CHAPTER

1

Why Should K–5 Educators Teach Coding?

It's time we realize and put to action Steve Job's words: "Everyone in this country should learn how to program a computer...because it teaches you how to think." This quote appears at the beginning of a video entitled "What Most School Don't Teach" (**youtu.be/nKIu9yen5nc**) that was published in 2013 by Code.org. Starring many famous people, including Bill Gates of Microsoft, Mark Zuckerberg of Facebook, and will.i.am of the Black-Eyed Peas, the video's message is a simple one: anyone and everyone should learn to code.

Within our schools, we must build upon the belief that coding is for everyone. There are many reasons for teaching coding, among them:

- It's about teaching perseverance.
- It's about teaching students how to think and reason (computational thinking).
- It's about creativity and expression.
- It's another way to demonstrate content knowledge (just like creating a PowerPoint or display board).
- It's a way to see math in action.

Coding Is for Everyone, and Parents Agree!

Code.org is a non-profit organization dedicated to giving every student in every school the opportunity to learn computer programming. An article published on the Code.org blog *Anybody Can Learn* reports that of all the new wages within the U.S., only 16% are in computer science (Code.org, 2016).

The Bureau of Labor Statistics data on mean salaries showed the average salary across all occupations is $48,320, while the average salary across computing jobs is $86,170. They also acknowledge that the computer science field is growing faster than other fields, and so they expect the number to grow. The average "demand rate" (online ads divided by current employment) is 14.8% for computing categories, and 3.8% on average across all other jobs (Code.org, 2016).

The article also shared the scary statistic that in 2014, only 42,969, or two and a half percent of all bachelor's degrees were earned in computer science. Why aren't more university students studying computer science? One reason is because students don't learn about this field in grades K–12, despite evidence that early exposure is highly correlated to majoring in computer science.

The solution proposed in the article is for schools to teach computer science in grades K–12 and, according to a survey Code.org conducted, 90% of parents agree.

Five Reasons Why Coding Is Critical for K–5 Students

1. Making Their Thinking Visible

We know that young students are concrete thinkers and are beginning to follow step-by-step directions. These beginning stages of following first one step, then two steps, then multiple steps, are the start of algorithmic thinking in action. While the youngest learners may not understand this abstract concept, we can use computer science to make their thinking visible.

One of the behaviors of good readers is to visualize the story in their mind as they are reading. Students often struggle to work in reverse and put their thoughts into writing, as they cannot see them. By learning how to code, students have an opportunity to give shapes, thoughts, and actions to their thinking.

2. Sustaining Creativity

In Sir Ken Robinson's TED Talk, "Do Schools Kill Creativity?" the renowned educator and speaker tells the audience that creativity is as important in education as literacy. He points out that young learners will take a chance and are not afraid to

be wrong. As we get older, he says, adults lose their capacity for creativity because they are afraid of being wrong (Robinson, 2006).

Coding allows students to be creative without being wrong. If something doesn't work, the student must analyze what isn't working, ask why it isn't working, and determine how to correct it so that it works. In essence, coding is the process of continually making mistakes, learning from them, and correcting them.

3. Encouraging Computational Thinking

As a teacher, how many times have you heard the following feedback? *"Johnny is great at solving computational math problems, but he continues to struggle with word problems."*

Teaching how to read and write code supports a student's ability to think computationally. To make breakthroughs in teaching students how to solve word problems, we must help them understand how their brains work—like a highly complex computer. This process involves breaking apart a problem (Decomposition), identifying and creating the steps needed to solve the problem (Algorithms & Procedures), running the procedures (Data Collection), analyzing the results (Data Analysis) and determining if the results yielded an acceptable answer (Data Representation & Abstraction).

Our world continually presents us with roadblocks that, given the correct framework of thinking, have solutions that can harness the power of technology to make a difference in the world. Jeanette Wing defines computational thinking as "the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be efficiently carried out by an information-processing agent" (Wing, 2006). For the classroom teacher, this means they must look at the students' brain as the information-processing agent.

4. Fostering Future-Ready Skills

The Partnership for 21st Century Learning (P21) developed a framework describing the "skills, knowledge and expertise students should master to succeed in work and life in the 21st century (Partnerhip for 21st Century Learning, 2007)." The framework identified three learning and innovation skills essential in preparing our children for increasingly complex life and work environments that don't yet exist. These skills, often referred to as the 4 C's, are critical thinking, communication, collaboration, and creativity.

Collaboration and communication are rapidly changing with the use of technology. We can now collaborate by communicating across the state, country, and world in real time. Working on a project and receiving instant feedback, due to advances in computer programming, has pushed us into needing a workforce that can think computationally.

Creativity and critical thinking can be used with coders of all ages. Coding allows the user to become the creator of content, rather than just the consumer of content. When we consume content we are learning about the "What" and the "How," but when we create content we have engaged the "Why" of learning.

5. Empowering Students to Take Action

Coding is about applying skills and creativity to solve problems. For example, in the winter of 2011, a group of young coders were stuck in Boston during a snow storm. Their task was to create a new website for Boston's Public Schools; however, within days of their arrival the city had shut down. What resulted was the coders creating a website called Adopt a Hydrant (adoptahydrant.org) that allowed area residents to adopt a fire hydrant and keep it clear of snow for emergency personnel during the winter months.

Coding can be used to create real-world contexts for students. When we blur the lines between school and the real world, we allow children to examine problems, engage them in exploring the problems, and empower them to take action in finding a solution.

Many people think action is something you do and is therefore easy to define; however, if we go back to the Golden Circle, we recognize that action stems from the reason why we do something. If we want children to make a difference in the world, we need to help them understand that action is not a mandate from parents or teachers, but a lifelong mindset that teachers and parents can help develop. It must be developed with scaffolded lessons that include explicitly taught skills, modeled behaviors, and a gradual release of responsibility.

The following teacher reflection questions can help you begin to think about the process:

- What is inquiry's relationship to action?
- How does computational thinking support the skills needed to take action?
- How can technology be used to record, assess, and report on action?

- How do coding skills fit into one's ability to take action?

Why *Not* Teach Coding?

While there are many reasons for adding coding, there are an equal or greater number of arguments against. The first step to incorporating coding into existing curriculum is to identify the roadblocks.

Roadblock 1: *Getting Comfortable with Living in Beta*

In many schools across the country there are after school clubs dedicated to computer science and coding. While this is a great start for exposure, it does not allow all students to access the computational thinking skills that are developed by coding. Teachers who have a passion for computer science are usually the ones running these clubs, but how do we get *all* K–5 teachers comfortable with coding?

This brings us to the ability to live in beta. This concept is explored by Molly Schroeder in her 2013 TED talk "Living in Beta" (youtube.com/watch?v=0nnYI3ePrY8). Schroder defines beta as "the space between A and B" and adds that this is where the learning happens. She points out that with the ever-changing nature of technology, we would best serve our students by jumping in with them as they approach solving a problem. New discoveries in neuro-science and revelations on how the brain works are changing the way teachers interact with student learning opportunities. Coding is a new literacy that is upon us and, as she states in her TED Talk, "if we sit around waiting until it all shakes out, we're going to miss the boat. Not only will we be too late and learn nothing along the way, but we'll also lose out on the opportunity to help influence how things evolve and take shape (Schroeder, 2013)."

Roadblock 2: *Fear of Failure*

Often I have heard K–5 teachers say, "but I can't do that without a lot of professional development." Teachers may fear the unknown and feel they don't understand, or have time to learn, how to code. More importantly, the field of education has an overall fear of failing, spreading to the paralyzing fear of risk-taking. This is our biggest hurdle! Allowing teachers to take risks, fail, and learn to improve the process from their failure is what coding is all about.

The essential question becomes, how do we get off this escalator we are stuck on? Take a moment to watch the YouTube video "Stuck on An Escalator–Take Action" (**youtu.be/VrSUe_m19FY**). It shows two people who are literally stuck on an escalator when it stops moving. When I showed this to a group of teachers, they all laughed as the lady asks the gentleman stuck on the escalator with her for a cell

phone. The gentleman then proceeds to shout, "Hello...there are two people stuck on an escalator and we need help...now! Would somebody please do something?" The two proceed to sit down on their respective steps until a repair technician comes to fix the escalator, only to get stuck about half way up the escalator. The video concludes by stating that most problems are easy to solve, you just need to get off the escalator. In order to get off our escalator and face our fear of coding, we need to understand the Law of Diffusion of Innovation and how it relates to coding in K–5.

Everett Rogers, a professor of communication studies, popularized this theory in his book *Diffusion of Innovations* in 1962. The book has since been revised and the fifth edition was published in 2003. The basis of the theory is that diffusion is the process by which an innovation is communicated over time among an organization. According to Rogers' theory, there are five categories of those who adopt an innovation.

Innovators. Individuals who are willing to take risks, have financial backing, and have scientific resources and interaction with other innovators, all of which allows them to adopt innovations that may ultimately fail.

Early Adopters. Those who have the highest degree of opinion leadership and advanced education. They are more cautious in adoption choices than the innovators.

Early Majority. Those who will adopt an innovation after a degree of time in which they have seen its successful use. This group usually has above average social status, but this group is not seen as having opinion leadership.

Late Majority. Those who approach an innovation with skepticism.

Laggards. The last group to adopt the innovation. This group of individuals typically has an aversion to change.

The innovators of the K–5 Coding include Seymore Papert, Michel Resnick, Fred Martin, and others who developed and adopted early programming languages; beginning in the 1970s with Logo, which was used to draw shapes, designs, and patterns by typing in simple commands on the screen; and leading to the development of Scratch, a visual block programming language made specifically for young coders, in 2004.

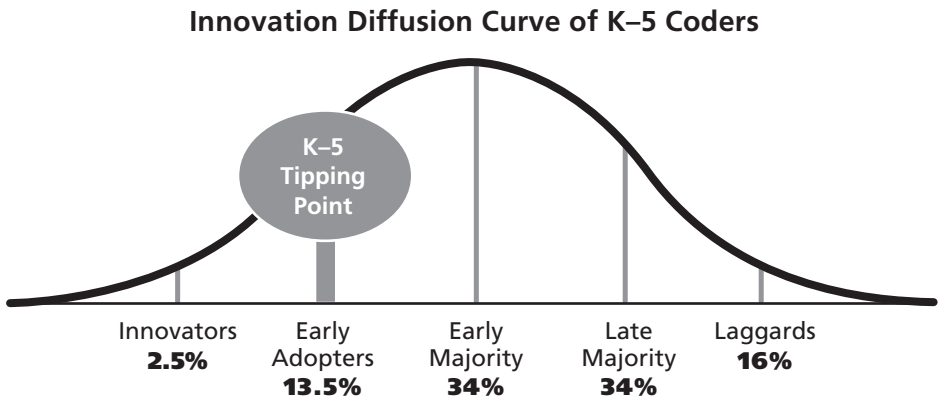

Figure 1.1. Innovation diffusion curve showing rate of adoption of coding by K–5 students.

The early adopters began catching on to these coding platforms and coding grew in popularity throughout the first decade of the 21st Century. In 2004, the Computer Science Teachers Association (CSTA) was founded and it created recommended computer science standards. Coding gained more popularity though collaborative social platforms like the one created for Scratch (**scratch.mit.edu**) in 2012. Hadi Partovi followed suit and launched Code.org in 2013. Each year, Code.org organizes the Hour of Code, which, according to their 2016 Annual Report, has engaged more than 350 million students, reaching one out of every ten children on the planet (Code.org, 2017).

Figure 1.1 shows the tipping point that Code.org hopes to achieve through the coding resources they provide for students and educators.

By picking up this book, and trying out at least one activity, you will be an early adopter by bringing coding into the curriculum and fostering development of this important 21st century literacy in your school!

Roadblock 3: *Time*

For K–5 teachers to incorporate coding and computational thinking, we simply cannot add another subject to their already full schedules. Teachers are already tasked with creating learning experiences in reading, writing, science, social studies, health, and habits of mind. How can they be expected to add learning and teaching coding to their schedules?

What teachers may not realize is that they are already teaching many of the foundational components of computational thinking. When we ask students to read a story and then sequence the events into the correct order, they are using the same process a computer programmer uses in decomposition, data analysis, and data representation. For example, when computer programmers are presented with a complex problem to solve, they first need to break the problem down into simple parts. Then the programmer will analyze the parts and put them in the correct order. Finally, the computer will represent the data in a simple way that the user can understand. Our youngest readers use this same process when the teacher reads them a story and they must take the picture representation of the story and put it into the correct sequence of events to make meaning. It is not more time we need, but the ability to help our students make the connections.

The required skills within the Common Core State Standards and particularly the Standards for Mathematical Practice can be taught using coding as a tool. I often hear of teachers struggling to find concrete ways to teach and observe these eight mathematics standards. Appendix B shares connections between computational thinking and the Standards for Mathematical Practice, and strategies for implementing coding within the math classroom. Again, it is not a matter of adding more time to another subject, but using a 21st century tool in a way that will help the concrete understanding of often time abstract mathematical concepts.