# Design an Assessment for an Introductory Computer Science Course: A Systematic Literature Review

Qingwan Cheng
*Boston College*
Chestnut Hill, MA
chengqf@bc.edu

Angela Tao
*Boston College*
Chestnut Hill, MA
taoab@bc.edu

Huangliang Chen
*Boston College*
Chestnut Hill, MA
chenbtw@bc.edu

Maira Marques Samary
*Boston College*
Chestnut Hill, MA
marquemo@bc.edu

***Abstract*—This Research Full Paper presents a literature review on introductory CS assessments. As Computer Science (CS) becomes increasingly popular in the world of big data, more college students are taking introductory CS courses to follow the trend. However, students have different levels of exposure to CS when they start CS in college, varying from no experience to years of coding experience. This disparity in the amount of exposure can pose many problems. In the same intro-level class, some students may think they do not learn anything new, while others believe the professor goes through concepts so fast that they cannot follow the pace. CS professors may also feel confused about how to control the pace and content of introductory courses. Therefore, CS departments need an assessment to measure students' CS knowledge and then assign them to the appropriate CS introductory courses. Though a vast number of research papers discuss CS introductory courses, few of them focus on developing and validating assessments that evaluate students' background knowledge (i.e., concept inventory). This paper aims to conduct a systematic literature review to gain an overview of the most important introductory CS contents that need to be assessed, the methods to evaluate students' understanding of introductory CS concepts, and some existing assessments/concept inventories. By reviewing relevant papers over the last 15 years, we selected 48 papers that are related to assessments and corresponding introductory courses. We collected assessment information and made tables of contents, question types, programming languages, and origins to analyze data. We found that iterations, conditionals, and variables are the most popular assessment contents. Multiple-Choice and Code Writing questions, which are more popular than Code Explaining questions, are the preferred ways of assessing students' knowledge. Scratch is commonly used as a CS0 assessment language, while Python, Java, C, and Pseudo-codes are employed in CS1 assessments. FCS1, SCS1, and BDSI are existing assessments/concept inventories for CS1 or CS2, but there is not yet a validated assessment or concept inventory that encompasses introductory concepts of CS0 (computational thinking), CS1 (fundamentals), and CS2 (data structure). Designing a comprehensive CS0/CS1/CS2 assessment by using the information collected from our systematic literature review would help students choose the right course and benefit educators and researchers for general use.**

***Index Terms*—computer science education, introductory CS courses, assessment, concept inventory, systematic literature review**

## I. Introduction

In today's world, there is no need to state the significance of teaching computational thinking or having a basic Computer Science (CS) knowledge. Not everybody will or can become a software engineer, but people with CS foundations may feel comfortable using advanced technologies, including efficiently finishing everyday work and avoiding online scams (phishing, password theft, and many others). There is currently a global trend to engage younger students with CS education. The Computer Science Teachers Association (CSTA) develops curriculum, provides resources, and promotes CS Programs at the K-12 educational level [63]. Some students are also trying to learn coding proactively through school courses, after-school programs, and coding academies. However, many students still do not know anything about computational thinking and/or programming [35]. At the same time, a bachelor's degree in CS and other related fields is becoming competitive and popular. According to the National Center for Education Statistics [1], from 2018 to 2019, CS climbed to the 9th most popular major and the 4th most popular STEM major for college-bound students. This trend predicts the growing popularity of the CS major. Therefore, when students declare the same degree and go to the same class, colleges and universities have to tackle this heterogeneous scenario of student knowledge in an introductory CS course: students with no background exposure to CS, students with a bit of knowledge (computational thinking knowledge), students with 1-2 years of programming experience, and students with more than two years of programming experience.

Addressing the heterogeneity of students' knowledge of CS when they enter introductory courses is a big issue for colleges and universities, since the approach of one course with specific requirements does not fit students with this diverse knowledge background [2]. Therefore, CS departments may need to assign students with more coding experience to a higher-level introductory course (i.e., CS2), while keeping students with no or little coding experience in a basic-level introductory course (i.e., CS0). However, during the course registration process, when students declare that they have several years of CS practice, we cannot allow them to skip introductory courses as they desire. CS exposure, or even a good grade on AP Computer Science, does not necessarily mean that they have a strong CS foundation. Students should be allowed to skip introductory CS courses only when the institution is able to guarantee that they have enough knowledge of the required contents. As a consequence, the need for guaranteed knowledge raises some questions about how educators can

evaluate students' knowledge of introductory CS contents and what contents should be included in an introductory CS concept inventory.

This research aims to find out what introductory CS concepts need to be evaluated and how these concepts are assessed for different introductory CS courses. In other words, we hope to determine how higher education institutions are handling the issue of student knowledge heterogeneity in an introductory CS course. With the research information, it is possible to define a group of concepts and design an assessment that encompasses CS0, CS1, and CS2 materials. After implementing the assessment, CS department advisors may be able to provide students with good advice on what is the most appropriate introductory CS course for them according to their foundational knowledge. Therefore, we performed a systematic literature review, with the goal of answering the following research questions:

- RQ1: What are the most important topics/content that need to be assessed as part of the CS introductory courses?
- RQ2: How are students evaluated on their understanding of introductory CS concepts in introductory CS courses (e.g., CS0, CS1, and CS2)?
- RQ3: Is there a placement test or concept inventory that measures CS0/CS1/CS2 contents or part of them?

We will present some known issues of CS introductory courses in section II and discuss what introductory CS is in Section III. Section IV analyzes some previous work, and Section V presents the methodology we used. We show our results in Section VI, talk about threats to the validity of our literature review in Section VII, and conclude our paper with remarks of what we found in Section VIII.

## II. Issues with introductory CS courses

Though a CS degree is becoming increasingly popular, finishing a major in CS is challenging. According to Beaubouef and Mason [3], dropout and failure rates are exacerbated during the first two years of the program, reaching as high as 30%-40%. Even though students begin studying for CS degrees, a high proportion of students do not graduate [25]. Many students face difficulty in their introductory CS courses, so they are not well prepared for upper-level CS courses which focus more on applications and algorithms. Without a good foundation constructed during the intro-level courses, students will confront more problems, lose confidence, and, finally, fail the class or drop the major. In general, there are two potential risks for students who take introductory CS courses.

Firstly, students have different levels of knowledge and programming experiences. Some students entering an intro-level CS course have basic knowledge, while others do not. Some students enter the course knowing nothing about CS/programming, while others have extensive experience either from high school or by being self-taught. It is normal for a CS instructor to hear conflicting voices from students; some complain that the pace of the class is too fast, while others complain that the pace is too slow. This conflict is difficult to reconcile. Most intro-level CS courses have a large amount of students, making it difficult for instructors to adjust teaching strategies. In some courses, the instructor is only responsible for teaching, while teaching assistants are in charge of any other coursework. The instructor's huge class capacity and limited approachability result in extraordinary attrition, which is associated with high failure rates and disenchantment with programming [39].

Secondly, the structure of the CS major course list provides limited choices for students on introductory courses. Other STEM subjects, such as mathematics, have a more developed and mature structure of course lists. For example, students with different levels of foundation may choose different courses; they can start their degrees with either Calculus I, Calculus II, or Multivariable Calculus. The knowledge of those courses is consistent meaning the former can serve as the prerequisite for the next. However, in CS programs, courses are not fluid because the knowledge is not going deeper but wider. The connection between CS1 and CS2 is weak because it is difficult to find any common definition among them [20]. According to Hertz [20], "after surveying CS1 and CS2 instructors, there is little agreement on how important topics are to each of these courses and less agreement on how well students master the material." The contentious argument among instructors foreshadows the confusion that students experience during the class. Thus, to provide a more solid connection, the goal for the CS1-CS2 sequence includes giving students a thorough understanding of object-oriented design and programming. In this case, students may be expected to miss connections between CS and non-programming knowledge or activities [54]. Even in a single-semester course of CS1, the contexts of each unit are not necessarily related. Thus, to succeed in the CS course, students need to make up for missing knowledge on their own. As a result, this might hinder students' progression in academic performance and take away their freedom to choose which path they want to follow and what they want to learn.

With the issues of having students in different levels of background knowledge in one class and inconsistent requirements for each introductory class (CS0/CS1/CS2), there is a need to design a concept inventory or an assessment to evaluate what topics students have already understood. After having a sense of students' CS background, CS departments can help students choose the right introductory course based on the higher education institution's course design. In order to have some background knowledge on what an introductory CS concept inventory should include, we first do some research on what introductory CS is as a whole, from computational thinking (CS0) to basic data structure (CS2).

## III. What is introductory CS?

ACM-IEEE has curriculum guidelines for five sub-disciplines of computing, and one of them is CS [21] ( CS-Guidelines). CS-Guidelines discuss introductory courses in a very general way, stating there are multiple ways that it can be taught: "Computer science, unlike many technical disciplines,

does not have a well-described list of topics that appear in virtually all introductory courses," and they continue: "An important challenge for introductory courses, and a key reason the content of such courses remains a vigorous discussion topic after decades of debate, is that not everything relevant to a computer scientist (programming, software processes, algorithms, abstraction, performance, security, professionalism, etc.) can be taught from day one. In other words, not everything can come first, and as a result, some topics must be pushed further back in the curriculum, in some cases significantly so." While CS-Guidelines cannot set a certain and specific standard of what should be taught in each introductory course, higher education institutions have more freedom on designing their own structures of courses and deciding what to be taught in which intro-level class. However, at the same time, CS-Guidelines provide colleges and universities with some instructions on how to approach introductory CS courses.

According to the CS-Guidelines [21], there are five typical approaches for introductory programming courses:

- "Imperative-first" in which implementations are done in programming-first model that controls structure first and accommodates students from other fields;
- "Objects-first" in which objects and object-oriented design are introduced in early phases;
- "Functional-first" in which the language with a simple functional syntax is used to teach algorithmic concepts;
- "Breadth-first" where the course is designed to provide a broader introduction to concepts in computing without the constraints of learning the syntax of a specific programming language;
- "Algorithms-first" where algorithms is much more emphasized than syntax;
- "Hardware-first" where circuits are first introduced and the layers of abstract machine hierarchy are built on with increasing sophistication levels.

The Computing Curricula 2001 [61] and the CS-Guidelines [21] advise that changing introductory CS courses from two-course sequence to three-course sequence can be an optimal option for some schools. With the increasing number and complexity of CS topics, three-course sequence is needed to help students build a better Math foundation and assimilate all the essential concepts with enough time. We may deal with this in one of two ways:

- front-load way, where a new course is added at the beginning of the traditional CS1/CS2 sequence;
- back-load way, where after the traditional sequence CS1/CS2, a third course is offered.

The CC2001 Task Force [61] developed two-course and three-course models for imperative-first approach, objects-first approach, breadth-firs approach and found that the three-course model might be better under these tracks.

After understanding how CS-Guidelines set a big picture for higher education institutions, we did some research on how colleges and universities applied them to the real practice. Dale [10] published a survey in 2004 with 351 faculty members, where 66.9% of the respondents said that they used object-oriented paradigm for at least part of their introductory courses, but they did not mention anything specific regarding the approach. In 2006, Schulte and Bennedsen [48] did another survey with 242 faculty members, where 85% of respondents said that they used an object-oriented paradigm, and 60% declared their use of the "objects-first" approach. In 2011, Davies et al. [11] did a new survey to evaluate what changed and expanded their search separating CS0, CS1, and CS2. They reported that 65% of the 371 respondents' first programming courses followed the object-oriented paradigm. Institutions tended to use the same approach in the course sequence CS1-CS2. They also said that 60% of the respondents reported having a CS0 course, which is usually offered for non-majors. The majority of institutions that offer it are CS Ph.D. granting institutions, which are 75% more likely to offer a CS0 course.

A systematic literature review was published in 2018 where Luxton-Reilly et al. [33] performed an in-depth analysis of the introductory programming related work. They reported that, in Europe, students usually follow the procedural paradigm, then followed by the object-oriented paradigm. They also talked about courses where the two paradigms are taught in the same course, procedural followed by object-oriented paradigm. They concluded their analysis by saying that object-oriented paradigm is the most used, but there is no final conclusion; "it is by no means clear that any paradigm is superior to any other for the teaching of introductory programming."

The guidelines and applied approaches to introductory CS courses would help build a comprehensive concept inventory. Moreover, since higher education institutions have different approaches to introductory CS courses, it is vital to research if there is an existing concept inventory or an assessment that encompasses CS0, CS1, and CS2 materials. If there is not, it is necessary to understand what concepts, question types, and languages need to be included in an assessment to better evaluate a student's knowledge foundation and then refer him/her to the most appropriate class.

## IV. Related Work

As far as our knowledge and search, there is no systematic literature review paying attention to introductory CS concept inventory or assessment. Different from our aim to find out what concepts should be addressed and evaluated in an introductory CS assessment, many systematic reviews focus on the various teaching approaches and implementations during or after the course to support students' learning and to keep the retention rate. Though these systematic reviews are not directly related to our goal, they are still helpful for our research or action steps after our research.

An outstanding example is a systematic review published in 2014 by Arto et al. [64], describing literature about introductory programming teaching approaches. It provides an analysis of the effect that various interventions can have on the pass rates of introductory programming courses. Although the

study focuses on the behavioral interventions, such as peer-collaboration and in-class gamification, it offers constructive information about how to introduce technical concepts that are sorted out by our research.

Another notable example is a systematic review published in 2018 by Crow et al. [9], which synthesizes literature about tutoring systems in CS education. A key contribution of this paper is outlining how different features have been implemented and the prevalence of specific features across different Intelligent Programming Tutor (IPT) systems. Inspiring our research, this study attempts to add supplementary resources outside of programming tasks in IPT. The resources also play significant roles in helping students' understanding of course materials and increasing the effectiveness of the intelligent component. Therefore, Crow et al.'s review encourages more consideration in our research about adding extra resources other than technical programming concepts.

The third example is a systematic review released in 2021 done by Madeleine et al. [32], which intends to determine study behaviors of computing students and the approaches of educational design in shaping them. It provides a general study of students' behaviors during the course, including processes, strategies, habits, tactics, etc. Not mentioning any of the technical concepts of computing, this study reviews CS courses as a general educational context. Although it is not the research direction of our work, we can still apply some of the interventions in this study to implement our introductory CS concept inventory later in real life.

After getting inspiration from related work and making sure other systematic literature reviews could not answer our research questions, we started our data collection.

## V. Methodology

Our research followed the Systematic Literature Review (SLR) Guidelines of Kitchenham and Charters [23]. One of the critical points in conducting an SLR is that it uses specific and clear guidelines that directly relate to the reliability and reproducibility of the results generated by a sound and unbiased literature review.

Having formulated the research questions (stated in Section I), we identified the keywords to perform our search. Our initial set of keywords was: *assessment*, *placement*, *introduction to programming*, *introductory programming*, *novice programming*, *task assignment*, *beginner programming*, *CS0*, *CS0.5*, *CS.5*, *CS1*.

Subsequently, we created the following search string: *("assessment" OR "placement") AND ("CS.5" OR "CS0.5" OR "CS0" OR "CS1" OR "introduction to programming" OR "introductory programming" OR "beginner programming" OR "novice programming"))*

Miró Julià et al. [36] categorized and separated conferences in CS research from those in education based on the main focus of the conference. The majority of the relevant conferences they listed are indexed through ACM Digital Library (http://www.dl.acm.org). Therefore, we decided to use ACM Digital Library as the primary source of information collection and paper search. The first phase of the SLR was completed with a set of 1,743 papers.

Having concluded the first stage of collection, we began the second phase of the SLR. During this phase, all of the authors of this SLR were responsible for reviewing the title, abstract, and keywords of the selected papers and evaluating whether they met our inclusion or exclusion criteria:

- Inclusion criteria: peer-reviewed papers in conferences, workshops, journals, or book parts. These papers were written in English and were presented in a higher education environment;
- Exclusion criteria: We did not include papers that are not available online, or papers whose full text was unavailable, or papers that are systematic literature review, or those that analyze the results from case studies performed in high schools, middle schools or elementary schools.

We kept the papers that complied with our inclusion criteria and removed papers that met our exclusion criteria.

In the final phase, two authors reviewed one article by assessing the inclusion/exclusion criteria of the full text and evaluating if the full text was related to our research questions outlined in Section I. If no agreement was reached between the two authors, a further author would read the article and act as a presiding judge to decide whether the article would constitute part of the final selection. We created a spreadsheet to record our data extraction process from the first to the final phase and compile important paper information by category (region, school type, class size, case subject, programming language, assessment question type, and assessment content). Overall, a total of 48 studies were accepted as part of this research and fully analyzed.

## VI. Results

To check whether there are any trends or similarities in the assessment content, we collected information from the selected papers, and the summary is in Table I. The first column shows the content topics (alphabetically-ordered), and the other columns show in which course(s) the topic was listed. We categorized the selected papers based on what the paper explicitly said as course(s) being reported. Some papers reported more than one course, and when it was not possible to divide it into a single course, we reported it here as it was in the original paper. The main intention was to see similarities and differences among these introductory CS courses. We found that iterations, conditional statements, and variables are the top three assessment contents for both CS0 and CS1 courses. For the CS1 course, there are 22 papers' assessments including iterations (loops), 20 papers' assessments including conditional statements, and 14 papers' assessments including variables. For the CS0 course, six papers use iterations(loops), five papers use conditional statements, and four papers use variables. Though few papers study CS2 (CS2, CS1/CS2, CS0/CS1/CS2), the topic of iterations is counted three times, which is more than any other topic. The finding is plausible because iterations, conditional statements, and variables are the basics of introductory CS education, which should be tested in

TABLE I
ASSESSMENT CONTENTS

| Topics | Course | | | | | |
|---|---|---|---|---|---|---|
| | CS0 | CS1 | CS2 | CS0/CS1 | CS1/CS2 | CS0/CS1/CS2 |
| **Abstraction** | | [34], [43] | | | | |
| **Algorithm (sort/search/tree)** | [65] | [37], [34], [43], [45] | | | | |
| **Array** | [4], [68] | [8], [51], [14], [50], [29], [67], [27], [41] | [28] | | [47] | |
| **Assignment** | [4], [68], [57] | [34], [44], [30], [5], [15] | | | [47] | |
| **Boolean operators** | | [6], [34], [24], [14], [30], [5], [45] | | [12] | | |
| **Code Logic** | [4] | [37], [17], [34], [55] | | [12] | | [38] |
| **Code Testing** | | [37], [34], [43], [51], [14] | | | | |
| **Comparison** | | [8], [13], [34], [44], [30] | | | | |
| **Computational Thinking** | [7] | | | | | [16] |
| **Conditional statements** | [18], [4], [68], [57], [65] | [53], [8], [52], [56], [13], [31], [16] | | [12] | [47], [58] | [38] |
| | | [17], [19], [34], [43], [51], [14], [41] | | | | |
| | | [30], [67], [27], [15], [49], [66] | | | | |
| **Data structure** | | [53], [6], [34], [45], [40] | [46] | [12], [42] | [58] | [38] |
| **Documentation** | | [37], [34] | | | | |
| **Edit/Compile/Execute cycle** | | [56] | | | | |
| **Error Handling/Debug** | | [34], [51] | | | | |
| **Function** | [57], [65] | [31], [19], [34], [14], [30], [5], [41] | | [12], [42] | | |
| | | [27], [15], [66] | | | | |
| **Graph** | | | | | [26] | |
| **Hash table** | | [14] | | | [26] | |
| **Input / Output** | | [56], [34], [51], [50] | | | | |
| **Introductory Programming** | [57] | [34], [43], [24] | | | | |
| **Iterations (loops)** | [18], [7], [4], [68], [65] | [53], [8], [52], [31], [17], [6], [19] | [28] | [12] | | [38], [16] |
| | [57] | [34], [43], [55], [14], [30], [50], [5] | | | | |
| | | [29], [67], [45], [27], [40], [15], [49] | | | | |
| | | [41] | | | | |
| **List** | [4], [68] | [14], [45] | [46] | [12], [42] | | |
| **Methods** | [68], [65] | [34], [51], [14], [30], [50], [5] | | | | |
| **Object-oriented Design** | [4], [68], [57] | [53], [56], [34], [43], [51], [16], [41] | | [12] | | |
| **Operators** | [4], [57] | [41] | [28] | [42] | [47] | |
| **Parameters** | [4], [68], [57], [65] | [6], [34], [51], [14], [30], [41] | [28] | [12] | [47] | |
| **Pointers** | | [6], [44] | [46] | [12] | | |
| **Program design** | | [37], [51], [14] | | [12] | [26], [47] | |
| **Queue** | | | | [12] | | |
| **Recursion** | [57] | [6], [50], [45], [27], [49], [41] | | | | |
| **Sequence/Procedure** | [4] | [43], [51], [27] | | | [47] | |
| **Stack** | | [14] | [46] | [12] | | |
| **Variables** | [18], [7], [57], [65] | [53], [13], [17], [6], [19], [34], [51] | | [12], [42] | [47] | |
| | | [14], [30], [50], [5], [40], [49], [41] | | | | |

TABLE II
ASSESSMENT TYPE

| Course | Question Type | Paper |
|---|---|---|
| CS0 | Multiple Choice | [18], [65] |
| | Code Writing | [7], [4], [68] |
| | Code Explaining | [18] |
| CS1 | Multiple Choice | [53] [8] [52] [13] [37] [6] [19] [43] [24] |
| | | [44] [55] [51] [14] [30] [29] [45] [27] [66] |
| | | [60] [41] |
| | Code Writing | [53] [8] [62] [56] [13] [37] [6] [24] [44] |
| | | [55] [51] [14] [30] [5] [22] [67] [45] [27] |
| | | [40] [15] [41] |
| | Code Explaining | [53] [62] [56] [13] [55] [51] [30] [45] [15] |
| CS2 | Multiple Choice | [28] |
| | Code Writing | [28] |
| CS0/CS1 | Multiple Choice | [12] [42] |
| | Code Writing | [42] |
| CS1/CS2 | Multiple Choice | [47] [58] |

TABLE III
ASSESSMENT LANGUAGE

| Course | Language | Paper |
|---|---|---|
| CS0 | Scratch | [18], [7], [4] |
| CS1 | Python | [52] [31] [19] [34] [44] [30] [5] [45] |
| | Java | [52] [56] [34] [43] [51] [14] [50] [5] |
| | | [22] [29] [45] |
| | C/C++ | [52] [13] [6] [5] |
| | Pseudo-codes | [12] [55] [60] [41] |
| CS1/CS2 | C | [26] |

TABLE IV
ASSESSMENT ORIGIN

| Course | Region | Paper |
|---|---|---|
| CS0 | U.S. | [18], [7], [4], [57], [65] |
| | Germany | [68] |
| CS1 | U.S. | [8] [62] [56] [37] [43] [24] [67] [27] |
| | | [40] [66] [60] [41] |
| | Australia | [53] [62] [55] [30] [29] |
| | New Zealand | [52] [31] [22] |
| | South Africa | [17] |
| | Europe | [14] [42] [15] [49] [41] |
| | Philippines | [41] |
| | Brazil | [5] |
| | Canada | [44] |
| | International | [19] [51] [50] [45] [27] [15] |
| CS2 | U.S. | [28] |
| CS0/CS1 | U.S. | [12] |
| | International | [42] |
| CS1/CS2 | U.S. | [59] [26] [58] |
| | International | [47] |
| CS0/CS1/CS2 | U.S. | [16] |
| | International | [38] |

any level of CS classes. Except for the most popular ones, we also found that function, parameter, and object-oriented design are commonly-used contents in the assessments. Also, for a few papers that talk about more than one course (CS0/CS1, CS1/CS2, CS0/CS1/CS2), most of them cover wide-range

topics. For example, Dierbach et al.'s [12] paper, reporting CS0/CS1, mentioned 14 assessment topics, from the basic ones, such as iterations and conditional statements, to the high-level ones, such as object-oriented design and program design; though Mühling et al.'s [38] paper, reporting CS0/CS1/CS2, only covered three topics, they also included the basics, conditional statements, and the advanced, code logic and data structure.

To understand which type of question the assessments used most frequently, we made Table II. We divided the question types into three categories - Multiple Choice, Code Writing, and Code Explanation. Then we related these categories to the courses being reported. Again, some papers reported more than one course, so we kept the information as it was reported. According to Table II, Multiple Choice and Code Writing are more frequently used than Code Explaining for all the introductory courses. For both CS0 and CS1 courses, though a bit more papers included Code Writing than Multiple Choice in their assessments, the numbers are very close, showing that Multiple Choice and Code Writing are about equally important. They complement each other, since Multiple Choice is the easiest to grade while Code Writing assesses students' CS abilities more comprehensively. We could not discern any trends for CS2, CS0/CS1, and CS1/CS2 courses as their sample sizes were too small.

To have an overview of what programming languages are used in assessments and where these assessments come from, we made Table III and Table IV. Scratch, Python, Java, C/C++, Pseudo-codes were mentioned in the papers we reviewed, so we used them as programming language categories and related corresponding courses and papers to them. We used the same method to categorize Region, adding a new region when the paper raised a new one and then assigning courses/papers. According to Table III, we found that most papers reporting CS0 used scratch as their assessment language. For the CS1 course, papers employed Python, Java, C, and pseudo-codes as their assessment languages, and Java was the most popular one. Table IV shows that the U.S. was involved in assessment design and application for all the introductory CS courses. The U.S. (23 assessments) was also a region that had the most assessments created there, which was followed by International (9 assessments) and Europe (6 assessments). Note that we only attributed a paper to an assessment language or a region if there was an explicit mention on the paper.

Finally, to answer our third research question, we paid attention to existing assessments/concept inventories from the selected papers. Tew et al. [60] developed the Foundational CS1 Assessment (FCS1), the first assessment instrument for introductory computer science concepts, which was validated with an empirical study that considered interview data, statistical analysis, and multi-faceted argument. It indicated that pseudo-code was "an appropriate mechanism for achieving programming language independence." Parker et al. [41] developed the Second CS1 Assessment (SCS1), which is an isomorphic version of the Foundations CS1 Assessment (FCS1). They replicated FCS1 "to enable free use by a broader research community" and validated the success of the replication. SCS1 includes fundamentals, logical operators, conditionals, loops, arrays, functions, recursion, and object-oriented basics as its assessment concepts. Porter et al. [46] designed and validated a concept inventory for Basic Data Structures (BDSI), which could be used in a CS2 course. The concepts include linked list, pointer, binary search tree, stack, etc. The questions were constructed with the intention of assessing students' understanding of different data structures, various data structures operations (insertion, deletion, search, etc.), abstract data types, and so on. Peteranetz et al. [43] developed and validated a test of computational thinking concepts and compared different scoring methods for the assessment. The assessment consists of 26 multiple-choice questions and is used to evaluate students' understanding of the main computing concepts which are basic object-oriented programming (classes, instances, events, and methods), basic program control constructs (sequence, selection, and iteration), and basic sorting and searching algorithms on arrays. Caceffo et al. [6] developed the concept inventory to reveal students' misconceptions of CS topics. They identified misconceptions related to function parameter use and scope, variables, recursion, iteration, structures, pointers, and Boolean expressions by analyzing open-ended exams and instructor interviews. To our knowledge and research at this moment, existing assessments/concept inventories focus on either CS1 or CS2. There is no assessment/concept inventory that encompasses the contents of CS0, CS1, and CS2 all in one.

## VII. Threats to validity

In a systematic literature review, there is always a threat that the words and syntax used as the search string were not the best ones possible. Though we have tried to be as comprehensive as possible, we cannot assert that this systematic literature review covers all possible papers related to the introductory CS assessment. After performing all the steps of an SLR, we ended up with 48 primary studies that presented information regarding assessments in introductory CS courses, so the amount of published work on assessments is much fewer than that focusing on how to develop introductory CS courses. It is also difficult to look at research literature and expect a fully comprehensive look at all the assessments that are done or exist. On the one hand, many institutions/researchers do their day-to-day business and do not necessarily publish what they do and how they do it. On the other hand, relying on publications presenting an assessment results differs from analyzing the assessment itself. In publications, authors publish according to their goals, and the granularity or detail of the assessments do not make them all exactly comparable to the other. Not all the papers reported contexts or assessment details that we analyzed in Section VI.

## VIII. Conclusion

Though there might be quality concerns, we can answer our research questions and generally apply them to most

institutions after doing the literature review. Iterations, conditional statements, and variables are the most popular and important topics. Multiple-choice and code writing are the most significant question types for introductory CS course assessments. There are few assessments for CS1 or CS2 (FCS1, SCS1, BSDI), but there is no assessment or concept inventory that encompasses CS0, CS1, and CS2 topics.

Stopping at answering our research questions may not be enough for the real world. As introductory CS courses are being taught globally, there is a need to see more representations from different countries and find a way to be more inclusive as a community. Moreover, we can contribute more to building a CS introductory assessment or concept inventory that encompasses CS0/CS1/CS2 materials. In this way, higher-education institutions can direct students to the right CS intro-level class, improve students' experience in CS, and decrease the CS major/minor attrition rate by helping students build better CS foundations.

## REFERENCES


[1] National center for education statistics, Aug 2015.

[2] C. Alvarado, Z. Dodds, and R. Libeskind-Hadas. Increasing women's participation in computing at harvey mudd college. *acm Inroads*, 3(4):55–64, 2012.

[3] T. Beaubouef and J. Mason. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, 2005.

[4] M. Berges and P. Hubwieser. Evaluation of source code with item response theory. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 51–56, 2015.

[5] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo. Identifying and validating java misconceptions toward a cs1 concept inventory. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, pages 23–29, 2019.

[6] R. Caceffo, S. Wolfman, K. S. Booth, and R. Azevedo. Developing a computer science concept inventory for introductory programming. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 364–369, 2016.

[7] V. Cateté, E. Snider, and T. Barnes. Developing a rubric for a creative cs principles lab. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 290–295, 2016.

[8] B. Chen, S. Azad, R. Haldar, M. West, and C. Zilles. A validated scoring rubric for explain-in-plain-english questions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 563–569, 2020.

[9] T. Crow, A. Luxton-Reilly, and B. Wuensche. Intelligent tutoring systems for programming education: A systematic review. ACE '18, page 53–62, New York, NY, USA, 2018. Association for Computing Machinery.

[10] N. B. Dale. Most difficult topics in cs1: results of an online survey of educators. *ACM SIGCSE Bulletin*, 38(2):49–53, 2006.

[11] S. Davies, J. A. Polack-Wahl, and K. Anewalt. A snapshot of current practices in teaching the introductory programming sequence. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 625–630, 2011.

[12] C. Dierbach, B. Taylor, H. Zhou, and I. Zimand. Experiences with a cs0 course targeted for cs1 success. *ACM SIGCSE Bulletin*, 37(1):317–320, 2005.

[13] D. D'Souza, J. Sheard, J. Harland, A. Carbone, and M.-J. Laakso. Can computing academics assess the difficulty of programming examination questions? In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, pages 160–163, 2012.

[14] R. Duran, J.-M. Rybicki, J. Sorva, and A. Hellas. Exploring the value of student self-evaluation in introductory programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 121–130, 2019.

[15] R. S. Duran, J.-M. Rybicki, A. Hellas, and S. Suoranta. Towards a common instrument for measuring prior programming knowledge. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '19, page 443–449, New York, NY, USA, 2019. Association for Computing Machinery.

[16] A. Goncharow, M. Mcquaigue, E. Saule, K. Subramanian, J. Payton, and P. Goolkasian. Mapping materials to curriculum standards for design, alignment, audit, and search. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 295–301, 2021.

[17] L. A. Gouws, K. Bradshaw, and P. Wentworth. Computational thinking in educational activities: an evaluation of the educational game lightbot. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 10–15, 2013.

[18] S. Grover. Designing an assessment for introductory programming concepts in middle school computer science. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 678–684, 2020.

[19] J. Henry and B. Dumas. Developing an assessment to profile students based on their understanding of the variable programming concept. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 33–39, 2020.

[20] M. Hertz. What do "cs1" and "cs2" mean? investigating differences in the early courses. SIGCSE '10, page 199–203, New York, NY, USA, 2010. Association for Computing Machinery.

[21] A. f. C. M. A. Joint Task Force on Computing Curricula and I. C. Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. Association for Computing Machinery, New York, NY, USA, 2013.

[22] N. Kasto and J. Whalley. Measuring the difficulty of code comprehension tasks using software metrics. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, pages 59–65, 2013.

[23] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering, 2007.

[24] B. Kokensparger. Do students lie on placement surveys? an action research study. *Journal of Computing Sciences in Colleges*, 32(5):156–162, 2017.

[25] T. Koulouri, S. Lauria, and R. D. Macredie. Teaching introductory programming: A quantitative evaluation of different approaches. *ACM Transactions on Computing Education (TOCE)*, 14(4):1–28, 2014.

[26] S. Krause-Levy, S. Valstar, L. Porter, and W. G. Griswold. Exploring the link between prerequisites and performance in advanced data structures. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 386–392, 2020.

[27] W. M. Kunkle and R. B. Allen. The impact of different teaching approaches and languages on student learning of introductory programming concepts. *ACM Transactions on Computing Education (TOCE)*, 16(1):1–26, 2016.

[28] L. Layman, Y. Song, and C. Guinn. Toward predicting success and failure in cs2: A mixed-method analysis. In *Proceedings of the 2020 ACM Southeast Conference*, ACM SE '20, page 218–225, New York, NY, USA, 2020. Association for Computing Machinery.

[29] R. Lister. One small step toward a culture of peer review and multi-institutional sharing of educational resources: a multiple choice exam for first semester programming students. In *Conferences in Research and Practice in Information Technology Series*, 2005.

[30] R. Lister, C. Fidge, and D. Teague. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *Acm sigcse bulletin*, 41(3):161–165, 2009.

[31] R. Lobb and J. Harlow. Coderunner: A tool for assessing computer programming skills. *ACM Inroads*, 7(1):47–51, 2016.

[32] M. Lorås, G. Sindre, H. Trætteberg, and T. Aalberg. Study behavior in computing education—a systematic literature review. 22(1), 2021.

[33] A. Luxton-Reilly, I. Albluwi, B. A. Becker, M. Giannakos, A. N. Kumar, L. Ott, J. Paterson, M. J. Scott, J. Sheard, and C. Szabo. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pages 55–106, 2018.

[34] A. Luxton-Reilly, B. A. Becker, Y. Cao, R. McDermott, C. Mirolo, A. Mühling, A. Petersen, K. Sanders, and J. Whalley. Developing assessments to determine mastery of programming fundamentals. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*, pages 47–69, 2018.

[35] C. Marling and D. Juedes. Cs0 for computer science majors at ohio university. SIGCSE '16, page 138–143, New York, NY, USA, 2016. Association for Computing Machinery.

[36] J. Miró Julià, D. López, and R. Alberich. Education and research: evidence of a dual life. In *Proceedings of the ninth annual international conference on International computing education research*, pages 17–22, 2012.

[37] B. B. Morrison, B. Coutts, and T. Gallagher. Can your students pass this test? In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 943–944, 2021.

[38] A. Mühling, A. Ruf, and P. Hubwieser. Design and first results of a psychometric test for measuring basic programming abilities. In *Proceedings of the workshop in primary and secondary computing education*, pages 2–10, 2015.

[39] U. Nikula, O. Gotel, and J. Kasurinen. A motivation guided holistic rehabilitation of the first programming course. *ACM Trans. Comput. Educ.*, 11(4), 2011.

[40] L. Ott, B. Bettin, and L. Ureel. The impact of placement in introductory computer science courses on student persistence in a computing major. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, page 296–301, New York, NY, USA, 2018. Association for Computing Machinery.

[41] M. C. Parker, M. Guzdial, and S. Engleman. Replication, validation, and use of a language independent cs1 knowledge assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, page 93–101, New York, NY, USA, 2016. Association for Computing Machinery.

[42] J. Parkinson, Q. Cutts, and S. Draper. Relating spatial skills and expression evaluation. In *United Kingdom & Ireland Computing Education Research conference.*, pages 17–23, 2020.

[43] M. S. Peteranetz, P. M. Morrow, and L.-K. Soh. Development and validation of the computational thinking concepts and skills test. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 926–932, 2020.

[44] A. Petersen, M. Craig, and P. Denny. Employing multiple-answer multiple choice questions. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 252–253, 2016.

[45] A. Petersen, M. Craig, and D. Zingaro. Reviewing cs1 exam question content. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 631–636, 2011.

[46] L. Porter, D. Zingaro, S. N. Liao, C. Taylor, K. C. Webb, C. Lee, and M. Clancy. Bdsi: A validated concept inventory for basic data structures. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, ICER '19, page 111–119, New York, NY, USA, 2019. Association for Computing Machinery.

[47] K. Sanders, M. Ahmadzadeh, T. Clear, S. H. Edwards, M. Goldweber, C. Johnson, R. Lister, R. McCartney, E. Patitsas, and J. Spacco. The canterbury questionbank: Building a repository of multiple-choice cs1 and cs2 questions. In *Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports*, pages 33–52, 2013.

[48] C. Schulte and J. Bennedsen. What do teachers teach in introductory programming? In *Proceedings of the second international workshop on Computing education research*, pages 17–28, 2006.

[49] C. Schulte and J. Bennedsen. What do teachers teach in introductory programming? In *Proceedings of the Second International Workshop on Computing Education Research*, ICER '06, page 17–28, New York, NY, USA, 2006. Association for Computing Machinery.

[50] J. Sheard, A. Carbone, D. Chinn, T. Clear, M. Corney, D. D'Souza, J. Fenwick, J. Harland, M.-J. Laakso, D. Teague, et al. How difficult are exams? a framework for assessing the complexity of introductory programming exams. 2013.

[51] J. Sheard, A. Carbone, D. Chinn, M.-J. Laakso, T. Clear, M. de Raadt, D. D'Souza, J. Harland, R. Lister, A. Philpott, et al. Exploring programming assessment instruments: A classification scheme for examination questions. In *Proceedings of the seventh international workshop on Computing education research*, pages 33–38, 2011.

[52] J. Sheard, J. Dermoudy, D. D'Souza, M. Hu, and D. Parsons. Benchmarking a set of exam questions for introductory programming. In *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*, pages 113–121, 2014.

[53] S. Shuhidan, M. Hamilton, and D. D'Souza. A taxonomic study of novice programming summative assessment. In *Proceedings of the Eleventh Australasian Conference on Computing Education-Volume 95*, pages 147–156, 2009.

[54] N. Sigurdson and A. Petersen. Student perspectives on mathematics in computer science. page 108–117, New York, NY, USA, 2017. Association for Computing Machinery.

[55] S. Snowdon. Explaining program code: giving students the answer helps-but only just. In *Proceedings of the seventh international workshop on Computing education research*, pages 93–100, 2011.

[56] C. W. Starr, B. Manaris, and R. H. Stalvey. Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. *ACM SIGCSE Bulletin*, 40(1):261–265, 2008.

[57] C. Stewart-Gardiner. Improving the student success and retention of under achiever students in introductory computer science. *Journal of Computing Sciences in Colleges*, 26(6):16–22, 2011.

[58] C. Taylor, M. Clancy, K. C. Webb, D. Zingaro, C. Lee, and L. Porter. The practical details of building a cs concept inventory. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 372–378, New York, NY, USA, 2020. Association for Computing Machinery.

[59] M. F. Tennyson and M. Beck. A study of knowledge retention in introductory programming courses. *Journal of Computing Sciences in Colleges*, 33(4):13–20, 2018.

[60] A. E. Tew and M. Guzdial. The fcs1: A language independent assessment of cs1 knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, page 111–116, New York, NY, USA, 2011. Association for Computing Machinery.

[61] C. The Joint Task Force on Computing Curricula. Computing curricula 2001. *J. Educ. Resour. Comput.*, 1(3es):1–es, sep 2001.

[62] E. Thompson, A. Luxton-Reilly, J. L. Whalley, M. Hu, and P. Robbins. Bloom's taxonomy for cs assessment. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 155–161, 2008.

[63] A. Verno. Supporting k-12 computer science education. *J. Comput. Sci. Coll.*, 23(3):145–146, jan 2008.

[64] A. Vihavainen, J. Airaksinen, and C. Watson. A systematic review of approaches for teaching introductory programming and their influence on success. ICER '14, page 19–26, New York, NY, USA, 2014. Association for Computing Machinery.

[65] D. Weintrop and U. Wilensky. Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*, ICER '15, page 101–110, New York, NY, USA, 2015. Association for Computing Machinery.

[66] J. Wrenn, S. Krishnamurthi, and K. Fisler. Who tests the testers? In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, ICER '18, page 51–59, New York, NY, USA, 2018. Association for Computing Machinery.

[67] L. Zavala and B. Mendoza. Precursor skills to writing code. *Journal of Computing Sciences in Colleges*, 32(3):149–156, 2017.

[68] R. Zhi, M. Chi, T. Barnes, and T. W. Price. Evaluating the effectiveness of parsons problems for block-based programming. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 51–59, 2019.