Isidore Newman School

Computer Science at Newman

A comprehensive investigation
and implementation strategy
for K-12 coding

Jenna deBoisblanc
Oscar Creech

December 2015

Foreword

While I strongly support the axiom that scientific research should eschew first person pronouns, I'd like to offer a foreword encapsulating my own experiences with computer science, a narrative that I believe is highly relevant in a document examining the importance of K-12 computer science education.

Coding is my greatest passion and my greatest asset. I've coded at every step of my career, from lab equipment for my physics thesis, databases for software companies, pollution data for environmental non-profits, and the list goes on. While my classmates struggled to find employment upon graduating in 2011 (the end of the recession), I immediately landed a cushy software job complete with a beer fridge and catered lunches. But perhaps most importantly: I code in my free time for *for pleasure* because coding is an incredibly versitile creative platform.

And yet, upon graduating from high school, I had no idea what computer science entailed or that software engineering might be a career I'd later consider persuing. If I had, I would have majored, or at least minored, in CS. In addition, I've written at length about the reasons why, *as a woman*, I almost didn't major in physics in college. I'll suffice it to say, closing the gender gap in hard sciences begins in elementary and secondary education.

Fortunately, I lucked out and graduated with a degree that mandated ample coding experience, and I consider myself highly priviledged to have found a subject that affords such opportunity. I hope to use my experiences not as a bias propelling a blind campaign in search of the latest trend, but rather as the bedrock passion behind the push to ensure that education keeps pace with evolving tools and means of exploration.

Onwards!

Jenna deBoisblanc

Self-Consistency...

This document was coded with the programming language LaTeX.

Contents

Chapter 1

Introduction

1.1 What is computer science?

For the purpose of avoiding potentially derailing semantics, it's instructive to begin with definitions of computer-related concepts used throughout this document.

"Coding" is the latest term to gain momentum in the education vernacular, popularized by the Hour of Code and online learning platforms like Scratch. While "coding" may be used to convey the beginning steps of **computer programming** [1], the terms are used synonymously and can be defined as follows: the process of writing or compiling machine instructions to accomplish a task. Coding is most commonly associated with the creation of websites, software applications, and mobile apps; however, there are innumerable potential applications of code. There are languages to program music, wearable tech, autonomous drones, interactive art visualizations, financial data, research documents (like this one!), 3D printers, robotics, and the list goes on.

Perhaps less common but equally as important is the concept of **"computational thinking"** (CT). Jeanette Wing, the Director of the Carnegie Mellon Center for Computational Thinking, is arguably the projenitor of the term [11]. The center describes CT as "a way of solving problems, designing systems, and understanding human behavior that draws on concepts fundamental to computer science... [by] making use of different levels of abstraction and thinking algorithmically" [2]. Google for Education defines CT as "a problem solving process that includes a number of characteristics, such as logically ordering and analyzing data and creating solutions using a series of ordered steps (or algorithms), and dispositions, such as the ability to confidently deal with complexity and open-ended problems" [3]. To summarize, CT is a set of problem-solving mindsets independent of computer programming that are applicable to all disciplines, not just computing. Emphasis on CT, especially in early K-12 education, will lay a strong foundation for future coursework in a myriad of fields.

Computer science (CS) employs computational thinking mindsets in the study and execution of computer programs and algorithmic processes. Coding is an essential, but not exclusive, component of CS, as is the study of hardware and software designs, their applications, and their impact on society.

While coding, computational thinking, and computer science are used interchangably throughout this document to refer to a set of K-12 curriculum encompassing all three, is important to distinguish "computer science" from **"computer literacy."** The National Center for Women and Information Technology states, "the ability to create and adapt new technologies distinguishes computer science from computer literacy, which focuses more on using existing technologies (e.g., word processing, spreadsheets)" [4]. Computer science teaches students how to think logically, solve hard problems, and understand the internal processes of computing rather than merely understanding how to use specific software. Although this document will focus primarily on CS, computer literacy is neither obsolete nor irrelevant (e.g. keyboarding) in 21st century education.

1.2 The importance of coding in K-12 education

In September 2015, New York City Mayor Bill de Blasio announced that all of the city's public schools would be required to offer computer science within the next 10 years. The announcement was neither the first of its kind nor the boldest; in December 2013, Chicago Mayor Rahm Emanuel launched Computer Science 4 All, a plan mandating a year-long computer science high school graduation requirement by 2019 [5]. And on the national stage, in early December 2015 President Obama signed into law a bipartisan bill - the Every Student Suceeds Act - recognizing computer science as a foundational academic subject on equal footing with subjects like math and English [6].

Municipal, state, and federal policymakers' recent focus on computer science education is well-placed. Politicians on all levels of government are attempting to answer the technology industry's clarion call to fill a growing shortage of engineers. Education, especially K-12 education, is failing to keep pace with a rapidly-evolving workplace, jeopardizing the success and employment potential of American students.

COMPUTER SCIENCE EMPLOYMENT

According to the U.S. Bureau of Labor Statistics, computer and information technology jobs will be some of the fastest growing occupations in America, making up approximately 2/3 of the 1.1 million new STEM jobs projected for 2012-2022. [7]. By 2022 the Bureau of Labor Statistics estimates there will be over 1 million open computing jobs [7]. Code.org, a non-profit dedicated to expanding computer science education, emphasizes that these computing jobs will

fall under every industry, not just tech. Finance, business, government, manufacturing, and education are all industries projected to need computating skills in the next decade [8].

Not only are unfilled computer science jobs abundant, they have high income potential. The highest paying degree in the U.S. is a computer science degree from Carnegie Mellon ($84,000), and among all universities, the National Association of Colleges and Employers (NACE) survey ranks computer science as first or second in income potential [9].

WORKFORCE SHORTAGE

Despite the high growth projections of well-paying computing jobs, skilled programmers aren't entering the workforce at a rate necessary to keep up with demand. Code.org estimates that there are 600,000 computing jobs open in the U.S. but only 38,175 students graduated with a computer science degree in 2013 [8]. 38,175 represents under 8% of all STEM degrees earned in 2013, a problematic number considering that projections estimate 2/3 of all STEM jobs will be in a computing field [7].

K-12 EDUCATION GAP

Education, especially K-12, has a significant role to play in the growing shortage of computer sciene graduates. Only 1 in 4 U.S. schools offer computer science classes with computer programming [10], and in 23 states computer science cannot be used to meet math or science graduation requirements [6]. According to the Gallup study [10], many school administrators do not perceive a demand by parents and students for computer science. The study states that the main reason school principals do not offer computer science is the lack of time for classes not required by testing, as well as the inability to hire computer science teachers due to availability and budget constraints [10].

UNDERREPRESENTED GROUPS

Women and people of color are especially at risk of falling behind and losing access to high-paying computing employment. In 2011, women made up 21% of students who took the AP computer science exam, and less than 1% were African American [11]. For many students, access to technology is the greatest hurdle. Hispanic students have less access to internet access at home than white or black students [10].

In the context of evaluating the importance of K-12 computer science education, it is critically important to point out that this opportunity gap begins in K-12 education. Table 1.1 indicates that by high school, women and people of color are underrepresented in the AP CS exam; this trend remains relatively stable through college and into the workforce.

While closing the gender and achievement gap may seem daunting, early and frequent exposure to computer science throughout K-12 education has been demonstrated to have significant impacts on career outcomes. Women who try AP Computer Science in high school are ten times more likely to major in it in college, and Black and Hispanic students are seven times more likely [12].

Group	**AP CS Exam**	**CS Bachelor's**	**Computing Jobs**
Females	22%	17%	23%
People of Color	13%	18%	14%

TABLE 1.1: Percentage of Underrepresented Groups in Computer Science [8]

CONCLUSION

Computing jobs are on the rise, and unless public policy and education quickly fall in line with the needs of an evolving workforce, American students will miss out on valuable opportunities to pursue careers in a myriad of fields. K-12 exposure to computer science, especially for women and people of color, is necessary to close the computing gap. To that end, placing computer science on equal footing with other core subjects or graduation requirements is a necessary step to ensure our students are able to succeed in the 21st century.

Chapter 2

Research

2.1 Case Studies

2.1.1 Index Schools

2.1.2 Other Independent Schools

Phillips Exeter Academy Exeter, NH.
Ransom Everglades School Coconut Grove, FL.
The College Preparatory School Oakland, CA.
Horace Mann School Bronx, NY.
Castilleja School Palo Alto, CA.
Trinity School New York, NY.
The Hotchkiss School Lakeville, CT.
San Francisco University High School San Francisco, CA.
Dalton School

2.1.3 Public Schools

2.1.4 Code Schools

2.1.5 Higher Education

2.2 National Standards

2.2.1 CSTA

The Computer Science Teachers Association (CSTA) released the revised "K-12 Computer Science Standards" in 2011. These standards can be found in Appendix C.

2.2.2 ISTE

The International Society for Technology in Education (ISTE) has compiled a set of standards for computer science educators. These standards are listed in Appendix **??**.

2.2.3 AP

Since 2003 the Advanced Placement Computer Science A exam has tested students' ability to solve problems and design algorithms using Java. The College Board's test is intended to cover the equivalent of a first semester college computer science course. The topics include:

1. Object-Oriented Program Design: Program and class design
2. Program Implementation: Implementation techniques. Programming constructs, Java library classes
3. Program Analysis: Testing, Debugging, Runtime exceptions, Program correctness, Algorithm analysis
4. Standard Data Structures: Primitive data types, Strings, Classes, Lists, Arrays (1-D and 2-D)
5. Standard Operations and Algorithms: Searching, Sorting
6. Computing in Context: System reliability, Privacy, Legal issues and intellectual property, Social and ethical ramifications of computer use

In 2016 the College Board plans to release a second AP CS course, AP Computer Science Principles, that will operate in tandem to the other AP CS class. From the AP website, "AP Computer Science Principles offers a multidisciplinary approach to teaching the underlying principles of computation." Unlike the AP CS A which relies exclusively on Java, the Principles course allows teachers to use any programming language. The course introduces key programming concepts with a "focus on fostering students to be creative." The "Big Ideas" covered by this exam include:

1. Creativity
2. Abstraction
3. Data and Information
4. Algorithms
5. Programming
6. Internet
7. Global Impact

For a detailed list of learning benchmarks associated with this exam, check out the AP Computer Science Principles Curriculum Framework.

2.2.4 Next Generation Science Standards

From Next Generation Science Standards:

> "Computer Science and Statistics Computer science and statistics are other areas of science that are not addressed here, even though they have a valid presence in K-12 education... Computer science, too, can be seen as a branch of the mathematical sciences, as well as having some elements of engineering. But, again, because this area of the curriculum has a history and a teaching corps that are generally distinct from those of the sciences, the committee has not taken this domain as part of our charge. Once again, this omission should not be interpreted to mean that computer science or statistics should be excluded from the K-12 curriculum. There are aspects of computational and statistical thinking that must be understood and applied in learning about the sciences, and we identify these aspects, along with mathematical thinking, in our discussion of science practices in Chapter 3."

AP Computer Science *A*	**AP Computer Science *Principles***
Curriculum:	
Focused on object-oriented programming and problem solving	Built around fundamentals of computing including problem solving, working with data, understanding the Internet, cybersecurity, and programming.
Language:	
Java is the designated programming language	Teachers choose the programming language(s)
Skillset:	
Encourages skill development among students considering a career in computer science or other STEM fields	Encourages a broader participation in the study of computer science and other STEM fields, including AP Computer Science A
Assessment:	
Multiple-choice and free-response questions (written exam)	Two performance tasks students complete during the course to demonstrate the skills they have developed (administered by the teacher; students submit digital artifacts), and multiple-choice questions (written exam)

TABLE 2.1: Comparison of AP Computer Science Exams [13]

Although the NGSS does not expliciting outline CS benchmarks, the Engineering Design standards are highly applicable to software engineering or computational thinking projects. These benchmarks are covered in Appendix D.

2.2.5 Common Core

2.3 Online Tools

2.4 Hardware Tools

Chapter 3

Newman Past and Present

3.1 Previous Programs

As Kafai and Burke [11] point out, in the 1980s many schools featured Basic or Logo programming, but by the mid 90s, CS declined due to:

1. lack of subject-matter integration.
2. no qualified instructors
3. rise of CD-ROMS and the internet, schools shifted emphasis from programming to software-specific proficiency and web surfing literacy.

The difference now:

1. plethora of coding sites
2. shift from code to applications
3. shift from tools to communities

[11]

3.2 Present Status

Chapter 4

Recommendations

4.1 Lower School

4.2 Middle School

Possible implementation strategies:

1. CS inegrated in the classroom
2. 6th grade science course
3. club time
4. flex time
5. new schedule

4.3 Upper School

Unlike the Middle School schedule, the Upper School sequence affords time and space for computer science electives, as well as a graduation requirement. The rotating eight block schedule means there are 32 possible credits, although Newman students rarely graduate with greater than [INSERT], typically averaging approximately [INSERT] credits. The student handbook stipulates that students must complete 24 academic credits to graduate, taken from the following subject areas:

The "elective" graduation requirement is the most obvious space to insert CS courses. It's worth examining which departments and classes may be impacted by these additional electives.

Subject	**Credits**
English	4
Mathematics	4
History/Social Studies	4
Science	3
World Languages	3
Arts	2
Physical Education	2
Electives	2
Total	**24**

TABLE 4.1: Upper School Graduation Requirements

[INSERT] these are the types of electives currently offered , number of students enrolled. Are art classes going to suffer? Do they make up the bulk of the extra electives?

what will senior year look like if you opt to take AP CS?

If the ultimate objective is to treat CS as foundational subject on par with core subjects like math and English, a critical analysis of the Upper School curriculum will be required to make space for a new set of mandated credits. Given the nascent status of computer science at Newman, this document will not attempt to evaluate the merits of the existing sequence or propose potentially subversive structural changes. It is, however, instructive to examine schools with robust CS requirements, .

[insert schools with lots of cs]

4.3.1 Stage 1

This document recommends, at a bare minimum, that the Upper School adopt a 1/2 credit CS graduation requirement. A mandated credit, as opposed to an optional elective, ensures that every student- especially groups typically underrepresented in CS- get exposure to coding prior to college and have the opportunity to explore coding career possibilities prior to choosing a college major or minor.

In addition to introducing foundational computational/coding concepts (explored in the "Benchmarks" section), this introductory CS course should prioritize the following objectives:

1. Explore diverse career opportunities afforded by CS/computational thinking
2. Emphasize the potential for innovation and creative expression through code

Assuming this 1/2 credit course is the single point of CS exposure, the course needs to serve as the hook. There are infinite possibilities for highly-engaging, interactive coding experiences (see

the Appendix [insert] for a table of tools and platforms), interactive Javascript visualizations, and so much more. A fun project-based course that explores a myriad of hardware and software tools is *both feasible and critically important* for an effective introductory CS experience.

Included in the Appendix are suggested curricula for two 1/2 credit courses designed to introduce CS from an art and engineering perspective respectively: "Creative Coding" and "Physical Computing." Each course is designed to meet the learning objectives of an introductory series as outlined above, specified by CSTA Benchmarks, and necessary to introduce AP CS topics.

[what other courses are we referencing insert]

- Staffing requirements would not do online for a graduation requirement; what's out there isn't engaging enough; doesn't afford exploration or convey the diversity of subjects possible with coding how many sections - space requirements

4.3.2 Stage 2

To develop a robust series of CS electives, 1/2 credit before CS principles? AP CS Principles

4.3.3 CS as a Core Subject

4.3.4 Innovation, Design Thinking, & CS

[insert 60% of jobs don't exist]

barriers? - teachers staffing Number of CS requirements * number of students per grade / average class size = number of sections number of sections / 5 = number of faculty

Appendix A

Appendix A: Coding Tools & Curriculum

The following table is compiled from EdSurge[14] and Code.Org[**?**]. For full descritions and side-by-side comparisons, visit Code.org's 3rd Educator Party Resources.

TABLE A.1: K-12 Coding Tools and Curriculum

Org	Curriculum	PD	Cost	Self-Guided?	Student Tracking?
LOWER SCHOOL					
Scratch Jr.	An iPad app that introduces coding cocepts through games and visual programming. Assessments and curriculum included.	-	FREE	Y	N
ScratchEd	A 6-unit intro to Scratch, a visual programming language for game creation, animations, and art.	In-person educator meet-ups and online MOOC, FREE	FREE	N	N
Code Studio (Code.org)	4 courses blend online tutorials with "unplugged" activities	1-day weekend workshops across the US, FREE	FREE	Y	Y
Project Lead The Way	6, 10-hour computer science modules	Face-to-face and online, $700 for school-level lead teacher	$750 /school	Y	Y

Tynker	Inspired by Scratch, Tynker has a dashboard to allow teachers to create a more structured way of teaching code with visual blocks. Includes assessment, classroom management, lesson plans, and a built in tutor.	2-day PD, $2000/day + travel	FREE tutorials. $399 /class, $2,000 /school	Y	Y
Kodable	Kodable is a freemium educational iPad game offering a kid-friendly introduction to programming concepts and problem solving.	-	FREE	Y	Y
SNAP!	SNAP!'s visual blocks (similar to Scratch) support higher level computer science concepts like recursion, procedures, and continuations, SNAP! can work with LEGO Mindstorms NXT.	-	FREE	N	N
Hopscotch	This free iPad app uses a visual programming language similar to Scratch to help kids learn the basics of programming logic, such as sequencing, loops, variables, functions and conditionals.	-	FREE	N	N
MIDDLE SCHOOL					
Bootstrap	Teach algebra through video-game programming, with a 20-hr module to go alongside or inside a math class	3-day workshops for schools and districts. Fees range	FREE	N	N
CodeHS	Year-long Introduction to Computer Science and AP Computer Science in Java.	Online PD for teaching intro computer science, 30-40 hour course, $1000/teacher	FREE Intro; Pro version is $2000/ section /year	Y	Y
Code.org	20-lesson CS in Algebra and CS in Science modules for math and science classes	In-person and online workshops available to partner districts, FREE, 50% of teacher stipends covered by Code.org	FREE	Y	Y

Globaloria	6 game-design courses	3-day, in-person training and ongoing online PD, fee included in student price	$75 /student	Y	Y
Project Lead The Way	6, 10-hour computer science modules	Face-to-face and online, $700 for school-level lead teacher	$750 /school	Y	Y
ScratchEd	A 6-unit intro to Scratch	In-person educator meet-ups and online MOOC, FREE	FREE	N	N
Tynker	Inspired by Scratch, Tynker has a dashboard to allow teachers to create a more structured way of teaching code with visual blocks. Includes assessment, classroom management, lesson plans, and a built in tutor.	2-day PD, $2000/day + travel	Free tutorials. $399 /class, $2,000 /school	Y	Y
SNAP!	SNAP!'s visual blocks (similar to Scratch) support higher level computer science concepts like recursion, procedures, and continuations, SNAP! can work with LEGO Mindstorms NXT.	See Scratch	FREE	N	N
NCLab	Courses in Karel coding, Turtle coding, 3D modeling, and Python.	Online	$899 /class	Y	Y
Pluaralsight	Free video-led coding courses for kids on Scratch, HTML, App Inventor, Kodu and Hopscotch	Online	FREE	Y	N
UPPER SCHOOL					
Beauty and Joy of Computing	Year-long CS Principles course	In-person in NYC, Berkeley, CA and North Carolina, FREE, stipends in NYC, stipends + travel elsewhere paid as available	FREE	N	N
Bootstrap	Teach algebra through video-game programming, with a 20-hr module to go alongside or inside a math class	3-day workshops for schools and districts. Fees range	FREE	N	N

Codecademy	6 online courses (Javascript, SQL, Python, Ruby on Rails, Git, HTML/CSS, and more) and 65 projects. FREE	Not provided	FREE	Y	Y
CodeHS	Year-long Introduction to Computer Science and AP Computer Science in Java.	Online PD for teaching intro computer science, 30-40 hour course, $1000/teacher	FREE Intro; Pro version is $2000/section/year	Y	Y
Code.org	Exploring Computer Science intro course and Computer Science Principles, a pilot course with an AP exam coming in 2016	In-person and online workshops available to partner districts, FREE, 50% of teacher stipends covered by Code.org	FREE	N	N
Edhesive	Year-long AP Computer Science course	Online PD, community and content/technical/program support available, $2,200 per school	FREE	Y	Y
Globaloria	6 game-design courses	3-day, in-person training and ongoing online PD, fee included in student price	$75/student	Y	Y
Mobile CSP	Year-long AP Computer Science Principles course based on App Inventor, a mobile programming language for Android devices. Materials available online	Online, regional in-person offered in CT, MA, NH and CA (others may be available), FREE, stipends available	FREE	Y	N
Project Lead The Way	Several courses: Intro CS, AP, Cybersecurity, and CS Applications	5 or 10-day in-person training, $1200 or $2400, depending on course	$2000 /school	Y	Y
Khan Academy	Online curriculum that teaches JavaScript programming, HTML/CSS, and SQL, in an interactive online environment, plus courses on Algorithms and Cryptography.	Online	FREE	Y	Y

NCLab	Courses in Karel coding, Turtle coding, 3D modeling, and Python.	Online	$899 /class	Y	Y
Hello Processing	Video tutorials and interactive coding environment for learning Processing.	Online	FREE	Y	N
Code Avengers	In-browser exercises and courses in JavaScript, HTML5, CSS3 and Python. Introductory courses are free, with intermediate and advanced courses for 29−39.	Online	FREE intro; intermediate and advanced courses for $29-$39.	Y	Y

—

Appendix B

Appendix B: Hardware Tools

physical computing Arduino Raspberri Pi Android Robotics

Sphero LEGO Robotics

LittleBits (?) Makey Makey + Scratch

—

Appendix C

Appendix C: CSTA Benchmarks

The Computer Science Teachers Association (CSTA) K-12 Computer Science Standards (revised 2011) [15] are broken into tables by level: K-3rd grade, 3rd-6th grade, level 2 (i.e. introductory high school sequence), level 3A, and level 3B. The benchmark subject areas include collaboration, computers and communication devices, computational thinking, community and ethical impacts, and computing practice and programming.

TABLE C.1: K-3rd Grade CS Benchmarks

ID	DESCRIPTION
	COLLABORATION
L0C01	Gather information and communicate electronically with others with support from teachers, family members, or student partners.
L0C02	Work cooperatively and collaboratively with peers, teachers, and others using technology
	COMPUTERS AND COMMUNICATION DEVICES
L0CD01	Use standard input and output devices to successfully operate computers and related technologies.
	COMPUTING PRACTICE AND PROGRAMMING
L0CP01	Use technology resources to conduct age-appropriate research.
L0CP02	Use developmentally appropriate multimedia resources (e.g., interactive books and educational software) to support learning across the curriculum.
L0CP03	Create developmentally appropriate multimedia products with support from teachers, family members, or student partners.
L0CP04	Construct a set of statements to be acted out to accomplish a simple task (e.g., turtle instructions).
L0CP05	Identify jobs that use computing and technology.
L0CP06	Gather and organize information using concept-mapping tools.
	COMPUTATIONAL THINKING
L0CT01	Use technology resources (e.g., puzzles, logical thinking programs) to solve ageappropriate problems
L0CT02	Use writing tools, digital cameras, and drawing tools to illustrate thoughts, ideas, and stories in a step-by-step manner.
L0CT03	Understand how to arrange (sort) information into useful order, such as sorting students by birth date, without using a computer.
L0CT04	Recognize that software is created to control computer operations.

L0CT05	Demonstrate how 0s and 1s can be used to represent information.

Table C.2: 3rd-6th Grade CS Benchmarks

ID	DESCRIPTION
	COLLABORATION
L1C01	Use productivity technology tools (e.g., word processing, spreadsheet, presentation software) for individual and collaborative writing, communication, and publishing activities.
L1C02	Use online resources (e.g., email, online discussions, collaborative web environments) to participate in collaborative problemsolving activities for the purpose of developing solutions or products.
L1C03	Identify ways that teamwork and collaboration can support problem solving and innovation.
	COMPUTERS AND COMMUNICATION DEVICES
L1CD1	Demonstrate an appropriate level of proficiency with keyboards and other input and output devices
L1CD2	Understand the pervasiveness of computers and computing in daily life (e.g., voice mail, downloading videos and audio files, microwave ovens, thermostats, wireless Internet, mobile computing devices, GPS systems).
L1CD3	Apply strategies for identifying simple hardware and software problems that may occur during use.
L1CD4	Identify that information is coming to the computer from many sources over a network.
L1CD5	Identify factors that distinguish humans from machines.
L1CD6	Recognize that computers model intelligent behavior (as found in robotics, speech and language recognition, and computer animation).
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS
L1CI01	Discuss basic issues related to responsible use of technology and information, and the consequences of inappropriate use.
L1CI02	Identify the impact of technology (e.g., social networking, cyber bullying, mobile computing and communication, web technologies, cyber security, and virtualization) on personal life and society.
L1CI03	Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and biases that occur in electronic information sources.
L1CI04	Understand ethical issues that relate to computers and networks (e.g., equity of access, security, privacy, copyright, and intellectual property).
	COMPUTING PRACTICE AND PROGRAMMING
L1CP01	Use technology resources (e.g., calculators, data collection probes, mobile devices, videos, educational software, and web tools) for problem-solving and self-directed learning.
L1CP02	Use general-purpose productivity tools and peripherals to support personal productivity, remediate skill deficits, and facilitate learning.
L1CP03	Use technology tools (e.g., multimedia and text authoring, presentation, web tools, digital cameras, and scanners) for individual and collaborative writing, communication, and publishing activities.
L1CP04	Gather and manipulate data using a variety of digital tools.
L1CP05	Construct a program as a set of step-by-step instructions to be acted out (e.g., make a peanut butter and jelly sandwich activity).

L1CP06	Implement problem solutions using a blockbased visual programming language.
L1CP07	Use computing devices to access remote information, communicate with others in support of direct and independent learning, and pursue personal interests.
L1CP08	Navigate between webpages using hyperlinks and conduct simple searches using search engines.
L1CP09	Identify a wide range of jobs that require knowledge or use of computing.
L1CP10	Gather and manipulate data using a variety of digital tools.
	COMPUTATIONAL THINKING
L1CT01	Understand and use the basic steps in algorithmic problem-solving (e.g., problem statement and exploration, examination of sample instances, design, implementation, and testing).
L1CT02	Develop a simple understanding of an algorithm (e.g., search, sequence of events, or sorting) using computer-free exercises.
L1CT03	Demonstrate how a string of bits can be used to represent alphanumeric information.
L1CT04	Describe how a simulation can be used to solve a problem.
L1CT05	Make a list of sub-problems to consider while addressing a larger problem.
L1CT06	Understand the connections between computer science and other fields.

Table C.3: Level 2 CS Benchmarks

ID	DESCRIPTION
	COLLABORATION
L2C01	Apply productivity/multimedia tools and peripherals to group collaboration and support learning throughout the curriculum.
L2C02	Collaboratively design, develop, publish, and present products (e.g., videos, podcasts, websites) using technology resources that demonstrate and communicate curriculum concepts.
L2C03	Collaborate with peers, experts, and others using collaborative practices such as pair programming, working in project teams, and participating in group active learning activities.
L2C04	Exhibit dispositions necessary for collaboration: providing useful feedback, integrating feedback, understanding and accepting multiple perspectives, socialization.
	COMPUTERS AND COMMUNICATION DEVICES
L2CD01	Recognize that computers are devices that execute programs.
L2CD02	Identify a variety of electronic devices that contain computational processors.
L2CD03	Demonstrate an understanding of the relationship between hardware and software.
L2CD04	Use developmentally appropriate, accurate terminology when communicating about technology.
L2CD05	Apply strategies for identifying and solving routine hardware problems that occur during everyday computer use.
L2CD06	Describe the major components and functions of computer systems and networks.
L2CD07	Describe what distinguishes humans from machines focusing on human intelligence versus machine intelligence and ways we can communicate.

L2CD08	Describe ways in which computers use models of intelligent behavior (e.g., robot motion, speech and language understanding, and computer vision).
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS
L2CI01	Exhibit legal and ethical behaviors when using information and technology and discuss the consequences of misuse.
L2CI02	Demonstrate knowledge of changes in information technologies over time and the effects those changes have on education, the workplace, and society.
L2CI03	Analyze the positive and negative impacts of computing on human culture.
L2CI04	Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources concerning real-world problems.
L2CI05	Describe ethical issues that relate to computers and networks (e.g., security, privacy, ownership, and information sharing).
L2CI06	Discuss how the unequal distribution of computing resources in a global economy raises issues of equity, access, and power.
	COMPUTING PRACTICE AND PROGRAMMING
L2CP01	Select appropriate tools and technology resources to accomplish a variety of tasks and solve problems.
L2CP02	Use a variety of multimedia tools and peripherals to support personal productivity and learning throughout the curriculum.
L2CP03	Design, develop, publish, and present products (e.g., webpages, mobile applications, animations) using technology resources that demonstrate and communicate curriculum concepts.
L2CP04	Demonstrate an understanding of algorithms and their practical application.
L2CP05	Implement problem solutions using a programming language, including: looping behavior, conditional statements, logic, expressions, variables, and functions.
L2CP06	Demonstrate good practices in personal information security, using passwords, encryption, and secure transactions.
L2CP07	Identify interdisciplinary careers that are enhanced by computer science.
L2CP08	Demonstrate dispositions amenable to openended problem solving and programming (e.g., comfort with complexity, persistence, brainstorming, adaptability, patience, propensity to tinker, creativity, accepting challenge).
L2CP09	Collect and analyze data that is output from multiple runs of a computer program.
	COMPUTATIONAL THINKING
L2CT01	Use the basic steps in algorithmic problemsolving to design solutions (e.g., problem statement and exploration, examination of sample instances, design, implementing a solution, testing, evaluation).
L2CT02	Describe the process of parallelization as it relates to problem solving.
L2CT03	Define an algorithm as a sequence of instructions that can be processed by a computer.
L2CT04	Evaluate ways that different algorithms may be used to solve the same problem.
L2CT05	Act out searching and sorting algorithms.
L2CT06	Describe and analyze a sequence of instructions being followed (e.g., describe a character's behavior in a video game as driven by rules and algorithms).
L2CT07	Represent data in a variety of ways including text, sounds, pictures, and numbers.

L2CT08	Use visual representations of problem states, structures, and data (e.g., graphs, charts, network diagrams, flowcharts).
L2CT09	Interact with content-specific models and simulations (e.g., ecosystems, epidemics, molecular dynamics) to support learning and research.
L2CT10	Evaluate what kinds of problems can be solved using modeling and simulation.
L2CT11	Analyze the degree to which a computer model accurately represents the real world.
L2CT12	Use abstraction to decompose a problem into sub problems.
L2CT13	Understand the notion of hierarchy and abstraction in computing including highlevel languages, translation, instruction set, and logic circuits.
L2CT14	Examine connections between elements of mathematics and computer science including binary numbers, logic, sets and functions.
L2CT15	Provide examples of interdisciplinary applications of computational thinking.

Table C.4: Level 3A CS Benchmarks

ID	DESCRIPTION
	COLLABORATION
L3AC01	Work in a team to design and develop a software artifact.
L3AC02	Use collaborative tools to communicate with project team members (e.g., discussion threads, wikis, blogs, version control, etc.).
L3AC03	Describe how computing enhances traditional forms and enables new forms of experience, expression, communication, and collaboration
L3AC04	Identify how collaboration influences the design and development of software products.
	COMPUTERS AND COMMUNICATION DEVICES
L3ACD01	Describe the unique features of computers embedded in mobile devices and vehicles (e.g., cell phones, automobiles, airplanes).
L3ACD02	Develop criteria for purchasing or upgrading computer system hardware.
L3ACD03	Describe the principal components of computer organization (e.g., input, output, processing, and storage).
L3ACD04	Compare various forms of input and output.
L3ACD05	Explain the multiple levels of hardware and software that support program execution (e.g., compilers, interpreters, operating systems, networks).
L3ACD06	Apply strategies for identifying and solving routine hardware and software problems that occur in everyday life.
L3ACD07	Compare and contrast client-server and peer-to-peer network strategies.
L3ACD08	Explain the basic components of computer networks (e.g., servers, file protection, routing, spoolers and queues, shared resources, and fault-tolerance).
L3ACD09	Describe how the Internet facilitates global communication.
L3ACD10	Describe the major applications of artificial intelligence and robotics.
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS
L3ACI01	Compare appropriate and inappropriate social networking behaviors.
L3ACI02	Discuss the impact of computing technology on business and commerce (e.g., automated tracking of goods, automated financial transactions, e-commerce, cloud computing).
L3ACI03	Describe the role that adaptive technology can play in the lives of people with special needs.

L3ACI04	Compare the positive and negative impacts of technology on culture (e.g., social networking, delivery of news and other public media, and intercultural communication).
L3ACI05	Describe strategies for determining the reliability of information found on the Internet.
L3ACI06	Differentiate between information access and information distribution rights.
L3ACI07	Describe how different kinds of software licenses can be used to share and protect intellectual property.
L3ACI08	Discuss the social and economic implications associated with hacking and software piracy.
L3ACI09	Describe different ways in which software is created and shared and their benefits and drawbacks (commercial software, public domain software, open source development).
L3ACI10	Describe security and privacy issues that relate to computer networks.
L3ACI11	Explain the impact of the digital divide on access to critical information.
	COMPUTING PRACTICE AND PROGRAMMING
L3ACP01	Create and organize Web pages through the use of a variety of web programming design tools.
L3ACP02	Use mobile devices/emulators to design, develop, and implement mobile computing applications.
L3ACP03	Use various debugging and testing methods to ensure program correctness (e.g., test cases, unit testing, white box, black box, integration testing)
L3ACP04	Apply analysis, design, and implementation techniques to solve problems (e.g., use one or more software lifecycle models).
L3ACP05	Use Application Program Interfaces (APIs) and libraries to facilitate programming solutions.
L3ACP06	Select appropriate file formats for various types and uses of data.
L3ACP07	Describe a variety of programming languages available to solve problems and develop systems.
L3ACP08	Explain the program execution process.
L3ACP09	Explain the principles of security by examining encryption, cryptography, and authentication techniques.
L3ACP10	Explore a variety of careers to which computing is central.
L3ACP11	Describe techniques for locating and collecting small and large-scale data sets.
L3ACP12	Describe how mathematical and statistical functions, sets, and logic are used in computation.
	COMPUTATIONAL THINKING
L3ACT01	Use predefined functions and parameters, classes and methods to divide a complex problem into simpler parts.
L3ACT02	Describe a software development process used to solve software problems (e.g., design, coding, testing, verification).
L3ACT03	Explain how sequence, selection, iteration, and recursion are building blocks of algorithms.
L3ACT04	Compare techniques for analyzing massive data collections.
L3ACT05	Describe the relationship between binary and hexadecimal representations.
L3ACT06	Analyze the representation and trade-offs among various forms of digital information.
L3ACT07	Describe how various types of data are stored in a computer system.
L3ACT08	Use modeling and simulation to represent and understand natural phenomena.

L3ACT09	Discuss the value of abstraction to manage problem complexity.
L3ACT10	Describe the concept of parallel processing as a strategy to solve large problems.
L3ACT11	Describe how computation shares features with art and music by translating human intention into an artifact.

Table C.5: Level 3B CS Benchmarks

ID	DESCRIPTION
	COLLABORATION
L3BC01	Use project collaboration tools, version control systems, and Integrated Development Environments (IDEs) while working on a collaborative software project.
L3BC02	Demonstrate the software life cycle process by participating on a software project team.
L3BC03	Evaluate programs written by others for readability and usability
	COMPUTERS AND COMMUNICATION DEVICES
L3BCD01	Discuss the impact of modifications on the functionality of application programs.
L3BCD02	Identify and describe hardware (e.g., physical layers, logic gates, chips, components).
L3BCD03	Identify and select the most appropriate file format based on trade-offs (e.g., accuracy, speed, ease of manipulation).
L3BCD04	Describe the issues that impact network functionality (e.g., latency, bandwidth, firewalls, server capability).
L3BCD05	Explain the notion of intelligent behavior through computer modeling and robotics.
	COMMUNITY, GLOBAL, AND ETHICAL IMPACTS
L3BCI01	Demonstrate ethical use of modern communication media and devices.
L3BCI02	Analyze the beneficial and harmful effects of computing innovations.
L3BCI03	Summarize how financial markets, transactions, and predictions have been transformed by automation.
L3BCI04	Summarize how computation has revolutionized the way people build real and virtual organizations and infrastructures.
L3BCI05	Identify laws and regulations that impact the development and use of software.
L3BCI06	Analyze the impact of government regulation on privacy and security.
L3BCI07	Differentiate among open source, freeware, and proprietary software licenses and their applicability to different types of software.
L3BCI08	Relate issues of equity, access, and power to the distribution of computing resources in a global society.
	COMPUTING PRACTICE AND PROGRAMMING
L3BCP01	Use advanced tools to create digital artifacts (e.g., web design, animation, video, multimedia)
L3BCP02	Use tools of abstraction to decompose a large-scale computational problem (e.g., procedural abstraction, object-oriented design, functional design).
L3BCP03	Classify programming languages based on their level and application domain
L3BCP04	Explore principles of system design in scaling, efficiency, and security.
L3BCP05	Deploy principles of security by implementing encryption and authentication strategies.

L3BCP06	Anticipate future careers and the technologies that will exist.
L3BCP07	Use data analysis to enhance understanding of complex natural and human systems.
L3BCP08	Deploy various data collection techniques for different types of problems.
	COMPUTATIONAL THINKING
L3BCT01	Classify problems as tractable, intractable, or computationally unsolvable.
L3BCT02	Explain the value of heuristic algorithms to approximate solutions for intractable problems.
L3BCT03	Critically examine classical algorithms and implement an original algorithm.
L3BCT04	Evaluate algorithms by their efficiency, correctness, and clarity.
L3BCT05	Use data analysis to enhance understanding of complex natural and human systems.
L3BCT06	Compare and contrast simple data structures and their uses (e.g., arrays and lists).
L3BCT07	Discuss the interpretation of binary sequences in a variety of forms (e.g., instructions, numbers, text, sound, image).
L3BCT08	Use models and simulations to help formulate, refine, and test scientific hypotheses.
L3BCT09	Analyze data and identify patterns through modeling and simulation.
L3BCT10	Decompose a problem by defining new functions and classes.
L3BCT11	Demonstrate concurrency by separating processes into threads and dividing data into parallel streams.

Appendix D

Appendix D: NGSS Standards

Table D.1: K-12 Engineering Design NGSS Standards [16]

ID	DESCRIPTION
K-2-ETS1-1.	Ask questions, make observations, and gather information about a situation people want to change to define a simple problem that can be solved through the development of a new or improved object or tool.
K-2-ETS1-2.	Develop a simple sketch, drawing, or physical model to illustrate how the shape of an object helps it function as needed to solve a given problem.
K-2-ETS1-3.	Analyze data from tests of two objects designed to solve the same problem to compare the strengths and weaknesses of how each performs.
3-5-ETS1-1.	Define a simple design problem reflecting a need or a want that includes specified criteria for success and constraints on materials, time, or cost.
3-5-ETS1-2.	Generate and compare multiple possible solutions to a problem based on how well each is likely to meet the criteria and constraints of the problem.
3-5-ETS1-3.	Plan and carry out fair tests in which variables are controlled and failure points are considered to identify aspects of a model or prototype that can be improved.
MS-ETS1-1.	Define the criteria and constraints of a design problem with sufficient precision to ensure a successful solution, taking into account relevant scientific principles and potential impacts on people and the natural environment that may limit possible solutions.
MS-ETS1-2.	Evaluate competing design solutions using a systematic process to determine how well they meet the criteria and constraints of the problem.
MS-ETS1-3.	Analyze data from tests to determine similarities and differences among several design solutions to identify the best characteristics of each that can be combined into a new solution to better meet the criteria for success.
MS-ETS1-4.	Develop a model to generate data for iterative testing and modification of a proposed object, tool, or process such that an optimal design can be achieved.
HS-ETS1-1.	Analyze a major global challenge to specify qualitative and quantitative criteria and constraints for solutions that account for societal needs and wants.
HS-ETS1-2.	Design a solution to a complex real-world problem by breaking it down into smaller, more manageable problems that can be solved through engineering.

HS-ETS1-3.	Evaluate a solution to a complex real-world problem based on prioritized criteria and trade-offs that account for a range of constraints, including cost, safety, reliability, and aesthetics as well as possible social, cultural, and environmental impacts.
HS-ETS1-4.	Use a computer simulation to model the impact of proposed solutions to a complex real-world problem with numerous criteria and constraints on interactions within and between systems relevant to the problem.

Bibliography

[1] K. Prottsman. Coding vs. programming – battle of the terms! *Huffington Post*, April 2015. URL http://www.huffingtonpost.com/kiki-prottsman/coding-vs-programming-bat_b_7042816.html.

[2] Carnegie Mellon Center for Computational Thinking. What is computational thinking? URL http://www.cs.cmu.edu/~CompThink/.

[3] Google for Education. Exploring computational thinking. URL https://www.google.com/edu/resources/programs/exploring-computational-thinking/.

[4] Moving beyond computer literacy: Why schools should teach computer science. *National Center for Women and Information Technology*. URL https://www.ncwit.org/resources/moving-beyond-computer-literacy-why-schools-should-teach-computer-science/moving-beyond.

[5] Mayor's Press Office. Mayor emanuel announces major gains in computer science 4 all program. *City of Chicago*, December 2014. URL http://www.cityofchicago.org/city/en/depts/mayor/press_room/press_releases/2014/dec/mayor-emanuel-announces-major-gains-in-computer-science-4-all-pr.html.

[6] Y. Koh. New education bill to get more coding in classrooms. *Wall Street Journal*, December 2015. URL http://blogs.wsj.com/digits/2015/12/10/new-education-bill-to-get-more-coding-in-classrooms/.

[7] Employment projections. *Bureau of Labor Statistics*, 2014. URL http://www.bls.gov/emp/tables.htm.

[8] Code.org. Summary of source data for code.org infographics and stats. 2015. URL https://docs.google.com/document/d/1gySkItxiJn_vwb8HIIKNXqen184mRtzDX12cux0ZgZk/pub.

[9] S. Adams. 25 college diplomas with the highest pay. *Forbes*, April 2013. URL http://www.forbes.com/pictures/mkl45kkeg/1-carnegie-mellon-school-of-computer-science/.

[10] Gallup. Searching for computer science: Access and barriers in u.s. k-12 education. 2014. URL http://csedu.gallup.com/.

[11] Quinn Burke Yasmin B. Kafai. Computer programming goes back to school. *The Phi Delta Kappan*, 95(1): 61–65, 2013. ISSN 00317217. URL http://www.jstor.org/stable/23617761.

[12] College Board. Ap® students in college: An analysis of five-year academic careers. 2007. URL http://research.collegeboard.org/sites/default/files/publications/2012/7/researchreport-2007-4-ap-students-college-analysis-five-year-academic-careers.pdf.

[13] College Board. Ap computer science principles curriculum framework. *Advances in AP*, December 2015. URL https://advancesinap.collegeboard.org/stem/computer-science-principles/course-details.

[14] K. Bell. A beginner's guide to bringing coding into the classroom. *edSurge*, November 2015. URL https://www.edsurge.com/news/2015-11-30-a-beginner-s-guide-to-bringing-coding-into-the-classroom.

[15] Computer Science Teachers Association (CSTA). K-12 computer science standards. 2011. URL http://www.csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf.

[16] Next Generation Science Standards. Engineering design standards. 2015. URL http://www.nextgenscience.org/search-standards-dci?field_idea_tid%5B%5D=113.