Hammill Institute on Disabilities

Intervention in School and Clinic 2022, Vol. 57(4) 262–267
© Hammill Institute on Disabilities 2021
Article reuse guidelines: sagepub.com/journals-permissions
DOI: 10.1177/10534512211024939
isc.sagepub.com
SAGE

Planning Computer Science Instruction for Students With High-Incidence Disabilities

Amy Hutchison, PhD[1], and Anya S. Evmenova, PhD[1]

Abstract
States increasingly are adopting computer science standards to help students develop coding and computational thinking skills. In an effort to support teachers in introducing computer science content to their students with high-incidence disabilities, a new model, computer science integration planning plus universal design for learning (CSIP+), offers ways to integrate computational thinking and coding into content area instruction. This column presents an example of how a teacher might implement the CSIP+ model when designing instruction accessible to all learners. Guiding questions to support teachers at each phase of the planning cycle are provided.

Keywords
computer science, lesson planning, universal design for learning, CSIP+

Aspects of computer science and sometimes computer science standards increasingly are becoming part of mainstream instruction in elementary schools. To date 33 states have computer science standards (see the list of all states at Code.org, 2019). However, these standards are relatively new and not all states have computer science standards. Little is known about how to best instruct students in understanding computer science standards and developing related skills such as programming. This is especially true for students with high-incidence disabilities.

Many of the computer science initiatives to date have been focused on bringing computer science awareness and opportunities to groups that have historically been underrepresented. These attempts are focused on closing the gender and diversity gap that exists in computing-related fields. For example, organizations such as Girls Who Code (https://girlswhocode.com), Brown Girls Code (https://www.browngirlscode.org), and Black Girls Code (http://www.blackgirlscode.com) emphasize the importance of giving females, especially females of color, the opportunity to learn computer science. However, there has been much less effort aimed at helping students with high-incidence disabilities, who make up about 73% of all students with disabilities (National Center for Education Statistics, 2017). High-incidence disabilities include mild intellectual disabilities (ID), learning disabilities, and emotional and behavioral disorders (EBD) (Bryant et al., 2017). Likewise, little is known about how to support students with high-incidence disabilities in learning computer science skills. In this column, an approach for planning instruction to support students with disabilities in the area of computer science is presented.

Computer Science in the Elementary Grades

In the elementary grades, much of the emphasis of computer science instruction is aimed at developing computational thinking and beginning coding skills. Computational thinking skills are essential for everything from learning in the content areas to being successful in workplace settings. Computational thinking is used by professionals including coaches, chefs, soldiers, delivery drivers, teachers, and software engineers. Although definitions of computational thinking vary, computational thinking is generally defined

[1]George Mason University, Fairfax, VA, USA

Corresponding Author:
Amy Hutchison, George Mason University, 4400 University Dr., MS 4B3, Fairfax, VA 22030, USA.
Email: ahutchi9@gmu.edu

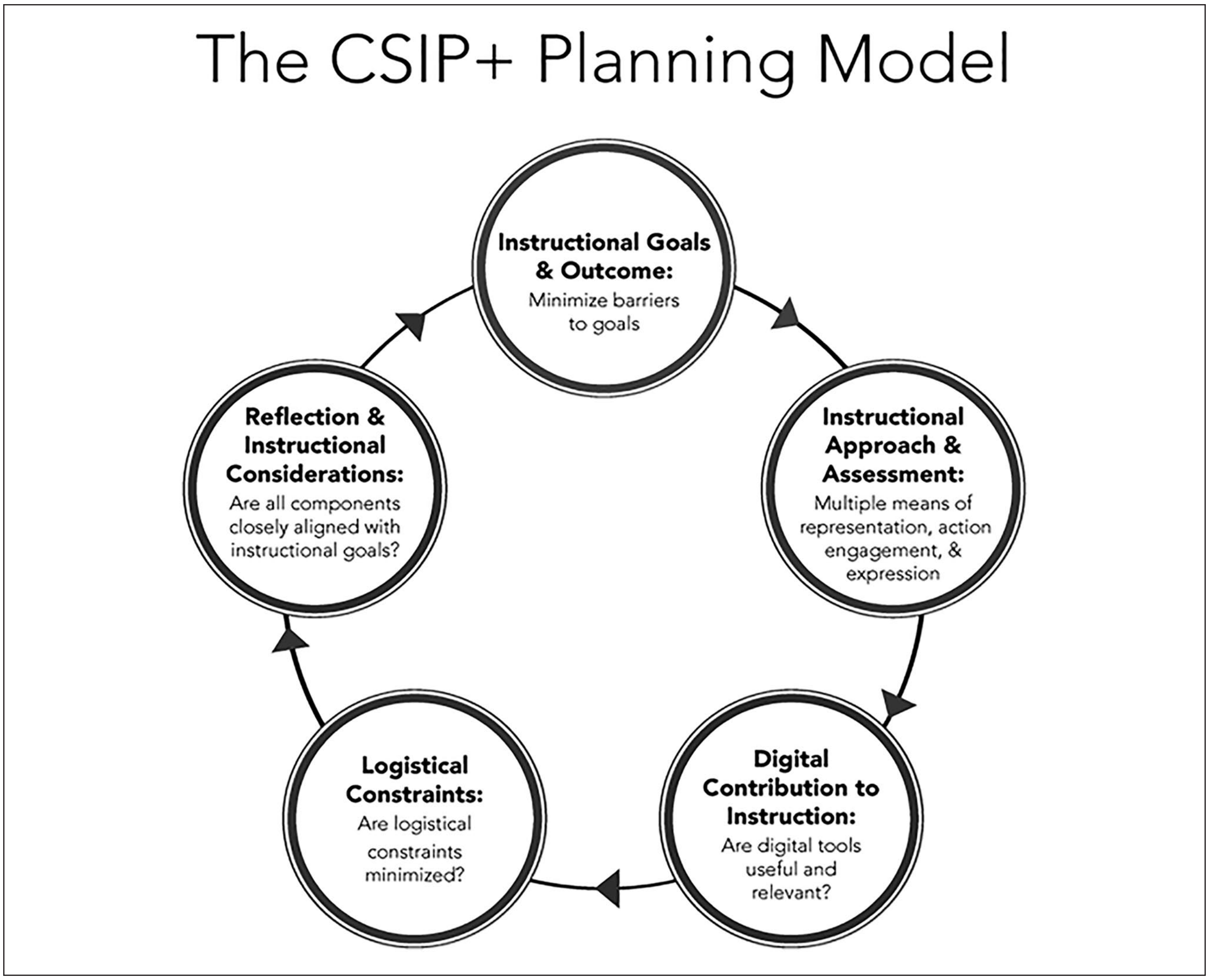
The CSIP+ Planning Model
Instructional Goals & Outcome: Minimize barriers to goals
Instructional Approach & Assessment: Multiple means of representation, action engagement, & expression
Digital Contribution to Instruction: Are digital tools useful and relevant?
Logistical Constraints: Are logistical constraints minimized?
Reflection & Instructional Considerations: Are all components closely aligned with instructional goals?

Figure 1. The computer science integration planning plus planning model.

as "the conceptual foundation required to solve problems effectively and efficiently (i.e., algorithmically, with or without the assistance of computers) with solutions that are reusable in different contexts" (Shute et al., 2017, p. 151). The primary components generally included as part of computational thinking are sequencing, abstraction, decomposition, and algorithmic thinking (Shute et al., 2017). Although the term *computational thinking* originated in the field of computer science (Wing, 2006), it is now widely believed that computational thinking skills are broadly applicable beyond computer science (Berland & Wilensky, 2015). Being able to engage in computational thinking is believed to be beneficial for solving problems of all kinds. Furthermore, although computational thinking skills are not the same as coding skills, these skills are necessary and beneficial for learning to code (Israel et al., 2015).

Similar to computational thinking, coding literacy is increasingly considered to be an essential skill (Hutchison et al., 2016). Vee (2017) argued that coding has moved beyond a necessary skill for high demand fields to include elements of expression, collaboration, and creativity. Further, coding helps students organize their thinking and express their ideas (scratchjr.org, 2015). The primary approach for developing coding skills is through the use of digital tools that employ a visual programming language or block-based coding such as Scratch (www.scratch.mit.edu) and Scratch Junior (see https://scratch.mit.edu/educators for more information). With block-based programming, users drag and drop command blocks in a sequential fashion to create animations and other creative products.

For all of the aforementioned reasons, it is essential that the needs of students with high-incidence disabilities are considered when designing instruction on computational thinking and coding. Aspects of computational thinking and coding that have previously been identified as difficult for students with high-incidence disabilities include (a) developing and executing multistep procedures, (b) conceptualizing and following conditional commands, and (c) understanding and applying new vocabulary, such as algorithms (Israel et al., 2015). There are also many other aspects of computational thinking and coding that are likely to be difficult for students with high-incidence disabilities, such as (a) moving beyond copying and modifying a code that is given to them to planning their own creative project, (b) determining what code is needed to create what they have planned, and (c) finding and fixing problems in the code as they arise.

A Model for Students With Disabilities

In a project funded by the National Science Foundation, a new approach to support elementary students with high-incidence disabilities in learning computer science was proposed. As part of that project, the CSIP+ model was developed to provide teachers with an approach for integrating computational thinking and coding into content area instruction. In this case, content area instruction is defined as one of the knowledge domains such as literacy, math, science, and social studies. By integrating computational thinking and coding into content area instruction, students with disabilities have the opportunity to learn these skills just as other underrepresented groups of students do. The acronym CSIP stands for computer science integration planning cycle. This approach was adapted from an existing instructional planning tool called the technology integration planning cycle (Hutchison & Woodward, 2014). In turn, CSIP+ includes the integration of the universal design for learning (UDL) cycle of instructional planning (Rao & Meo, 2016) and the UDL guidelines and checkpoints (CAST, 2018) into the CSIP approach. The UDL cycle of instructional planning offered ways to proactively incorporate three the UDL principles (i.e., multiple means of engagement, multiple means of representation, multiple means of action/expression), nine guidelines, and 31 checkpoints into UDL-based lessons (see http://udlguidelines.cast.org for more information). When the UDL cycle of instructional planning was integrated with the CSIP, the resulting CSIP+ planning model helped teachers focus on using UDL principles to integrate computer science into content area instruction. Figure 1 shows the CSIP+ planning model.

Explanation of the Model

As part of the CSIP+ model, a list of questions was developed to guide teachers in implementing each phase of the planning cycle. The CSIP+ checklist supported an instructional design into which teachers integrated computer science and also followed the UDL guidelines to ensure that the instruction was appropriate for all students. A list of questions follows that accompany each step of CSIP+ model.

Step 1: Instructional Goals and Outcomes

In Step 1, teachers start by selecting an instructional goal, learning objectives, and outcomes for the lesson. A teacher should consider the following questions:

- What content area instructional standards am I planning to teach?
- Which computer science standards or objectives could be integrated with the content area standard(s) I plan to teach?
- What are the goals or objectives and intended outcomes of the lesson when the content area and computer science standards are combined? What are students expected to learn in my lesson?
- What learner characteristics and barriers in the classroom might interfere with students reaching these goals?

Step 2: Instructional Approach and Assessment

Considerations guiding Step 2 of the CSIP+ model include the following:

- To what extent should my lesson or unit include direct instruction, modeling, guided practice, independent practice, and collaborative practice or work?
- Will the lesson or unit be longer or shorter in duration?
- How can I use multimedia materials and digital tools to represent content in multiple forms, highlight critical features, activate background knowledge, and support vocabulary? (i.e., multiple means of representation principle)
- Has some aspect of this content been introduced previously? If not, how will I scaffold the instruction to ensure that students understand the content for both the computer science and content area standards?
- How can I give students digital and non-digital options for expressing what they know? (i.e., multiple means of action/expression principle)
- How will I provide models, feedback, and supports for different levels of student proficiency?
- How can I provide autonomy and choice to promote self-regulation? (i.e., multiple means of engagement)
- How will I incorporate the UDL guidelines and checkpoints to formatively or summatively assess

students' progress toward the computer science and content area standards and objectives?

Step 3: Digital Contribution to Instruction

Considerations guiding Step 3 of the CSIP+ model include the following:

- Does the inclusion of computer science content enhance or support my content area goals?
- Is my use of digital technology useful and relevant to my instructional goals and outcomes?

Step 4: Digital Contribution to Instruction

Considerations guiding Step 4 of the CSIP+ model include the following:

- If using a digital tool, are there any potential logistical constraints that will interfere with the instructional goal or require excessive time or effort? (e.g., Does the tool require individual accounts? Can students save their work for later? Can students share work completed with the tool? Is the tool navigation intuitive?)
- How can I reduce logistical concerns to maximize instructional time and alignment with the instructional goals?

Step 5: Reflection and Instructional Considerations

Considerations guiding Step 5 of the CSIP+ model include the following:

- Does the planned use of digital tools closely align with the instructional goals and outcomes? Has introduction of new strategies, tools, and options created drift from my original goals?
- Do I have a clear plan for collecting and evaluating student work? (How will students submit digital products? How will digital products be evaluated?)
- Will any aspects of the physical environment need to change, as a result of the instructional activities, to create space, reduce sound, or minimize distraction?

Lesson Design Using the Model

In this section, an example is provided of how instruction could be designed with the CSIP+ model. In *Step 1*, instructional goals and learning outcomes are determined by selecting a content area standard and a computer science standard that can be paired. For example, computer science can be integrated into literacy instruction by pairing these two standards:

- CSTA1A-AP-12: Develop plans that describe a program's sequence of events, goals, and expected outcomes.
- CCSS.ELA-LITERACY.W.2.3: Write narratives in which they recount a well-elaborated event or short sequence of events; include details to describe actions, thoughts, and feelings; use temporal words to signal event order; and provide a sense of closure.

Based on these standards, an example of an instructional goal is as follows: Students will write a narrative essay describing a problem they have had and how they broke the problem into smaller steps to solve it. Students will use a graphic organizer to decompose their story into programmable scenes. Students will use Scratch or Scratch Jr. to create an animation illustrating each step of the problem they wrote about and how it was solved. As part of Step 1, learner characteristics and barriers in the classroom that might interfere with students reaching these goals should also be considered. For example, some students may have difficulty handwriting a story and may need alternative ways to produce it (e.g., typing, using word prediction, using speech-to-text, drawing). Some students may be overwhelmed with the task of describing something step-by-step and may need multiple options for planning their story with various scaffolds and supports. Some students may struggle to use Scratch or Scratch J. and may benefit from using templates or step-by-step directions. These supports will be planned in the next step.

In *Step 2*, the specific instructional approach and assessment are considered. A brief example of how to approach this lesson could be broken into the following steps:

1. Direct instruction: The teacher presents the word "decompose" and asks students to express everything they know about the meaning of the word. The teacher leads a discussion of conceptions and misconceptions about the term and explains that, in computing, decomposition is the process of breaking down a task into smaller, more manageable parts. It has many advantages. It helps us manage large projects and makes the process of solving a complex problem less daunting and much easier to take on. The teacher helps the students see that this process is also useful for writing tasks.
2. Modeling: The teacher models his or her thinking about a task that can be decomposed into smaller tasks, such as making breakfast, and how to use a graphic organizer to break apart each part of the task.

3. Guided practice: The teacher guides students in thinking of a problem or task that can be broken into steps and selects one of the ideas to guide students in breaking apart (i.e., decomposing) the problem or task. With guidance, the students add the example to a graphic organizer of their own.
4. Independent practice: Students will think of a time they have had to solve a problem or task and the steps they had to take to solve it. Students will use a graphic organizer to explain the problem and break apart each of the smaller steps they had to take to solve it. Students will then consider how they could turn each of the steps into a story scene that could be programmed into an animation illustrating each step of the problem and how it was solved. Students will use *Scratch Jr.* to create their animation illustrating a problem and how it was solved.

The teacher would then consider all the remaining questions listed for Step 2 and make further changes that may include the following:

- Providing multiple means of engagement (throughout the lesson)
    - Offering choice in whether students work in a group or individually, choice in the problem students want to describe, as well as choice in the characters and backgrounds students choose in Scratch Jr.
    - Proving templates or step-by-step directions for planning the story in Scratch Jr.
- Providing multiple means of representation
    - Adding illustrations and examples to the oral discussion or showing a video model (during direct instruction and guided practice)
    - Activating background knowledge on narratives, using graphic organizers, and using Scratch Jr. (during direct instruction and guided practice)
    - Offering various scaffolds and supports for independent practice such as sentence starters in the graphic organizer
- Providing multiple means of action and expression (during independent practice)
    - Allowing students to write, type, use speech-to-text, or draw their story

In *Step 3*, consideration is given to whether the inclusion of computer science content enhances or supports the content area goals and if the use of digital technology is useful and relevant to my instructional goals and outcomes. In this case the computer science content is well-aligned with the content area goal. However, the teacher may consider that it may not be necessary for the students to use the digital technology, Scratch Jr., as planned since it useful, but not necessary, for meeting the computer science standard.

In *Step 4*, consideration is given to the potential logistical constraints that may interfere with the instructional goal or require excessive time or effort. Then, efforts should be made to reduce logistical concerns to maximize instructional time and alignment with the instructional goals. In this example, since the use of Scratch Jr. is not necessary for meeting the computer science of literacy standard, the complexity of the lesson could be reduced by making Scratch Jr. an optional way for students to express their understanding instead of a requirement. Making Scratch Jr. optional, and providing other ways for students to express their understanding, makes it less likely that the logistical challenges will interfere with the overall goal of the lesson.

In *Step 5*, reflection on the lesson ensures that all aspects of the sample are aligned with the original content standards and instructional goal for the lesson. The introduction of new strategies, tools, and options has not created drift from the original goals. However, reflection on the assessment would reveal that the teacher still needs to develop a clear plan for evaluating student work, which should include multiple means of action and expression for students. Furthermore, the physical environment still needs to be considered to determine how to minimize distraction if some students are working on digital devices or working collaboratively.

Conclusion

Although little is known about how to best support elementary students with high-incidence disabilities in learning computer science, the CSIP+ instructional planning model provides a useful starting point for helping teachers consider the intersections of content area instruction and computer science, as well as how to design this instruction so that is accessible and effective for all students. The CSIP+ planning model may also promote awareness of the need to provide all students with the opportunity to learn computer science. The design of such instruction is critical to ensure that it is accessible and effective for all students.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

This work was partially supported by funding through the National Science Foundation, Award #1837380.

References

Berland, M., & Wilensky, U. (2015). Comparing virtual and physical robotics environments for supporting complex systems and computational thinking. *Journal of Science Education and Technology*, *24*, 628–647. https://doi.org/10.1007/s10956-015-9552-x

Bryant, D. P., Bryant, B. R., & Smith, D. D. (2017). *Teaching students with special needs in inclusive classrooms*. SAGE.

CAST. (2018). *Universal design for learning guidelines version 2.2*. http://udlguidelines.cast.org

Code.org. (2019, July 11). *33 states expand access to K-12 computer science education in 2019*. https://medium.com/@codeorg/32-states-expand-access-to-k-12-computer-scienceeducation-in-2019-7d2357fe6f3d

Hutchison, A., Nadolny, L., & Estapa, A. (2016). Using coding apps to support literacy instruction and develop coding literacy. *Reading Teacher*, *69*(5), 493–503. https://doi.org/10.1002/trtr.1440

Hutchison, A., & Woodward, L. (2014). A planning cycle for integrating digital technology into literacy instruction. *The Reading Teacher*, *67*(6), 455–466.

Israel, M., Wherfel, Q. M., Pearson, J., Shehab, S., & Tapia, T. (2015). Empowering K-12 students with disabilities to learn computational thinking and computer programming. *Teaching Exceptional Children*, *48*(1), 45–52.

National Center for Education Statistics. (2017). *The condition of education*. https://nces.ed.gov/programs/coe/indicator_cgg.asp

Rao, K., & Meo, G. (2016). Using universal design for learning to design standards-based lessons. *SAGE Open*, 6(4), 1–12. https://doi.org/10.1177/2158244016680688

Scratchjr.org. (2015). *What is ScratchJr?* http://www.scratchjr.org/about.html

Shute, V., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158. https://doi.org/10.1016/j.edurev.2017.09.003

Vee, A. (2017). *Coding literacy*. The MIT Press.

Wing, J. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–36.