutions. In *Proceedings of the 2011 American Society for Engineering Education annual conference and exposition* (pp. 26–29), Vancouver, BC.

Herman, G. L., Zilles, C. & Loui, M. C. (2009). Work in progress-students' misconceptions about state in digital systems. In *IEEE frontiers in education* (pp. 1–2), San Antonio, TX.

Hestenes, D., Wells, M., & Swackhamer, G. (1992). Force concept inventory. *The Physics Teacher, 30*, 141–146.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P. & Herman, G. L. (2010). Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on computer science education (SIGCSE)* (pp. 107–111), Milwaukee, WI.

Karpierz, K. & Wolfman, S. A. (2014). Misconceptions and concept inventory questions for binary search trees and hash tables. In *Proceedings of the 45th ACM technical symposium on computer science education (SIGCSE)* (pp. 109–114), Atlanta, GA.

Libarkin, J., Ward, E., Andersln, S., Kortemeyer, G., & Raeburn, S. (2011). Revisiting the geoscience concept inventory: A call to the community. *GSA Today, 21*, 26–28.

Lindell, R. S., Peak, E. & Foster, T. M. (2007). Are they all created equal? A comparison of different concept inventory development methodologies. In *Proceedings of the American Institute of Physics conference (AIP)* (Vol. 883, pp. 14–17), Syracuse, NY.

Linn, M. C., & Clancy, M. (1992). The case for case studies of programming problems. *Communications of the ACM, 35*, 121–132.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., ... Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin, 36*, 119–150.

Longino, J. T. (2006). *Boolean blunders: Identification and assessment of student misconceptions in a digital logic course* (Doctoral dissertation), University of Illinois. Retrieved from http://www.cs.illinois.edu/~zilles/papers/digital_concepts.msthesis.pdf

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2007). Investigating the viability of mental models held by novice programmers. *ACM SIGCSE Bulletin, 39*, 499-503.

Mazur, E. (2009). *Confessions of a converted lecturer*. Baltimore County: Guest lecture at the University of Maryland.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33*, 125–180.

Nevison, C. H., Kmoch, J., Noonan, R. E., Corica, T., Fix, S., & Kay, D. G. (1995). Changes in the advanced placement computer science course: Case studies and C++. *ACM SIGCSE Bulletin, 27*, 374-375.

Ogunfunmi, T. & Rahman, M. (2010). A concept inventory for an electric circuits course: Rationale and fundamental topics. In *Proceedings of the 2010 IEEE international symposium on circuits and systems* (pp. 2804–2807), Paris.

Paul, W. & Vahrenhold, J. (2013). Hunting high and low: Instruments to detect misconceptions related to algorithms and data structures. In *Proceedings of the 44th ACM technical symposium on computer science education (SIGCSE)* (pp. 29–34), Denver, CO.

Pea, R. D. (1986). Language-independent conceptual “bugs” in novice programming. *Journal of Educational Computing Research, 2*, 25-36.

Porter, L., Garcia, S., Tseng, H. W. & Zingaro, D. (2013). Evaluating student understanding of core concepts in computer architecture. In *Proceedings of the 18th annual conference on innovation and technology in computer science education (ITiCSE)* (pp. 279–284), Canterbury.

Porter, L., Taylor, C. & Webb, K. (2014). Leveraging open source principles for flexible concept inventory development. In *Proceedings of the 19th annual conference on innovation and technology in computer science education (ITiCSE)* (pp. 243–248), Uppsala.

Reay, N. W., Li, P., & Bao, L. (2008). Testing a new voting machine question methodology. *American Journal of Physics, 76*, 171-178.

Robins, A., Haden, P. & Garner, S. (2006). Problem distributions in a CS1 course. In *Proceedings of the 8th Australasian conference on computing education* (Vol. 52, 165–173), Darlinghurst.

Sadler, P. M., Coyle, H., Miller, J. L., Cook-Smith, N., Dussault, M., & Gould, R. R. (2009). The Astronomy and Space Science Concept Inventory: Development and validation of assessment instruments aligned with the K-12 National Science Standards. *Astronomy Education Review, 8*. doi:10.3847/AER2009005

Sanders, K., Ahmadzadeh, M., Clear, T., Edwards, S. H., Goldweber, M., Johnson, C., ... Spacco, J. (2013). The Canterbury questionbank: Building a repository of multiple-choice CS1 and CS2 questions. In *Proceedings of the 18th annual conference on innovation and technology in computer science education (ITiCSE) – working group reports* (pp. 33–52), Canterbury.

Schulte, C. & Bennedsen, J. (2006). What do teachers teach in introductory programming? In *Proceedings of the second international workshop on computing education research (ICER)* (pp. 17–28), Canterbury.

Simon. (2013). Soloway's rainfall problem has become harder. In *Proceedings of the 2013 international conference on learning and teaching in computing and engineering* (pp. 130–135), Macau.

Simon, B., Clancy, M., McCartney, R., Morrison, B. B., Richards, B. & Sanders, K. (2010). Making sense of data structures exams. In *Proceedings of the sixth international workshop on computing education research (ICER)* (pp. 97–106), Aarhus.

Smith, M. K., Wood, W. B., & Knight, J. K. (2008). The genetics concept assessment: A new concept inventory for gauging student understanding of genetics. *CBE-life sciences Education, 7*, 422–430.

Soloway, E., Bonar, J., & Ehrlich, K. (1983). Cognitive strategies and looping constructs: An empirical study. *Communications of the ACM, 26*, 853–860.

Soloway, E., Ehrlich, K., Bonar, J. & Greenspan, J. (1982). What do novices know about programming? In A. Badre & B. Shneiderman (Eds.), *Directions in human-computer interaction*. New York: Ablex.

Steif, P. S., & Dantzler, J. A. (2005). A statics concept inventory: Development and psychometric analysis. *Journal of Engineering Education, 94*, 363–371.

Steif, P. S., & Hansen, M. A. (2007). New practices for administering and analyzing the results of concept inventories. *Journal of Engineering Education, 96*, 205-212.

Tew, A. E. (2010). *Assessing fundamental introductory computing concept knowledge in a language independent manner* (Doctoral dissertation). Georgia Institute of Technology. Retrieved from http://smartech.gatech.edu/bitstream/handle/1853/37090/tew_allison_e_201012_phd.pdf

Tew, A. E. & Guzdial, M. (2010). Developing a validated assessment of fundamental CS1 concepts. In *Proceedings of the 40th ACM technical symposium on computer science education (SIGCSE)* (pp. 97–101), Milwaukee, WI.

Tew, A. E. & Guzdial, M. (2011). The FCS1: A language independent assessment of CS1 knowledge. In *Proceedings of the 41st ACM technical symposium on computer science education (SIGCSE)* (pp. 111–116), Dallas, TX.

Tew, A. E., McCracken, W. M. & Guzdial, M. (2005). Impact of alternative introductory courses on programming concept understanding. In *Proceedings of the first international workshop on computing education research (ICER)* (pp. 25–35), Seattle, WA.

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

Webb, K. & Taylor, C. (2014). Developing a pre- and post-course concept inventory to gauge operating systems learning. In *Proceedings of the 45th ACM technical symposium on computer science education (SIGCSE)* (pp. 103–108), Atlanta, GA.

Zeilik, M. (2002). Birth of the astronomy diagnostic test: prototest evolution. *Astronomy Education Review, 1*, 46–52.

Zilles, C., Longino, J. & Loui, M. (2006). Student misconceptions in an introductory digital logic design course. In *Proceedings of the 2006 American Society for Engineering Education annual conference and exposition*, Chicago, IL.

Zingaro, D., Petersen, A. & Craig, M. (2012). Stepping up to integrative questions on CS1 exams. In *Proceedings of the 42nd ACM technical symposium on computer science education (SIGCSE)* (pp. 253–258), Raleigh, NC.