Using an Evidence-Centered Design Approach to Examine the Alignment of Computer Science Curricula with Standards

Daisy Rutstein, edCount, LLC, dazerw@yahoo.com
Satabdi Basu, Hui Yang, Arif Rachmatullah, Carol Tate
"satabdi.basu, hui.yang, arif.rachmatullah, carol.tate"@sri.com
SRI International
Steven McGee, The Learning Partnership, lponline@mac.com

Abstract: In the early stages of K-12 Computer Science (CS) curriculum development, standards were not yet established, and the primary objective, especially in younger grades, was to spark students' interest in CS. While this remains a vital goal, the development of the CS standards underscores the importance of standards-aligned curriculum, ensuring equitable, content rich CS education for all students. We show that standards alignment is most useful when it includes details about which aspects of the standards a curriculum aligns with. This paper describes our process of decomposing five middle school CS standards into granular learning targets using an evidence-centered design approach and mapping the learning targets onto individual lessons from one widely popular middle school CS curriculum. We discuss the potential implications of this work on curriculum design, curriculum selection, and teacher professional learning.

Introduction

As the demand for computer science (CS) professionals and computationally literate citizens grows, the number of CS offerings in K-12 schools has significantly expanded (Vegas & Fowler, 2020). In 2022, 53% of high schools in the United States (US) offered foundational CS courses (K-12 Computer Science Framework Steering Committee et al., 2016). In at least 15 US states, middle school students are expected to complete one or more CS courses (CSTA, Code.org, & ECEP Alliance, 2022). In addition to offering stand-alone CS courses, efforts are also being made to integrate CS into other disciplines, STEM and non-STEM (Fancsali et al., 2018; Vegas & Fowler, 2020). Although this development seems promising, there are still significant disparities in access to quality CS instruction in the US (Martin, McAlear, & Scott, 2015) between states, school districts, and even among schools in the same district, in some cases. Many factors contribute to these disparities such as the decentralized nature of the US education system, state and district-level policies, levels of funding, and availability of qualified CS teachers. These disparities mean that the quality of CS curricula used by students may vary significantly because curriculum decisions are being made at district, school, or even classroom levels. Quality CS curriculum is a cornerstone of equitable CS education and determines what CS constructs students will learn and how they will learn them. The goal of a well-designed curriculum should be to ensure that students have a range of opportunities to gain the knowledge, skills, and abilities required by the standards in that discipline (The Center on Standards Assessment Implementation, 2018). In the US, the Computer Science Teachers Association (CSTA) released a set of CS content standards in 2017 defining what students should know and be able to do in different grade bands, playing a pivotal role in ensuring consistent quality in CS curriculum and instruction. Most state CS standards in the US are modeled on the CSTA standards. Standards can be particularly helpful for new CS teachers, who are more likely to teach historically marginalized students in CS or STEM in general (Fletcher et al., 2021), by providing them with clear guidelines for what their students should be learning.

Ideally, standards alignment can provide an objective way to compare different curricula for the same grade levels. However, mapping CS curricula to standards is complex as the standards language is often broad and open to interpretation (Basu et al., 2022). Standards outline the target content and practices and, in theory, support a coherent approach across grade levels, but their high-level descriptions make it challenging to use them as a guide for curriculum selection. When CS curricula specify standards alignment, it is not very meaningful without a description of how lessons align with standards and which aspects of standards lessons align with. Therefore, we contend that there is a need to unpack CS standards into fine-grained learning targets (LTs) and map CS curricula to the granular LTs instead of the broader standards. This would allow teachers and administrators to make more informed curriculum choices, ultimately enhancing student learning and making learning opportunities more equitable. Evidence centered design (ECD; Mislevy & Haertel, 2006) is an approach typically used for assessment design where a broad domain is analyzed and decomposed into granular skills that can be assessed; we propose using the same approach for examining curriculum alignment with standards. To this end, our paper seeks to address the following research questions, specifically in the context of middle school CS:

RQ1. How can broad CS standards be unpacked into granular learning targets to inform instruction?
RQ2. How does using an ECD approach to measure standards alignment of a middle school CS curriculum compare against the alignment information provided by the curriculum developers?

Background

CS curriculum and standards alignment

In this paper, we base our work on the following definitions: "Standards indicate what students should know and be able to do within a particular content area, while curriculum shapes how students will gain the knowledge, skills, and abilities as described in the standards. Alignment is the process of ensuring that the specified curriculum is consistent with enabling students to reach the milestones outlined in the standards" (The Center on Standards Assessment Implementation, 2018, p. 1). In some countries (e.g., UK, New Zealand, Israel), the goal of universal access to CS instruction has been accomplished with a national curriculum (Fowler & Vegas, 2021; Gal-Ezer & Stephenson, 2014; Webb et al., 2017). In the US, however, coverage of CS standards in still largely dependent on curriculum selection and the knowledge and choices of the teacher. A few states have curriculum frameworks in place based on CS standards, but full curricular pathways based on the standards do not yet exist. Also, the vast majority of teachers do not work with district-designated curriculum materials (Banilower et al., 2018). Instead, they search for materials on the internet, many of which were developed in advance of standards (EdReports, 2022), and have very limited research on their efficacy. Similar to the curricular landscape in more established disciplines prior to standards in the early 1990s, the content of CS classes is currently widely variable in the US.

Interestingly, there has been increasing interest among CS curriculum providers to specify standards alignment and validate it externally. CSTA, which spearheaded the development of the CS standards, currently validates the standards-alignment of K-12 curricular materials submitted by curriculum providers using a fee-based model involving independent reviews by two or more trained K-12 CS educators (CSTA, n.d.). CSTA requires curriculum providers to complete a crosswalk template, indicating the alignment status for each standard as “Fully Aligned,” “Partially Aligned,” or “Not Aligned.” For standards indicated as aligned, CSTA asks for 1-4 examples of alignment from the curriculum and an optional description of how the standard is addressed and which aspects of the standard are addressed. CSforAll also reviews CS curricular alignment, but with the K-12 CS Framework (K-12 Computer Science Framework Steering Committee et al., 2016), through their AlignCS program (CS for ALL, 2023). Experienced CS teachers conduct a thorough review of whether submitted curricula align with the specified concepts of the CS Framework. Upon successful review and approval, the curriculum appears in the CSforAll curriculum directory, displaying a “K-12 CS Framework Aligned” badge.

The landscape of middle school CS education currently includes several CS curricula, some common ones being Code.org's CS Discoveries, Google's CS First, CodeHS's middle school CS pathways, Project STEM's CS Explorations, and CS Unplugged by Colorado School of Mines. Most of these curricula claim to be aligned with either the CS framework or the CSTA standards. However, they do not describe how their curricula address the CS standards or concepts in the CS Framework, or which aspects of the standard their curricula address. This lack of detailed information makes the curriculum selection process challenging for school leaders and teachers.

Overview of the CS Discoveries curriculum

In this paper, to illustrate use of the ECD approach to measure standards alignment of middle school CS curricula, we focus on code.org's Computer Science Discoveries (CSD) curriculum. CSD is one of the most widely embraced middle school CS curricula in the US, and thus seems like an appropriate choice for starting to explore standards alignment of CS curricula. CSD is a versatile introductory curriculum, designed to captivate and engage middle school students with no, or limited, prior CS background. The Code.org 2021 Annual Report (2021) mentions that 788,377 students began using CSD that academic year. We focus on six units in the CSD 23’-24’ version, namely, Unit 1 - Problem Solving and Computing (13 lessons); Unit 2 - Web Development (21 lessons); Unit 3 - Interactive Animations and Games (28 lessons); Unit 4 - The Design Process (21 lessons); Unit 5 - Data and Society (16 lessons); Unit 6A - Creating Apps with Devices - Circuit Playground (19 lessons). CSD employs a range of programming environments, including code.org's App Lab and Game Lab, where students can toggle between a block-based and text-based version of JavaScript. Alongside programming tools, CSD offers extensive teacher PD opportunities, lesson plans, videos, slides, and assessments, all geared toward aiding teachers in implementing the curriculum. Teachers can customize the curriculum timeline based on their students’ needs.

Methods

In this paper, we focus on five middle school CSTA CS content standards (CSTA, 2017) from the ‘Algorithms and Programming' (AP) strand (2-AP-10 through 2-AP-14) covering the concepts of algorithms, decomposition, variables, combined control structures, and procedures. Middle school is when many schools start introducing CS. Student engagement in CS is a focus at this age level, while student proficiency on standards is still less understood and ill-defined for these grades. AP is currently the most commonly covered CS strand and often the only strand covered in some middle schools in the U.S. (strands such as computing systems and networks and the internet are often skipped). We selected five out of 10 middle school AP standards that focus on programming concepts (other standards focus on program development skills such as incorporating feedback from team members, providing attribution, and documenting code). We report on two major activities - decomposing the target standards into granular learning targets and exploring the alignment of the targets to CSD lessons.

Decomposing CS content standards

We used an Evidence-Centered Design (ECD) approach (Mislevy & Haertel, 2006) to break down the standards into meaningful LTs. ECD is a systematic multi-step approach to assessing broad domains by breaking them down into knowledge, skills and abilities and designing tasks to elicit evidence of mastery on individual skills and knowledge components. While ECD is generally considered an approach for developing assessments, the domain modeling done as part of the ECD process results in a set of LTs that is useful for curriculum, instruction, and assessment. In our work, we used this approach to develop a set of LTs for each standard that could then be used to check the alignment of curricular units to the standards. The LTs were developed by a group of researchers with expertise in K-12 CS education and the ECD approach who referenced existing literature discussing the scope of CS standards and computational thinking (CT) concepts, desired student performance for middle school CS, CS constructs targeted for middle school CS and CT assessment development, learning trajectories for CS concepts, and student challenges and misconceptions about CS concepts (e.g., Basu et al., 2021; Rich et al., 2017; Grover & Basu, 2017; Sirkiä & Sorva, 2012). The LTs do not follow any specific sequence or learning trajectory and when taken together constitute the full scope of the standard. LTs ranged in level of complexity; some focused on knowledge students need to have while others focused on things students should be able to do to demonstrate proficiency on a standard. Once we developed the initial set of LTs for each of the five standards, we conducted an expert review (Baxter & Glaser, 1997) with six external reviewers. Reviewers brought rich expertise and experience in K-12 CS education research and/or instructional design and included university professors, senior researchers at R&D institutes, and a STEAM and CS coordinator from a K-8 school. Reviewers independently reviewed the LTs and rated them based on how well they covered the standards, their level of granularity, and their appropriateness for middle school. Reviewers also provided open-ended feedback and sometimes offered rewording suggestions. We revised the LTs based on the expert review feedback.

Examining standards alignment

Once the set of LTs were finalized, we examined the alignment between our five target standards and Units 1-6A of the CSD curriculum (we left out the last two optional units) by focusing on the alignment of LTs to individual CSD lessons. We included all lessons, including those focused on student projects. For each standard, we studied its alignment with the CSD curriculum along three dimensions - number of lessons that aligned with the standard in some way, the mapping between specific LTs for the standard and specific CSD lessons, and the strength of alignment between aligned LTs and lessons. For each lesson, we started by noting the standards that the curriculum developers identified as aligned with the lesson. Next, we went through the set of activities in the lesson to determine if any of them relate to any of the LTs across all five standards. We identified two levels of alignment, direct and indirect. A direct alignment indicates that an activity provides students with instruction that is directly related to the LT. For example, when an activity involved identifying problems with variable names, this was directly aligned to our 2-AP-11 LT on naming of variables. For an indirect alignment, the activity may or may not provide students with opportunities to engage with the LT, depending on how the teacher frames the activity or how students respond to open-ended activities. For example, in lessons where students are given the freedom to pick their own projects, alignment with particular LTs that focus on control structures may depend on which control structures, if any, students decide to use. For an indirect alignment, students may fully engage with the activities in a lesson without being exposed to the concepts in the LTs. We used a lesson's alignment to LTs to determine its overall alignment to a standard. If a given lesson was directly aligned to any LT for a standard, we stated that the overall standard was directly aligned to the lesson. If the only LT alignments for a standard were indirect alignments, we called the overall standard indirectly aligned to the lesson. If no LTs for a standard were specified as either direct or indirect alignment, the standard was deemed not aligned to the lesson. For each standard, two researchers examined the alignment of the corresponding LTs with CSD units, lessons, and activities. Inter-rater reliability was high among the researchers and any discrepancies in coding were discussed with the first and second authors of this paper who helped resolve the discrepancies.

Findings

Unpacking the five target standards

Here, we present our list of LTs for our five target standards, which we then use to examine the standards alignment of the CSD curriculum. As noted earlier, the LT numbering does not indicate any learning trajectory.

Unpacking 2-AP-10 on algorithms. This standard comprises knowledge of algorithms and their representations using flowcharts and pseudocode, ability to create these algorithmic representations, ability to trace and compare algorithms, and ability to test and debug algorithms.
LT1. Knowledge that an algorithm is a step-by-step, ordered set of instructions for solving a problem, and in order to be computer-understandable, the instructions must be precise and unambiguous.
LT2. Knowledge that pseudocode is an informal way to describe code without following any specific programming language and can be used to plan out code before programming.
LT3. Knowledge that a flowchart is a diagrammatic representation of an algorithm that specifies a step-by-step way to complete a task.
LT4. Ability to trace an algorithm (in the form of pseudocode or a flowchart) and describe its behavior or output when given a specific set of inputs.
LT5. Ability to recognize relevant information from a problem to identify possible inputs to an algorithm, decision points that may require branching, test cases, stopping conditions, and constraints such as cost or time.
LT6. Ability to select and/or create appropriate representations to plan a problem solution that handles the desired range of inputs and is able to deal with edge cases.
LT7. Ability to generate multiple algorithms (flowchart or pseudocode) to solve a problem.
LT8. Ability to compare the trade-offs between different algorithms or approaches to problem solving based on certain evaluation criteria or constraints.
LT9. Ability to identify meaningful test cases (including edge cases) for testing an algorithm.
LT10. Ability to test and debug algorithms using a systematic and iterative process.

Unpacking 2-AP-11 on variables. The scope of this standard includes creating, naming and initializing variables with different data types, manipulating variable values, and combining variables with other CS constructs such as loops and conditionals.
LT1. Ability to determine what variables are required in a program and what data type the variables should be to achieve the goals of the computational solution.
LT2. Ability to choose meaningful yet concise variable names.
LT3. Ability to create required variables and assign appropriate data types to the variables.
LT4. Ability to initialize variables.
LT5. Ability to assign scope of variables and distinguish between global and local variables.
LT6. For variables representing simple data types, ability to create and/or debug code to assign values to variables and/or perform operations on values of variables.
LT7. For variables representing compound data types, ability to create and/or debug code (without control structures) to assign values to variables, find properties of variables, and/or perform operations on variable values.
LT8. Ability to describe how the value of a variable changes in a given code segment without a control structure such as a loop or a conditional statement.
LT9. Ability to create and/or debug code to assign or update variable values repeatedly or under specific conditions, using control structures such as loops and conditionals.
LT10. Ability to describe how the value of a variable changes in a given code segment containing one or more control structures such as loops and/or conditional statements.
LT11. Ability to create and/or debug code that uses a variable to specify the condition in a conditional statement or to specify the number of times a loop repeats.
LT12. Ability to interpret the output of code that uses a variable to specify the condition in a conditional statement or to specify the number of times a loop repeats.

Unpacking 2-AP-12 on combined control structures. Students are expected to be able to create, debug, and predict the output of programs containing combined control structures such as nested loops, nested conditionals, compound conditionals, repeated conditionals and control structures within procedures.
LT1. Ability to create a nested loop and/or debug a given nested loop to represent a given scenario.
LT2. Ability to identify or predict the output of a nested loop.
LT3. Ability to create and/or debug nested conditionals to represent a given narrative description.

LT4. Ability to predict the output of nested conditionals.
LT5. Ability to create and/or debug a conditional statement that includes logical operators (AND, OR, NOT).
LT6. Ability to predict the output of a conditional statement that uses logical operators (AND, OR).
LT7. Ability to create and/or debug a conditional statement that is evaluated repeatedly (e.g., repeat-until).
LT8. Ability to identify the output of a conditional statement that is evaluated repeatedly (e.g., repeat-until).
LT9. Ability to create and/or debug a procedure that is called within a different control structure.
LT10. Ability to identify the output of a procedure that is called within a different control structure.

Unpacking 2-AP-13 on decomposition. This standard includes knowledge of the benefits of decomposition, ability to engage in decomposing problems, and ability to re-use decomposed parts of solutions.
LT1. Ability to describe benefits to using decomposition for individual or collaborative problem solving.
LT2. Ability to identify intermediate goals and break up a problem into meaningful sub-problems.
LT3. Ability to identify connections and dependencies between different solutions to sub-problems in order to combine these solutions to solve a problem as a whole.
LT4. Ability to identify whether a decomposition (set of sub-problems or sub-tasks) is sufficient to address the bigger problem or task completely and coherently.
LT5. Ability to describe how to solve a problem using solutions to existing problems if appropriate.

Unpacking 2-AP-14 on Procedures. This standard expects students to be able to identify when to use procedures (or functions), create procedures, call procedures and interpret the output of procedures.
LT1. Ability to identify scenarios where a procedure would be appropriate.
LT2. Knowledge of existing system-defined procedures in a programming language and their functionalities.
LT3. Ability to concisely and meaningfully name a procedure based on what it does.
LT4. Ability to create and/or debug a procedure with/without input parameters and with/without a return value.
LT5. Ability to call a procedure without input parameters.
LT6. Ability to call a procedure with input parameters using constants or variables as part of the procedure call.
LT7. Ability to assign the return value of a procedure to an appropriate variable when calling the procedure.
LT8. Ability to predict the output of code containing a procedure with/without input parameters, with/without return values.

Mapping CS Discoveries to CS standards

We found that all CSD units covered at least one of our five target standards from the AP strand. Units that did not focus on programming (units 1, 2, 4 and 5) were only aligned with 2-AP-10 (algorithms) and/or 2-AP-13 (decomposition) and were not aligned with the programming standards (2-AP-11, 2-AP-12, 2-AP-14). In contrast, units 3 and 6 focused on programming and were aligned with programming and non-programming standards. Figure 1 illustrates the alignment of CSD units 1-6 with our five target standards. Part a shows the number of lessons in each unit with direct alignment while part b also includes lessons with indirect alignment.

In Unit 1 (Problem Solving and Computing), students are introduced to computers and problem-solving concepts. CSD indicates the unit is aligned primarily with elementary school standards (not covered in our target standards) and is only aligned to one of our five target standards, 2-AP-10. We also found evidence of this alignment but found that this alignment of CSD Unit 1 to 2-AP-10 focused only on LT 5, which is about recognizing relevant information from a problem description needed to create an algorithm. We did not find any alignment of Unit 1 with any other 2-AP-10 LT such as creating flowcharts or comparing or testing algorithms.

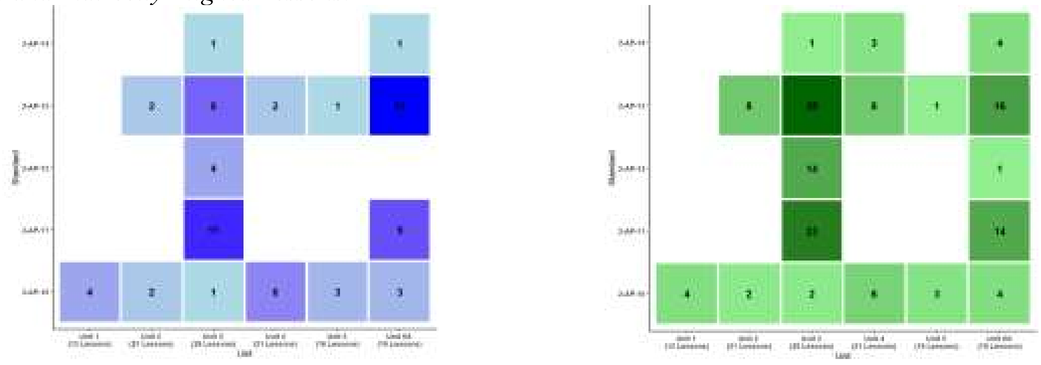
For Unit 2, the web design unit, we found alignment to both 2-AP-10 (algorithms) and 2-AP-13 (decomposition). CSD does not claim Unit 2 to be aligned to 2-AP-10, but we found instruction relating to the ability to test and debug algorithms (LT 10). We recognize that 2-AP-10, LT 10 overlaps with 2-AP-17, a standard that focuses on testing and refining programs (not one of our five target standards). This is probably why CSD indicates that the lessons we aligned to 2-AP10 are aligned with 2AP-17. For 2-AP-13, we found direct alignment with LT 2, the ability to identify intermediate goals and break up a problem. We also found indirect alignment with LT 1 (describing the benefits of decomposition), LT 3 (identify connections and dependencies) and LT 5 (describing how to solve a problem using other solutions).

We found that the interactive animation and games unit, Unit 3, had lessons aligned with all 5 target standards with a focus on 2-AP-11 and 2-AP-12. For 2-AP-10, CSD claimed lessons 2 and 28 to be aligned. Our analysis found lessons 1 and 2 to be directly aligned to LT1, knowledge of algorithms, by engaging students with activities where they had to think about precision in their instructions. Other lessons such as lesson 28 were indirectly aligned to 2-AP-10, as students had to plan out their programs, which may or may not involve flowcharts and pseudocode. For 2-AP-11, CSD claimed 22 of 28 lessons were aligned. We found 11 lessons to be directly aligned and that most of the 2-AP-11 LTs were covered in this unit. The only LTs not covered were LT 5, focusing on the difference between local and global variables, and LT 7 focusing on compound variables such as lists. 12 lessons in which students used variables but were not explicitly instructed on how to use the variables were considered to be indirectly aligned. For 2-AP-12, CSD considered 17 of 28 lessons to be aligned. We found only 4 lessons to be directly aligned with 2-AP-12, 10 lessons to be indirectly aligned, and that not all types of control structures were included. Specifically, lessons covered compound conditionals and repeated conditionals, but we did not find alignment with nested loops or nested conditionals which are called out in the standard. Even for repeated conditions (LT7, LT8), the loop that was often included was a system-defined procedure called "draw" that repeats infinitely; hence students did not engage with more typical forms of iteration such as a *for* loop. For 2-AP-13, we agreed with CSD that all lessons aligned with this standard in some way, but we found only 8 lessons with direct alignment. There was direct alignment with LT2 (breaking up a problem into meaningful pieces) and one lesson had direct alignment with LT 5 (reusing existing pieces of solutions). The other 2-AP-13 LTs were not covered in this unit. For 2-AP-14, Unit 3 contained only one lesson on procedures that aligned with only LTs 1, 4 and 5 focused on defining procedures and creating and calling procedures without input parameters.

Similar to CSD, we found Units 4 and 5 to be mainly aligned to 2-AP-10 and 2-AP-13, the non-programming standards. Unit 4 focuses more on standards within the impacts of computing and computer systems strands. At the end of the unit, students create their own programs and are encouraged to create algorithms and decompose problems. In addition, students may engage with more of the algorithms and programming concepts, which is why we consider there to be indirect alignment to 2-AP-14. Unit 5 focuses on data and society, which aligns better with standards from the Data and Analysis strand. However, in 3 lessons (lessons 8, 13, and 16) students did engage with planning algorithms and decomposing problems.

In Unit 6, we found that students engaged with programming concepts as they focus on app development. CSD indicates alignment with 2-AP-12 for two lessons, but our review of the lessons did not reveal any direct alignment. For example, in lesson 1, students are learning how to set properties and design their app screen. We did not see students engaging with nested or combined control structures and therefore did not align this lesson with 2-AP-12. In this unit, students are asked to plan algorithms, decompose problems, and create and use variables when they are coding. When engaging with variables, we found the focus to be on the first 4 LTs (identifying variables, and naming and creating variables), with some alignment to LT 6, 7 and 8 (assigning values to simple and compound data types, describing how the value of a variable changes in a program).

Figure 1. *Number of CSD Lessons Aligned with Target Standards: (a) Only Direct Aligned Lessons; (b) Directly and Indirectly Aligned Lessons.*


(a) *(b)*

Next, we look across the six units to determine how much of each of the five target standards is covered.

<u>Alignment with 2-AP-10.</u> This is the only standard we found to be aligned across all CSD units. Across all units, we did not find any alignment with LT 3 (knowledge of flowcharts) and LT 7 (ability to develop multiple ways to solve a problem). We found varying degrees of coverage of the other aspects of 2-AP-10, though some of the alignment depends on the representations a class chooses to use for planning programs. LT8 (comparing algorithms) was an aspect of this standard that was covered very infrequently, only in 2 lessons across all units.

<u>Alignment with 2-AP-11.</u> We found alignment to all aspects of the standard at some point across the CSD units. The only LT with no direct alignment was LT 5 (knowledge of global and local variables) though we did find some introduction to scope of variables in Unit 6. In general, we found several instances of indirect alignment with several aspects of this standard. Also, we found that many aspects of the 'variables' standard are not covered till Unit 6 which many classes typically do not get to during the school year.

<u>Alignment with 2-AP-12.</u> We found CSD to be very weakly aligned with combined control structures,

an important CS construct. As mentioned above, we did not see any alignment with the LTs related to nested loops (LT 1 and 2) or nested conditionals (LT 3 and 4). We found alignment with LTs 5 and 6 (compound conditionals using logical operators) in only one lesson across all units. The most common LTs we aligned with were LTs 7 and 8 (repeated conditionals). Even there, the alignment was often not direct - students engaged with conditionals within the "draw" procedure which, in CSD, works similar to an infinite loop. While students have few opportunities to learn about combined control structures in CSD, we recognize that students may choose to engage with these concepts when they are designing and programming their own apps and games.

Alignment with 2-AP-13. For 2-AP-13 we found direct alignment with all our LTs. LT 2 (breaking up a problem into meaningful subproblems) was the most frequently covered LT as students were often working in pairs or groups where they needed to break up the assignment into multiple parts and distribute across the members of the group. LT5 (re-using pieces of existing solutions) was found to be least frequently aligned.

Alignment with 2-AP-14. Students are introduced to procedures or functions in CSD but are not required to use them for most of the assignments. Procedures is the standard with the lowest alignment. Some aspects of the standard are not covered in any CSD unit, for example LT1 (identifying when procedures are useful), LT3 (naming procedures), and LT7 (using the return value from a procedure). Procedures with input parameters is introduced in Unit 6, and procedures with return values are not covered at all. Even procedures with input parameters are challenging concepts for middle school students, but CSD only offers students 1-2 opportunities to engage with the concept, that too in Unit 6 which many students will not get to.

Discussion and conclusion

This paper presents an unpacking of five broad CS standards into granular LTs using an ECD approach and uses the LTs to examine the alignment of a popular middle school CS curriculum with the five standards. Currently, most CS curricula indicate their alignment to the CSTA standards. We point out that the usefulness of this approach is limited if curricula do not specify how they align with standards and which aspects of standards lessons align with. Using the popular CSD curriculum, we demonstrate what detailed standards alignment might look like and what it can reveal about content coverage of a curriculum. Examining alignment with only 5 CS standards, we find that while CSD covers all the standards to varying degrees, it does not provide complete coverage of all components of the standards. For example, CSD does not provide students enough opportunities to learn about procedures and different combined control structures. This is not an indication of the quality of the curriculum, but more a recognition that the CS standards are complex and broad and providing full coverage is not easy. Additionally, students come from a wide range of backgrounds and curriculum must cover some precursors to the grade-level standards, increasing the number of concepts that should be covered. A detailed and more transparent account of standards alignment can help school leaders make informed choices when selecting curricula and indicate to teachers that they may need to supplement student learning on certain concepts.

While we present one way to decompose a small set of standards, we want to emphasize the need for more detailed decomposition of all CS standards so that they are not open to interpretation and can be a more useful guide for instruction and curriculum development. Also, our unpacking did not consider dependencies among LTs as the goal was to develop statements that represent what students should know and be able to do if they have a full mastery of the standards. This opens up the possibility of additional research on learning trajectories for middle school CS standards and exploring how the LTs may build on each other. In addition, we recognize that our results are limited here by the focus on one curriculum and 5 CS standards. A goal of this work was to present a methodology that could be used for curriculum mapping. Our paper demonstrates the value of expanding this work to unpack more CS standards and examine standards alignment for additional curricula.

The challenges with providing coverage of all aspects of a standard also raises the question of whether the middle school CS standards are too ambitious and if it is time for K-12 CS to revisit the standards and consider what is actually feasible in middle school. Providing guidance on what constitutes the non-negotiable critical aspects of CS in middle school can support more consistency among curriculum providers, ensuring all middle school students receive instruction on the same aspects, in turn supporting equitable student access to CS education. Alongside quality curriculum, another cornerstone to equitable CS education is qualified CS teachers. It is difficult for teachers to bridge broad goals such as broadening participation in computing with specific standards-aligned activities to accomplish the goals. Our work with unpacking standards into granular LTs can also be a valuable resource for middle school CS teachers and can support teacher professional learning at a conceptual level that goes beyond merely learning how to facilitate specific CS curricula in classrooms.

References

Banilower, E. R., Smith, P. S., Malzahn, K. A., Plumley, C. L., Gordon, E. M., & Hayes, M. L. (2018). Report of the 2018 NSSME+. *Horizon Research, Inc.*

Basu, S., Rutstein, D., Tate, C., Rachmatullah, A., & Yang, H. (2022). Standards-Aligned Instructional Supports to Promote Computer Science Teachers' Pedagogical Content Knowledge. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1* (pp. 404-410).

Basu, S., Rutstein, D. W., Xu, Y., Wang, H., & Shear, L. (2021). A principled approach to designing computational thinking concepts and practices assessments for upper elementary grades. *Computer Science Education*, *31*(2), 169-198.

Baxter, G. P., & Glaser, R. (1997). *An approach to analyzing the cognitive complexity of science performance assessments*. Center for the Study of Evaluation, National Center for Research on Evaluation, Standards, and Student Testing, Graduate School of Education & Information Studies, UCLA.

Code.org. (2021). 2021 Annual Report. https://code.org/files/code.org-annual-report-2021.pdf

Computer Science Teachers Association (CSTA). (2017). *CSTA K-12 Computer Science Standards: Revised 2017*. http://www.csteachers.org/standards

Computer Science Teachers Association (CSTA). (n.d.). *Standards Alignment Review.* https://csteachers.org/k12standards/standards-alignment-review/

CS for ALL. (2023). Curriculum directory. https://www.csforall.org/projects_and_programs/curriculum_directory/

CSTA Code.org and ECEP Alliance. (2022). *2022 State of Computer Science Education: Understanding Our National Imperative*. https://advocacy.code.org/stateofcs

Fancsali, C., Tigani, L., Toro Isaza, P., & Cole, R. (2018, February). A landscape study of computer science education in nyc: Early findings and implications for policy and practice. In *Proceedings of the 49th acm technical symposium on computer science education* (pp. 44-49).

Fletcher, C. L., Dunton, S. T., Torbey, R., Goodhue, J., Biggers, M., Childs, J., ... & Richardson, D. (2021, March). Leveraging Collective Impact to Promote Systemic Change in CS Education. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (pp. 994-999).

Fowler, B., & Vegas, E. (2021). How England Implemented Its Computer Science Education Program. *Center for Universal Education at The Brookings Institution*.

Gal-Ezer, J., & Stephenson, C. (2014). A tale of two countries: Successes and challenges in K-12 CS education in Israel and the United States. *ACM Transactions on Computing Education (TOCE)*, *14*(2), 1-18.

Grover, S., & Basu, S. (2017, March). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education* (pp. 267-272).

K-12 Computer Science Framework Steering Committee. (2016). *K-12 computer science framework*. ACM.

Martin, A., McAlear, F., & Scott, A. (2015). Path not found: Disparities in access to computer science courses in California high schools. *Online submission*.

Mislevy, R. J., & Haertel, G. D. (2006). Implications of evidence-centered design for educational testing. *Educational measurement: issues and practice*, *25*(4), 6-20.

Rich, K. M., Strickland, C., Binkowski, T. A., Moran, C., & Franklin, D. (2017, August). K-8 learning trajectories derived from research literature: Sequence, repetition, conditionals. In *Proceedings of the 2017 ACM conference on international computing education research* (pp. 182-190).

Sirkiä, T., & Sorva, J. (2012). Exploring programming misconceptions: an analysis of student mistakes in visual program simulation exercises. In *Proceedings of the 12th Koli calling international conference on computing education research* (pp. 19-28).

EdReports (2022). State of the Instructional Materials Market 2021: The Availability and Use of Aligned Materials. https://cdn.edreports.org/media/2022/05/EdReports-State-of-the-Instructional-Materials-Market-6.2022.pdf?_gl=1*1p99xn8*_gcl_au*NzQyNTA2OTI3LjE2OTcyMTE1NjA.

The Center on Standards Assessment Implementation. (2018). *Standard alignment to curriculum and assessment.* https://files.eric.ed.gov/fulltext/ED588503.pdf

Vegas, E., & Fowler, B. (2020). What do we know about the expansion of K-12 computer science education?.

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when?. *Education and Information Technologies*, *22*, 445-468.

Acknowledgments
This material is based upon work supported by the National Science Foundation under Grant No. DRL-2010591. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.