# CSTA K-12 Computer Science Standards
## Revised 2011

## CSTA Standards Task Force

Allen Tucker
Bowdoin College, Chair

Deborah Seehorn
North Carolina Department of Public Instruction, Chair

Stephen Carey
Brunswick School Department

Daniel Moix
Ouachita Technical College

Brian Fuschetto
Lyndhurst High School

Irene Lee
Santa Fe Institute

Dianne O'Grady-Cuniff
Howard High School

Chris Stephenson
Computer Science Teachers Association

Anita Verno
Bergen Community College

# CSTA Logo

The Association for Computing Machinery, Inc.
2 Penn Plaza, Suite 701
New York, New York 10121





ACM ISBN: #
ACM Order Number: #
Cost: $15.00
Additional copies may be ordered prepaid from:

ACM Order Department
P.O. Box 11405
Church Street Station
New York, NY 10286-1405

Phone: 1-800-342-6626
(U.S.A. and Canada)
+1-212-626-0500
(All other countries)
Fax: +1-212-944-1318
E-mail: acmhelp@acm.org

# CSTA K-12 Computer Science Standards

Allen Tucker—Bowdoin College
Deborah Seehorn—North Carolina Department of Public Instruction
Stephen Carey—Brunswick School District
Brian Fuschetto—Lyndhurst High School
Irene Lee—Santa Fe Institute
Daniel Moix—Ouachita Technical College
Dianne O'Grady-Cunnif—Howard High School
Chris Stephenson—Computer Science Teachers Association
Anita Verno—Bergen Community College



**Executive Summary**

This document provides comprehensive standards for K–12 computer science education that can be used to integrate computer science fluency and competency throughout primary and secondary schools, both in the United States and throughout the world. It is written in response to the pressing need to provide academic coherence between academic coursework and the rapid growth of computing and technology in the modern world, alongside the need for an educated public that can effectively utilize and build that technology most effectively to the benefit of society.

Computer science is an established discipline at the collegiate and post-graduate levels. It is best defined as "the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (p. 6). The integration of computer science concepts into the K–12 curriculum has not kept pace with other academic disciplines in the United States. As a result, the general public is not as well educated about computer science as it should be, and a serious shortage of computer scientists at all levels exists and may continue into the foreseeable future. These computer science standards aim to help address these problems. They provide a framework within which state departments of education and school districts can revise their curricula to better address the need to educate young people in this important subject area, and thus better prepare students for effective citizenship in the 21st century.

These standards provide a three-level framework for computer science. The first two levels are aimed at grades K–6 and 6–9 respectively. It is expected that the learning outcomes in Level 1 will be addressed in the context of other academic subjects. The learning outcomes in Level 2 may be addressed either through other subjects or in discrete computer science courses. Level 3 is divided into three separate courses: *Computer Science in the Modern World*, *Computer Science Principles*, and *Topics in Computer Science*. The standards provided in *Computer Science in the Modern World* reflect learning content that should be mastered by all students. *Computer Science Principles*, and *Topics in Computer Science* are courses intended for students with special interest in computer science and other computing careers, whether they are college-bound or not.

These recommendations are not made in a vacuum. We understand the serious constraints under which school districts are operating and the uphill battle that computer science faces in the light of other educational priorities, as well as time and budget constraints. Thus, we conclude this report with a series of recommendations that are intended to provide support for a long-term evolution of computer science in K–12 schools. Significant progress has been made since the *ACM Model Curriculum for K–12 Computer Science Education* was first published in 2003 (a revised edition was also published in 2006). Many follow-up efforts are still needed, however, to sustain the momentum these standards generate. Teacher training, curriculum innovation, teaching resources, and dissemination are but a few of these challenges.

We hope these learning standards will serve as a catalyst for widespread adoption of computer science education for all K–12 students. We invite you to read the entire document and then to take part in the effort to implement these standards in a way that mutually benefits both you and the K–12 education community. Additional information about ongoing activities to support computer science education in K–12 can be found at the Computer Science Teachers Association's Web site (csta.acm.org).

Table of Contents Page

# National Standards for K-12 Computer Science

## 1. Introduction

The digital age has transformed the world and workforce and, as a result, computer science and the technologies it enables now lie at the heart of our economy, our daily lives, and our scientific enterprises. To be well-educated citizens in a computing-intensive world and to be prepared for careers in the 21st Century, it is imperative that students have a clear understanding of the principles and practice of computer science. No other subject will open as many doors in the 21st Century, regardless of a student's ultimate field of study or occupation, as computer science. As the report *Running on Empty: The Failure to Teach Computer Science in the Digital Age* (ACM/CSTA, 2010) clearly points out, at a time when computing is driving job growth and new scientific discovery, it is unacceptable that roughly two-thirds of the fifty-states in the U.S. do not have computer science standards for secondary school education, K–8 computer science standards often confuse computer science and simple application use, few states count computer science as a core academic subject for graduation, and computer science teacher certification is seriously flawed. These are national failings and ones that we can ill afford in this digital age.

The purpose of this document is to set forth, at all stages of their learning, the computer science knowledge and skills that students must have to enable them to thrive in this new global information economy. It defines a set of national standards for K–12 computer science and suggests steps that will be needed to enable their implementation. By implementing these standards, schools can introduce the principles and methodologies of computer science to all students, whether they are college bound or workplace bound. The standards outlined in this document complement existing K–12 computer science and IT curricula where they are already established, especially the advanced placement (AP) computer science curricula (AP, 2010) and professional IT certifications.

Much evidence (National Research Council, 1999) confirms an urgent need to improve the level of public understanding of computer science as an academic and professional field, including its distinctions from management information systems (MIS), information technology (IT), mathematics, and the other sciences. Elementary and secondary schools have a unique opportunity and responsibility to address this need. That is, to function in society, every citizen in the 21$^{st}$ century must understand at least the principles of computer science. A broad commitment to, and rigorous implementation of, K–12 computer science courses not only will create such broad public understanding but also will help to address the worldwide shortage of computer specialists.

The delineation of a core set of learning standards that can provide the foundation for a complete computer science curriculum and its implementation at the K–12 level is a necessary first step toward reaching these goals. In summary, these standards will:

1. introduce the fundamental concepts of computer science to all students, beginning at the elementary school level;
2. present computer science at the secondary school level in a way can fulfill a computer science, math, or science graduation credit;
3. offer additional secondary-level computer science courses that will allow interested students to study facets of computer science in more depth and prepare them for entry into the work force or college; and
4. increase the availability of computer science for all students, especially those who are members of underrepresented groups.

Our goal is for these standards to be coherent and comprehensible to teachers, administrators, and policy makers. For this reason, our discussions in the early part of this document focus on the current proliferation and confusion of terms that often make this discipline seem ill defined and incomprehensible to those outside the field. We also attempt to describe the importance of computer science education as part of the intellectual development of students at all levels emphasizing and the linkages between computer science and innovation across all disciplines.

All drafts of this report have been informed by feedback from many sources; we hope that this final draft will receive widespread dissemination and continued scrutiny from everyone who has interests or experience in K–12 education. To that end, these standards are published on the CSTA web site (http://csta.acm.org) as well as in hardcopy. Feedback has been actively sought from the following professional organizations:

Academy of Information Technology/National Academy Foundation (AOIT/NAT)
Association for Computing Machinery (ACM) Special Interest Group for Computer Science Education (SIGCSE)
ACM Education Board
Association for Supervision and Curriculum Development (ASCD) Curriculum Directors in school districts
Institute of Electrical and Electronics Engineers (IEEE) Computer Society Educational Activities Board
International Society for Technology in Education (ISTE) Special Interest Group for Computer Science (SIGCS)
National Association of Secondary School Principals (NASSP)
National Education Association (NEA)
National School Board Association (NSBA)

We recognize that while many of the recommendations in this report are ambitious but we believe that they are critical to ensuring that students achieve the necessary level of knowledge, skills, and experience to thrive in the modern world. We offer this work as a comprehensive set of standards—an ideal toward which many districts can evolve over time. This report thus provides a catalyst for a long-term process—it defines the "what" from which the "how" will emerge during the next several years.

## 2. Computer Science as a Core Discipline

It is not an exaggeration to say that our lives depend upon computer systems and the people who maintain them to keep us safe on the road and in air, help physicians diagnose and treat health care problems, and play a critical role in the development of many scientific advances. A fundamental understanding of computer science enables students to be both educated users of technology and innovators capable of designing computing systems to improve the quality of life for all people.

We have observed that children of all ages love computers. When given the opportunity, young students enjoy the sense of mastery and magic that programming provides. Older students are drawn to the combination of art, narrative, design, programming, and sheer enjoyment that comes from creating their own virtual worlds. Blending computer science with other interests also provides rich opportunities for learning. Students with an interest in music, for example, can learn about digital music and audio, a field that integrates electronics; several kinds of math; music theory; computer programming; and a keen ear for what sounds beautiful, harmonious, or just plain interesting.

Computer science has also made possible profound leaps of innovation and imagination as it facilitates our efforts to solve pressing problems (for example, the prevention or cure of diseases, the elimination of world hunger). It expands our understanding of ourselves as biological systems and of our relationship to the world around us. These advances, in turn, drive the need for educated individuals who can bring the power of computing to help solve complex problems. It is no longer sufficient to wait until students are in college to introduce these concepts. All of today's students will go on to live a life heavily influenced by computing, and many will work in fields that directly involve computing. They must begin to work with algorithmic problem solving and computational methods and tools in K–12.

The following sections explore the benefits of computer science education in more detail.

### 2.1 *Computer Science is Important Intellectually*

The invention of the computer in the 20th century was a “once in a millennium” event, comparable in importance to the development of writing or the printing press. Computers are fundamentally different from other technological inventions in that they directly augment human thought, rather than, say, the functions of our muscles or our senses. Computers have enormous impact on the way we live, think, and act. It is hard to overestimate their importance in the future. In fact, many believe that the true computer revolution will not happen until everyone can understand the technology well enough to use it in truly innovative ways.

So why is it important to study computer science? We live in a digitized, computerized, programmable world, and to make sense of it, we need computer science. An engineer using a computer to design a bridge must understand how the maximum capacity estimates were computed and how reliable they are. An educated citizen using a voting machine or bidding in an eBay auction should have a basic understanding of the underlying algorithms of such conveniences, as well as the security and privacy issues that arise when information is transmitted and stored digitally.

Computer science students learn logical reasoning, algorithmic thinking, design and structured problem solving—all concepts and skills that are valuable well beyond the computer science classroom. Students gain awareness of the resources required to implement, test, and deploy a solution and how to deal with real-world constraints. These skills are applicable in many contexts, from science and engineering to the humanities and business, and they have enabled deeper understanding in these and other areas. Computer simulations are essential to the discovery and understanding of the fundamental rules that govern a wide variety of systems from how ants gather food to how stock markets behave. Computer science is also one of the leading disciplines helping us understand how the human mind works, one of the great intellectual challenges of all time. Thus, much computer-enabled innovation lies ahead of us and computer science is an essential tool for achieving our vast potential.

### 2.2 *Computer Science Leads to Multiple Career Paths*

The vast majority of careers in the 21st century will require an understanding of computer science. Many jobs that today’s students will have in 10 to 20 years haven’t been invented yet. Professionals in every discipline—from artists and entertainers, to communications and health care professional, to factory workers, small business owners, and retail store staff— need to understand computing to be productive and competitive in their fields. Thomas Friedman, in his best-selling book *The World is Flat* (2006), argues that our economy most needs “Versatilists,” people who have expertise both in some domain and in technology. Computer science is the glue that makes it possible for Versatilists to bridge domain specific expertise and technological innovation.

There is an unmistakable link between success, innovation, and computer science. Movies like *The Incredibles* and *Lord of the Rings* required the development of new computing techniques. Progress on understanding the genetics of disease or creating an AIDS vaccine requires professionals to think in terms of computer science—these problems are unsolvable without it. Those who understand the technology can make the new movies and invent the new techniques, and they are the professionals who will go beyond simply using what others have invented.

Studying computer science can prepare a student to enter many career areas, both within and outside of computing. Despite the depressing reports in the media, the reality is that professionals with computer science training have never been more in demand in the U.S. than they are today. Network managers need computer science expertise to install new kinds of routers. So do database designers who help people represent their data in a form that the computer can manipulate. Contrary to public perception, professional computer scientists rarely spend all of their days writing program code. Often they are working with experts in other fields, designing and building computer systems that support the functioning of modern society. In addition computing skills are now beneficial, if not required for work in almost any profession.

### *2.3 Computer Science Teaches Problem Solving*

Artists, philosophers, designers, and scientists in all disciplines all share the intensely creative activity of problem solving. Every painting by Picasso is an attempt to solve the problem of capturing an active, three-dimensional world on a flat canvas. Every TV commercial during the Super Bowl is an attempt to solve the problem of how to entice people to want, and then purchase, a product. And every well-designed scientific experiment produces data to support or refute a theory.

Computer science teaches students to think about the problem-solving process itself. In computer science, the first step in solving a problem is to state it clearly and unambiguously. Often a computer scientist works closely with business people, scientists, and other experts to understand the issues, and to define the problem so explicitly that it can be represented in a computer. This cooperative process requires people with different expertise and perspectives to work together to clarify the issues while considering each other's priorities and constraints. A computer expert helping to design a new computer system for a medical office, for example, has to take into account the current work flow, patient privacy concerns, training needs for new staff, current and upcoming technology, and of course, the budget.

Once the problem is well defined, a solution must be created. Computer hardware and peripheral devices must be selected or built. Computer programs must be designed, written, and tested. Existing software systems and packages may be modified and integrated into the final system. In all phases, the computer scientist thinks about reflective use of computer time and shared resources. Building a system is a creative process that also requires scientific thinking. With each fix of a bug or addition of a new feature, there's a hypothesis that the problem has been solved. Experiments are designed, data are collected, results are analyzed, and if the hypothesis is untrue, the cycle repeats.

A computer scientist is concerned with the robustness, the user-friendliness, the maintainability, and above all the correctness of computer solutions to business, scientific, and engineering problems. These issues often require both intense analysis and creativity. How will the system respond if the power goes out, or two nurses try to access the same patient record simultaneously, or the insurance company's system is changed, or someone enters unexpected data into the system? Cooperation is again the key. The users and clients have to think about how the system will be used in day-to-day life and anticipate its use in the future. Computer specialists draw on their training and experience to confront problems and to create the best possible solutions.

### *2.4 Computer Science Supports and Links to Other Sciences*

Progress in science has always been linked with progress in technology and vice versa. For example, bacteria were first discovered not by a biologist but rather by a Dutch merchant who refined the art of making microscope lenses (and enjoyed peering at plaque he scraped off his unbrushed teeth). Nowadays, it's typical for computer scientists to work in other scientific disciplines. To solve the big scientific problems of the 21st century, such as grappling with new diseases and climate change, we will need people with diverse skills, abilities, and perspectives.

Computer science can also help us learn what it really means to be human. The sequencing of the human genome in 2001 was a landmark achievement of molecular biology, which would not have been possible without computer scientists. After short DNA fragments of the genome were sequenced in biology labs, computers were used to figure out how to piece the fragments together. This knowledge is paving the way for better computational methods of detecting and curing diseases, such as cancer, because we are now better able to simulate and hence understand the genetic mutations involved.

It doesn't take a neuroscientist to appreciate the fact that the human brain is complex and amazing. We know, for example, that an infant can effortlessly recognize a familiar face from many different viewpoints, and yet, we have a very poor understanding of the computational mechanisms that the brain uses to solve such tasks. Inferring meaning from images is a computational task, and computer scientists and neuroscientists are working together to figure out how to build computers that can process images and, ultimately, how we can better understand intelligence itself. The use of modeling and simulation, visualization, and management of massive data sets has fostered the emergence of a new field that bridges science, technology, engineering and math—computational science. This field integrates many aspects of computer science such as the design of algorithms and graphics with their application in the sciences.

In science classes, students use sophisticated simulation software to make molecules and geological processes come to life. Writing computer programs that model behavior allows scientists to generate results and test theories that are impossible to test in the physical world. Advances in weather prediction, for example, are largely dependent upon computer modeling and simulation. Computational methods have also transformed fields such as statistics and chemistry. Scientists who can understand and contribute to technological innovation have a huge advantage. Good training for future scientists must therefore include a solid basis in computer science.

### *2.5 Computer Science Can Engage All Students*

Computer science applies to virtually every aspect of life, so that it can be easily tied to the myriad of student interests. For example, students who are fascinated with specific technologies such as cell phones may have an innate passion for visual design, digital entertainment, or helping society. K–12 computer science teaching can thus nurture students' interests, passions, and sense of engagement with the world around them by offering opportunities for solve computational problems relevant to their own life experiences.

Pedagogically, computer programming has the same relation to studying computer science as playing an instrument does to studying music or painting does to studying art. In each case, even a small amount of hands-on experience adds immensely to life-long appreciation and understanding, even if the student does not continue programming, playing, or painting as an adult. Although becoming an expert programmer, violinist, or oil painter demands much time and talent, we still want to expose every student to the joys of being creative. The goal for teaching computer science should be to get as many students as possible enthusiastically engaged with every assignment. We can provide students with the tools to design and write programs that control their cell phones or robots, create physics and biology simulations, or compose music. Students will want to learn to use conditionals, loops, parameters, and other fundamental concepts just to make these exciting things happen.

In a fast-paced field such as computer science, we are all challenged to keep up with our peers and our students. Technology changes rapidly, and students are sometimes more likely than teachers to be familiar with the latest incarnations. No teacher should be apprehensive of learning from her or his students. Real learning involves everyone in the room living with a sense of wonder and anticipation.

We know that teaching computer science involves some unique challenges and that none of us has all of the answers. The CSTA Source web repository at http://csta.acm.org/WebRepository/WebRepository.html provides a comprehensive collection of resources for teachers These resources have been found to be helpful in our attempts to better interest, engage, and motivate our students. Not all of them will be completely applicable to every classroom, but we believe that many contain useful and varied suggestions that may inspire both students and teachers alike.

## 3. Defining the Terminology

Unlike many other disciplines, computer science is constantly being reshaped. New thinking and new technologies continue to expand our understanding of what computer scientists can and need to know. This has resulted in considerable debate concerning the definition of computer science itself. Before discussing K–12 curriculum standards, we first therefore clarify the context in which the standards are set as well as address some of the confusion currently swirling around the proliferation of terms used to describe the various kinds of computing education.

For high school educators, perhaps the most profound confusion arises when trying to distinguish between the three most common areas of computing education offered in schools. While each of these areas has been known by various names, for the purposes of this discussion we call them:

- Educational Technology
- Information Technology, and
- Computer Science.

*Educational Technology* can be defined as using computers across the curriculum, or more specifically, using computer technology (hardware and software) to learn about other disciplines. For example, the science teacher may use pre-existing computer simulations to provide students with a better understanding of specific physics principles, or an English teacher may use word-processing software to help students improve their editing and revision skills. While educational technology is concerned with using these tools, computer science is concerned with designing, creating, testing, and verifying these tools.

*Information technology* (IT) as defined by Tucker, Deek, Jones, McCowan, Stephenson, and Verno (2003) is "the proper use of technologies by which people manipulate and share information in its various forms" (p. 6). While Information Technology involves learning about computers, it emphasizes the technology itself. Information Technology specialists assume responsibility for selecting appropriate hardware and software products, integrating those products with organizational needs and infrastructure, and installing, customizing and maintaining those resources. Information Technology courses, therefore, focus on:

- installing, securing, and administering computer networks;
- installing, maintaining, and customizing software;
- managing and securing data in the physical and virtual world;
- managing communication systems;
- designing, implementing, and managing web resources; and
- developing and managing multimedia resources and other digital media.

While IT is an applied field of study, driven by the practical benefits of its knowledge, computer science adds scientific and mathematical, as well as practical, dimensions. Some of the practical dimensions of computer science are shared with IT, such as working with text, graphics, sound, and video. But while IT concentrates on learning how to use and apply these tools, computer science is concerned about learning how these tools are designed. While computer science and IT have a lot in common, neither one is fully substitutable for the other. For example, the complexity of algorithms is a fundamental idea in computer science but would probably not appear in an IT curriculum.

*Computer Science,* on the other hand, spans a wide range of computing endeavors, from theoretical foundations to robotics, computer vision, intelligent systems, and bioinformatics. The work of computer scientists is concentrated in three areas:

- designing and implementing software,
- developing effective ways to solve computing problems, and
- devising new ways to use computers.

For the purposes of this document, we rely heavily on the definition of computer science provided by Tucker et al. (2003) as we believe that this definition of computer science has the most direct relevance to high school computer science education. They defined the discipline as follows:

"*Computer science (CS)* is the study of computers and algorithmic processes, including their principles, their hardware and software designs, their applications, and their impact on society" (p. 6). They argue that in order to fulfill this definition, K–12 computer science curricula must include the following elements:

- programming,
- hardware design,
- networks,
- graphics,
- databases and information retrieval,
- computer security,
- software design,
- programming languages,
- logic,
- programming paradigms,
- translation between levels of abstraction,
- artificial intelligence,
- the limits of computation (what computers can't do),

- applications in information technology and information systems, and
- social issues (Internet security, privacy, intellectual property, etc.).

Citing the conclusion of the National Research Council (1999) that a basic understanding of all these topics is now an essential ingredient to preparing high school graduates for life in the 21st century, Tucker et al. (2003) further argued that the goals of a K–12 computer science curriculum are to:

- introduce the fundamental concepts of computer science to all students, beginning at the elementary school level,
- present computer science at the secondary school level in a way that would be both accessible and worthy of a curriculum credit (e.g., math or science),
- offer additional secondary-level computer science courses that will allow interested students to study it in depth and prepare them for entry into the work force or college, and
- increase the knowledge of computer science for all students, especially those who are members of historically underrepresented groups in computer science.

Two other terms that often appear in discussions of computing education are *Information Technology Literacy* and *Information Technology Fluency* (National Research Council, 1999). A recent National Academy study (National Research Council, 1999) defines *IT fluency* as something more comprehensive than IT literacy. Whereas IT literacy is the capability to use *today's* technology in one's own field, the notion of IT fluency adds the capability to independently *learn* and use *new* technology as it evolves (National Research Council, 1999) throughout one's professional lifetime. Moreover, IT fluency also includes the active use of computational thinking (including programming) to solve problems, whereas IT literacy does not. The idea of IT fluency (National Research Council, 1999) was proposed as a minimum standard that all college students should achieve by the time they graduate. Most colleges and universities (e.g., see National Research Council, 1999) have implemented these or similar standards and are expecting their graduates to achieve them. In this document, we strongly support the contention that this minimum standard should be implemented at the K–12 level as well.



## 4. Organization of the Learning Outcomes: Levels and Strands

Building on the lessons of the past and the needs of the present and the future, we propose a three-level model for K–12 computer science that focuses on fundamental concepts and has the following general goals:

1. The curriculum should prepare students to understand the nature of computer science and its place in the modern world.
2. Students should understand that computer science interweaves concepts and skills.
3. Students should be able to use computer science skills (especially computational thinking) in their problem-solving activities in other subjects.
4. The computer science standards should complement IT and AP computer science curricula in schools where they are currently offered.

If these standards are widely implemented and these goals are met, high school graduates will be prepared to be knowledgeable users and critics of computers, as well as designers and builders of computing applications that will affect every aspect of life in the 21$^{st}$ century.

### 4.1 Levels

The CSTA Standards for K–12 computer science are based on a three-level model with each of the levels representing a specific educational level. Level 1 provides the learning standards for students in Grades K–6, Level 2 provides the learning standards for students in Grades 6–9, and Level 3 provides the learning standards for students in each of three discrete courses in grades 9–12. (We note that the boundaries specified for each Level will vary from school to school.) The overall structure of this model is shown in Figure 1.

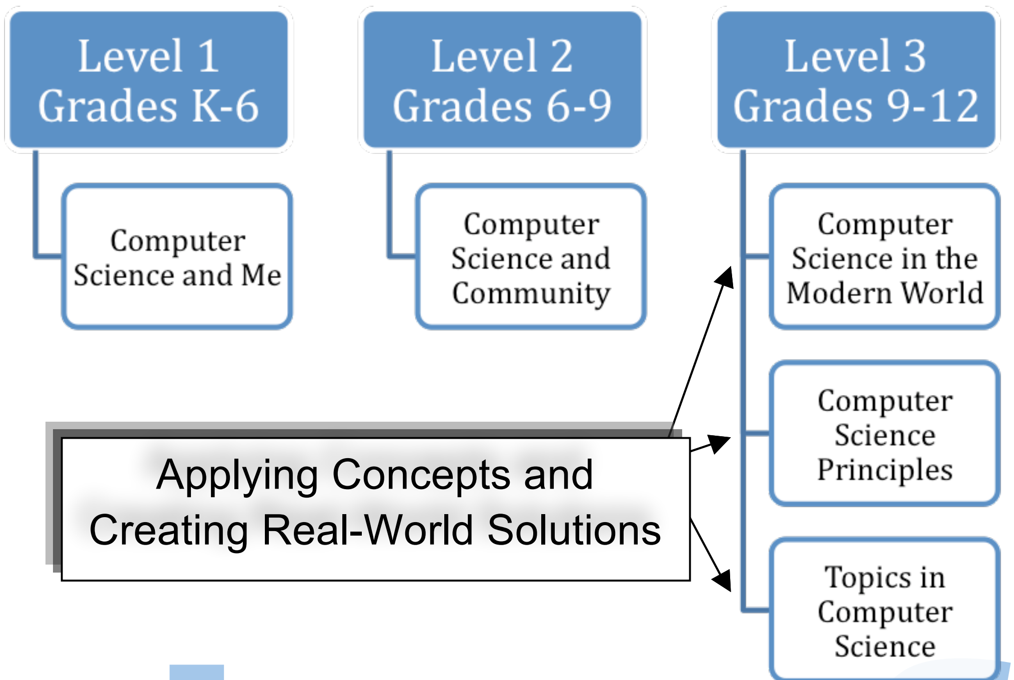



**Figure 1. Organizing Structure for the Computer Science Standards**



***Level 1* (recommended for grades K–6) *Computer Science and Me*:** elementary school students are introduced to foundational concepts in computer science by integrating basic skills in technology with simple ideas about computational thinking. The learning experiences created from these standards should be inspiring and engaging, helping students see computing as an important part of their world. They should be designed with a focus on active learning, creativity, and exploration and will often be embedded within other curricular areas such as social science, language arts, mathematics, and science.

***Level 2* (recommended for grades 6–9) *Computer Science and Community*:** middle school/junior high school students begin using computational thinking as a problem-solving tool. They begin to appreciate the ubiquity of computing and the ways in which computer science facilitates communication and collaboration. Students begin to experience computational thinking as a means of addressing community-relevant issues. The learning experiences created from these standards should be relevant to the students and should promote their perceptions of themselves as proactive and empowered problem solvers. They should be designed with a focus on active learning and exploration and can be taught within explicit computer science courses or embedded in other curricular areas such as social science, language arts, mathematics, and science.

***Level 3* (recommended for grades 9–12) *Applying concepts and creating real-world solutions:*** Level 3 is divided into three discrete courses, each of which focuses on different facets of computer science as a discipline. Throughout these courses, students can master more advanced computer science concepts and apply those concepts to develop virtual and real-world artifacts. The learning experiences created from these standards should focus on the exploration of real-world problems and the application of computational thinking to the development of solutions. They should be designed with a focus on collaborative learning, project management, and effective communication. Level 3 includes the following courses:

*Level 3A: (*recommended for grades 9 or 10) *Computer Science in the Modern World:* This course is recommended for all students. Its goal is to solidify students' understanding of computer science principles and practices so that they can make informed choices and use appropriate computational tools and techniques in whatever career they decide to pursue. They should also appreciate the breadth of computing and its influence in almost every aspect of modern life. Finally, they should understand the social and ethical impact of their various choices when using computing technology in their work and personal lives.

*Level 3B: (*recommended for grades 10 or 11) *Computer Science Principles:* This course is a more in-depth study of computer science and its relation to other disciplines, and contains a significant amount of algorithmic problem solving and related activities. One way to realize this course is by following the Computer Science Principles course (www.apcsprinciples.org). Students should come out of this course with a clear understanding of the application of computational thinking to real-world problems. They should also have learned how to work collaboratively to solve a problem and use modern collaboration tools during that work.

*Level 3C: (*recommended for grades 11 or 12) *Topics in Computer Science:* This is an elective course that provides depth of study in one particular area of computing. This may be, for example, an AP Computer Science A (AP, 2010) course, which offers depth of study in Java programming. Alternatively, this offering may be a projects-based course focusing on a single facet of computing or a course that leads to professional computing certification. Any Level 3 course will typically require at least the Level 2 course as a prerequisite.

## 4.2 Strands

Almost since its inception, computer science has been hampered by the perception that it focuses exclusively on programming. This misconception has been particularly damaging in grades K–12 where it often has led to courses that were exceedingly limited in scope and negatively perceived by students. It also fed into other unfortunate perceptions of computer science as a solitary pursuit, disconnected from the rest of the world and of little relevance to the interests and concerns of students.

We address these concerns by distinguishing five complementary and essential strands throughout all three levels in these standards. These strands are: computational thinking; collaboration; computing practice; computers and communication devices; and community, global and ethical impacts. These strands not only demonstrate the richness of computer science but also help organize the subject matter for students so that they can begin to perceive of computer science as more engaging, relevant, and more than a solitary pursuit. Figure 2 shows these strands graphically.




**Figure 2. Strands in the Computer Science Standards**

The following subsections discuss how these five strands can help students enrich their understanding and mastery of computer science during their formative years. More detailed discussions appear later in this report.

### 4.2.1 Computational Thinking

We believe that computational thinking (CT) can be used across all disciplines to solve problems, design systems, and improve understanding of the power and limitations of computing in the modern age. The study of computational thinking enables all students to better conceptualize, analyze, and solve complex problems by selecting and applying appropriate strategies and tools, both virtually and in the real world.

K–12 education is a highly complex, highly-politicized environment where multiple competing priorities, ideologies, pedagogies, and ontologies all vie for attention. It is also subject to widely diverse expectations, intense scrutiny, and diminishing resources. Any effort to achieve systemic change in this environment requires a deep understanding of these realities. Passionate debate about the nature of computer science or computational thinking may provide intellectual stimulation for those in the computing fields. However, embedding computational thinking in grades K–12 requires a practical approach, grounded in an operational definition.

Developing an approach to computational thinking that is suitable for K–12 students is especially challenging in light of the fact that there is, as yet, no widely agreed upon definition of computational thinking. For the purposes of this document, we rely upon the definition of computational thinking developed during a series of workshops hosted by the International Society for Technology in Education (ISTE) and the Computer Science Teachers Association (CSTA) and reported by Barr and Stephenson (2011).

“CT is an approach to solving problems in a way that can be implemented with a computer. Students become not merely tool users but tool builders. They use a set of concepts, such as abstraction, recursion, and iteration, to process and analyze data, and to create real and virtual artifacts. CT is a problem solving methodology that can be automated and transferred and applied across subjects. The power of computational thinking is that it applies to every other type of reasoning. It enables all kinds of things to get done: quantum physics, advanced biology, human- computer systems, development of useful computational tools.”

Computational thinking is thus a problem solving methodology can interweave computer science with all disciplines, providing a distinctive means of analyzing and developing solutions to problems that can be solved computationally. With its focus on abstraction, automation, and analysis, computational thinking is a core element of the broader discipline of computer science and for that reason it is interwoven through these computer science standards at all levels of K–12 learning.

### 4.2.2 Collaboration

Computer science is an intrinsically collaborative discipline. Significant progress is rarely made in computer science by one person working alone. Typically, computing projects involve large teams of computer scientists working together to design, code, test, debug, describe, and maintain software over time. New programming methodologies such as pair programming emphasize the importance of working together. Additionally, development teams working with discipline-specific experts ensure the computational solutions are appropriate, effective, and efficient. Developing collaboration skills is thus an important part of these K–12 national computer science standards.

In elementary school, students can begin to work cooperatively with fellow students and teachers using technology. They learn to gather information and communicate with others using a variety of traditional and mobile communication devices. They also learn to use online resources and participate in collaborative problem solving activities. These collaborative activities continue into middle school, where students apply multimedia and productivity tools for group learning exercises. In secondary school students enhance their collaborative abilities by participating in teams to solve software problems that are relevant to their daily lives. Skills learned at this level can include teamwork, constructive criticism, project planning and management, and team communication.

### 4.2.3 Computer and Communications Devices

K–12 students at all levels should understand the elements of modern computer and communication devices and networks. They should also understand how the Internet facilitates global communication and how to practice good Internet citizenship. Students should also use appropriate and accurate terminology when communicating about technology

At the elementary school level, students are introduced to many devices and media that can assist them with their learning activities, both within computer science and in other disciplines. Middle school students begin discriminating among different devices and their uses. They should also be able to describe the basic components of computers and computer networks. For instance, they should understand the organization of Web pages, URLs, and search engines. Secondary school students should understand computational devices in more detail, learning to form abstract ideas about specific components (e.g., input, output, processors, and databases) and their roles in the computational spectrum. Students should also understand why a compiler translates software into a machine-executable form.

### 4.2.4 Community, Global, and Ethical Impacts

The ethical use of computers and networks is a fundamental aspect of computer science at all levels. As soon as students begin using the Internet, they should learn the norms for its ethical use. Principles of personal privacy, network security, software licenses, and copyrights must be taught at an appropriate level in order to prepare students to become responsible citizens in the modern world. Students should be able to make informed and ethical choices among various types of software such as proprietary and open source and understand the importance of adhering to the licensing or use agreements. Students should also be able to evaluate the reliability and accuracy of information they receive from the Internet.

Computers and networks are a multicultural phenomenon that effect society at all levels. It is essential that K–12 students understand the impact of computers on international communication. They should learn the difference between appropriate and inappropriate social networking behaviors. They should also appreciate the role of adaptive technology in the lives of people with various disabilities.

Computing, like all technologies, has a profound impact on any culture into which it is placed. The distribution of computing resources in a global economy raises issues of equity, access, and power. Social and economic values influence the design and development of computing innovations. Students should be prepared to evaluate the various positive and negative impacts of computers on society and to identify the extent to which issues of access (who has access, who does not, and who makes the decisions about access) impact our lives.

### 4.2.5 Computing Practice

The use of computational tools is an essential part of computer science education at all levels. While this is traditionally branded as "Information Technology," it is impossible to separate IT from the other four strands in computer science. Computing practice at the K–12 level must therefore include the ability to create and organize Web pages, explore the use of programming in solving problems, select appropriate file and database formats for a particular computational problem, and use appropriate Application Program Interfaces (APIs), software tools, and libraries to help solve algorithmic and computational problems.

K–12 students must also be introduced to the variety of careers that exist in computing, from IT Specialist to Systems Analyst, Programmer, CIO, Computer Engineer, Software Engineer, and so forth. By the time they reach high school and are selecting career or educational paths, students should be well informed about their options to make intelligent decisions.

## 5. Comprehensive Standards for K–12 Computer Science

Drawing from the understandings and contexts described in earlier sections, this section defines new standards for K–12 computer science education, presenting them in a learning objective-based format that identifies the specific computer science concepts and skills students should achieve at each of the three levels (grades K–6, 6–9, and 9–12).

### 5.1 Level 1: Computer Science and Me

These standards introduce elementary school students to foundational concepts in computer science by integrating basic skills in technology with basic concepts about computational thinking. The learning experiences created from these standards should be inspiring and engaging, helping students see computing as an important part of their world. They should be designed with a focus on active learning, creativity, and exploration and will typically be embedded within other curricular areas such as social science, language arts, mathematics, and science.

It is important to recognize the significant impact that an early exposure to these five strands can have as students progress toward higher level computer science programs. K-6 students need to learn how computing tools can be used to help solve problems, communicate with others, and access and organize information by themselves or in collaboration with others. They must also learn to be responsible citizens in the ever-changing digital world. Ethical and safe uses of computers and networks should be introduced to even our youngest students.

We agree with teachers who believe that students at this age ought to begin thinking algorithmically as a general problem-solving strategy. Thus, it makes sense to develop more teaching strategies that encourage students to engage in the process of visualizing or acting out an algorithm. Seymour Papert's pioneering experiments in the 1980s corroborate this belief, and his seminal work *Mindstorms* and related curricula (Papert, 1980) provide many more examples of how elementary students can be engaged in algorithmic thinking. This engagement can be accomplished with or without the use of computing devices as in the following examples:

- finding your way out of a maze (Turtle graphics, robotics),
- a dog retrieving a thrown ball,
- baking cookies,
- going home from school,
- making a sand castle, and
- arranging a list of words in alphabetical order.

Any of the activities designed to introduce K-6 students to the five strands can involve individuals working alone or in collaborating with their peers. The concept of "team" is one that can be introduced at any grade level. Students working together can find multiple solutions to problems utilizing the resources available through online searches and then create multimedia presentations as a way of demonstrating their chosen solution. As they are developing these skills, they can start to become more aware of the many tools and programs that can be used to communicate with each other, their teachers, their parents, and even students in far away places. They will begin to think about how computers and programs work together to make things happen as they do.

As K-6 students are exposed to the many facets of communications and social networking through technology, it is essential that they learn the safe and ethical way to use these tools and the potential repercussions of their improper use. They can explore the many ways in which computing devices and technology impact their lives and the society around them.

Following are the standards that all students in Grades K–6 should meet in the five strands:

**Computational Thinking:**
*Grades K–3*
The student will be able to:
- Use technology resources (e.g., puzzles, logical thinking programs) to solve age-appropriate problems.
- Use writing tools, digital cameras, and drawing tools to illustrate thoughts, ideas, and stories in a sequential manner.
- Understand how to arrange (sort) information into useful order, such as sorting students by birth date, without using a computer.
- Demonstrate how 0s and 1s can be used to represent information such as digital black and white images.
- Recognize that software is created to control computer operations in a sequential manner.

*Grades 3–6*
The student will be able to:
- Develop a simple understanding of an algorithm (e.g., search, long division, sequence of events, or sorting) using computer-free exercises.
- Understand and use the basic steps in algorithmic problem-solving (e.g., problem statement and exploration, examination of sample instances, design, implementation, and testing).
- Demonstrate how 0s and 1s can be used to represent information.
- Participate in a simulation to act out the solution to a local issue.
- Make a list of issues to consider while addressing a larger problem.
- Understand the connections between other fields and computer science.

**Collaboration**
*Grades K–3*
The student will be able to:
- Work cooperatively and collaboratively with peers, teachers, and others using technology.
- Gather information and communicate virtually with others with support from teachers, family members, or student partners.

*Grades 3–6*
The student will be able to:
- Use productivity technology tools (e.g., word processing, spreadsheet, presentation software) for individual and collaborative writing, communication, and publishing activities.
- Use online resources (e.g., email, online discussions, collaborative Web environments) to participate in collaborative problem-solving activities for the purpose of developing solutions or products.
- Identify ways that teamwork and collaboration can support problem solving and innovation.

**Computing Practice and Programming**
*Grades K–3*
The student will be able to:
- Identify careers (across a wide spectrum) that use computing and technology.
- Use technology resources to conduct age-appropriate research.
- Gather and organize information using concept-mapping tools.
- Create developmentally appropriate multimedia products with support from teachers, family members, or student partners.
- Use developmentally appropriate multimedia resources (e.g., interactive books and educational software) to support learning across the curriculum.
- Construct a set of statements to be acted out (e.g., turtle instructions).
- *Grades 3–6*

The student will be able to:
- Use general-purpose productivity tools and peripherals to support personal productivity, remediate skill deficits, and facilitate learning.
- Gather and organize information using a variety of digital concept mapping tools.
- 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.
- Use computing devices, including mobile devices, to access remote information, communicate with others in support of direct and independent learning, and pursue personal interests.

- Understand the organization of Internet elements and webpages (e.g., links, navigation, search engines and strategies, and evaluation).
- Identify different careers in computing and the interdisciplinary nature of computing in the 21st Century.
- Use technology resources (e.g., calculators, data collection probes, mobile devices, videos, educational software, and Web tools) for problem-solving and self-directed learning.
- Use technology resources to enable extended learning activities, including research, across the curriculum.
- Determine which technology is useful and select the appropriate tool(s) and technology resources to address a variety of tasks and problems.
- Construct a program as a set of statements to be acted out (peanut butter and jelly sandwich activity).
- Implement problem solutions in a block-based visual programming environment.

**Computers and Communication Devices**
*Grades K–3*
The student will be able to:
- Use standard input and output devices to successfully operate computers and related technologies.

*Grades 3–6*
The student will be able to:
- Demonstrate an appropriate level of proficiency with keyboards and other input and output devices.
- Explain the concept of computers as models of intelligent behavior (as found in robotics, speech and language recognition, fingerprint, facial, and retinal recognition; computer animation and virtualization).
- Identify factors that distinguish humans from machines.
- Understand the pervasiveness of computers and computing in daily life (e.g., voice mail, downloading videos and audio files, microwaves, thermostats, wireless Internet, mobile computing devices, GPS systems, etc.).
- Apply strategies for identifying and solving routine hardware and software problems that occur during everyday use.
- Identify that information is coming to the computer from many sources over a network.

**Community, Global, and Ethical Impacts**
*Grades K–3*
The student will be able to:
- Identify positive social and ethical behaviors for using technology.
- Practice responsible digital citizenship (legal and ethical behaviors) in the use of technology systems and software.

*Grades 3–6*
The student will be able to:
- Discuss basic issues related to responsible use of technology and information, and the consequences of inappropriate use.
- Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and biases that occur in electronic information sources.
- Understand ethical issues that relate to computers and networks (e.g., equity of access, security, privacy, copyright, and intellectual property).
- 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.

## 5.2 Level 2: Computer Science and Community

The learning expectations covered by these standards support middle school/junior high school students in the use of computational thinking as a problem-solving tool. They begin to appreciate the ubiquity of computing and the ways in which computer science facilitates communication and collaboration. Students begin to experience computational thinking as a means of addressing community-relevant issues. The learning experiences created from these standards should be relevant to the students and should promote their perceptions of themselves as proactive and empowered problem solvers within their community. They should be designed with a focus on active learning and exploration that can be taught either as an explicit computer science course or as units embedded in other curricular areas such as social science, language arts, mathematics, and science.

The Level 2 curriculum standards assume that students already have been introduced to the computational thinking concepts of data representation, algorithms, and problem solving; and that they have had experience using technology tools and resources for learning, creating digital artifacts, and for collaboration. Students should also have learned about: the many careers that use computing and technology, standard I/O devices and computers, basic computer terminology, and the principles of acting responsibly and ethically when using computers independently and or working with others.

Lower secondary school students are blooming socially and emotionally. They interact in larger social spheres than elementary students and are transitioning from being "self" oriented behaviors to group oriented behaviors. Their social spheres are growing beyond the nuclear family to include fellow students, peer groups, teachers, coaches and other community members. Acknowledging these shifts, the focus at Level 2 is on using computers and computation as both individuals and community members. In this way, students begin to experience computational thinking as a means of communicating with others and as a means to address community-relevant issues.



The goals of the Level 2 curriculum are to engage students in using computational thinking as a problem solving tool, teach them to use programming concepts and methods while creating digital artifacts, and retain their interest in computing as a relevant and exciting field. Learning opportunities should be presented in ways that are active, connected, and relevant to them, and should promote the perception of themselves as proactive and empowered problem solvers, creators, and innovators capable of changing the world. Collaborative learning experiences at this level should prepare students to work in teams and build supportive partnerships.

As students begin to master fundamental computer science concepts and practices, it is vitally important that they learn that these concepts and practices empower them to create innovations, tools and applications. Students should also know that with this knowledge and access comes responsibility, so issues of ethical and responsible use of computing and information are also essential elements of this curriculum.

Following are the standards that all students in Grades 6–9 should meet in the five strands:

**Computational Thinking:**
The student will be able to:
- Represent data in a variety of ways: text, sounds, pictures, numbers (e.g., binary, hexadecimal)
- Use visual representations of problem states, structures, and data (e.g., graphs, charts, network diagrams, flowcharts).
- Define an algorithm as a sequence of instructions that are processed by a computer.
- Evaluate ways that different algorithms may be used to solve the same problem.
- Act out searching and sorting algorithms.
- 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).
- Use the basic steps in algorithmic problem-solving to design algorithms (e.g., problem statement and exploration, examination of sample instances, design, implementing a solution, testing, evaluation).
- Interact with content-specific models and simulations (e.g., ecosystems, epidemics, molecular dynamics) to support learning and research.
- Evaluate what kinds of problems can be solved using modeling and simulation.

- Analyze the degree to which a computer model accurately represents the real world.
- Use abstraction to decompose a problem into sub problems.
- Understand the notion of hierarchy and abstraction in computing including high-level languages, translation, instruction set, and logic circuits.
- Examine connections between elements of mathematics and computer science including binary numbers, logic, sets and functions.
- Provide examples of interdisciplinary applications of computational thinking.
- Describe the process of parallelization as it relates to problem solving.

**Collaboration**
The student will be able to:
- Collaboratively design, develop, publish, and present products (e.g., videos, podcasts, wikis) using technology resources that demonstrate and communicate curriculum concepts.
- Apply productivity/multimedia tools and peripherals to group collaboration and support learning throughout the curriculum.
- Collaborate with peers, experts, and others using collaborative practices such as pair programming, working in project teams, and participating in group active learning activities (e.g., CS unplugged, KLAs).
- Exhibit dispositions necessary for collaboration: providing useful feedback, integrating feedback, understanding and accepting multiple perspectives, socialization.

**Computing Practice & Programming Careers**
The student will be able to:
- Use a variety of multimedia tools and peripherals to support personal productivity and learning throughout the curriculum.
- Select appropriate tools and technology resources to accomplish a variety of tasks and solve problems.
- Design, develop, publish, and present products (e.g., Web pages, mobile applications, animations) using technology resources that demonstrate and communicate curriculum concepts.
- Implement problem solutions in a programming environment using: looping behavior, conditional statements, logic, expressions, variables, and functions.
- Demonstrate an understanding of algorithms and their practical application.
- Demonstrate good practices in personal information security using passwords, encryption, and secure transactions.
- Demonstrate dispositions amenable to open-ended problem solving and programming (e.g., comfort with complexity, persistence, brainstorming, adaptability, patience, propensity to tinker, creativity, accepting challenge).
- Identify interdisciplinary careers that are enhanced by computer science.
- Collect and analyze data that is output from multiple runs of a computer program.



**Computers & Communication Devices**
The student will be able to:
- Use developmentally appropriate, accurate terminology when communicating about technology.
- Identify a variety of electronic devices that contain computational processors.
- Recognize that computers are devices that execute programs.
- Demonstrate an understanding of concepts, underlying hardware and software and their practical applications.
- Describe the major components and functions of computer systems and networks.
- Apply strategies for identifying and solving routine hardware problems that occur during everyday computer use.
- Recognize that computers use models of intelligent behavior (e.g., robot motion, speech and language understanding, and computer vision).
- Describe what distinguishes humans from machines with a focus on human intelligence versus machine intelligence and ways we can communicate.

**Community, Global, and Ethical Impacts**
The student will be able to:

- Exhibit legal and ethical behaviors when using information and technology and discuss the consequences of misuse.
- Analyze the positive and negative impacts of computing on human culture.
- Demonstrate knowledge of changes in information technologies over time and the effects those changes have on education, the workplace, and society.
- Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources concerning real-world problems.
- Describe ethical issues that relate to computers and networks (e.g., security, privacy, ownership, and information sharing).
- Discuss how the unequal distribution of computing resources in a global economy raises issues of equity, access, and power.

## 5.3 Level 3: Applying Concepts and Creating Real-World Solutions

Level 3 is divided into three discrete courses, each of which focuses on a different aspect of computer science as a discipline. Throughout these courses, students will learn advanced computer science concepts and apply those concepts to develop virtual and real-world artifacts. The learning experiences created from these standards should focus on the exploration of real-world problems, the application of computational thinking to the development of problem solutions, and the interconnections between computer science and other academic subjects. These experiences should also include a focus on collaborative learning and effective communication.

Level 3 includes the following courses:

**3A: Computer Science in the Modern World**
**3B: Computer Science Principles**
**3C: Topics in Computer Science**

Normally, course 3A will be a prerequisite for either of the other two. The following sections describe these three courses in more detail.

### *5.3.A Computer Science in the Modern World*

Computer Science in the Modern World is a course designed to expose all students to the inter-disciplinary nature of computer science in today's dynamic and globally connected society. Students will have the opportunity to explore the uses of computer science as a tool in creating effective solutions to complex contemporary problems. The hands-on nature of the course is intended to provide students with the opportunity to explore conceptual understanding in a practical learning environment. This course is recommended for all students as it provides an overview of computer sciences and its applications in various disciplines, professions, and personal activities.

In this course, students will learn to use computational thinking by developing algorithmic solutions to real-world problems. They will begin to understand the different levels of complexity in problem solving and to determine when team projects might generate more effective problem solutions than individual efforts. Students will learn and use a programming language(s) and related tools, as well as appropriate collaboration tools, computing devices, and network environments. Finally, they will demonstrate an understanding of the social and ethical implications of their work and exhibit appropriate communication behavior when working as a team member.

*Computer Science in the Modern World* is a course designed for all students at the 9th and 10th grade levels. This course is built around the essential skills that all high school students should have upon graduation. It also provides the necessary skills needed for more advanced studies at levels 3.B and 3.C. It is recommended that this course be required of all students.

**Computational Thinking**

The student will be able to:

- Describe how various types of data are stored in a computer system.
- Analyze the representation and trade-offs among various forms of digital information.
- Convert values between binary, octal, hexadecimal, and decimal representations.
- Explain how sequence, selection, iteration, and recursion are building blocks of algorithms.
- Describe software development methods used to solve software problems (e.g., design, coding, testing, verification).
- Use predefined functions and parameters, classes and methods to divide a complex problem into simpler parts.
- Use modeling and simulation to represent and understand natural phenomena.
- Describe the concept of synchronization as a strategy to solve large problems.
- Discuss the value of abstraction to manage problem complexity.
- Compare techniques for analyzing massive data collections.
- Describe how computation shares features with art and music by translating human intention into an artifact.

**Collaboration**

The student will be able to:

- Use collaborative tools to create software artifacts (e.g., version control) and to communicate with team members (e.g., discussion threads).
- Work in a team to design and develop a software artifact.
- Identify how collaboration influences the design and development of software products.
- Describe how computing enhances traditional forms and enables new forms of experience, expression, communication, and collaboration.

**Computing Practice and Programming**

The student will be able to:

- Use mobile devices/emulators to design, develop, and implement mobile computing applications.
- Create and organize Web pages through the use of a variety of design tools including the programming of client- and server-side scripts.
- Use Application Program Interfaces (APIs) and libraries to facilitate programming solutions.
- Apply modern analysis, design, and implementation techniques to solve problems.
- Use debugging and unit testing to verify programs.
- Explain the program execution process.
- Describe the variety of programming languages available to solve problems and develop systems.
- Select appropriate file formats for various types of data (e.g., sequential, random, relational).
- Explain the principles of security by examining encryptions, cryptography and authentication techniques).
- Explore a variety of careers in computing (e.g., IT specialist, Web designer, systems analyst, programmer, CIO, computer scientist, computer engineer).
- Describe how mathematical and statistical functions, sets, and logic are used in computation.
- Describe techniques for locating and collecting small and large scale data sets.

**Computers and Communication Devices**

The student will be able to:

- Develop criteria for purchasing and/or upgrading computer system hardware.
- Describe the unique features of computers embedded in mobile devices and vehicles (e.g., cell phones, automobiles, airplanes).
- Discuss fundamental principles of hardware design.
- Explain the multiple levels of hardware and software that support program execution (e.g., compilers, interpreters, operating systems, networks).
- Compare various forms of input and output media.
- Describe the principal components of computer organization (e.g., input, output, processing, and storage).
- Apply strategies for identifying and solving routine hardware and software problems that occur in everyday life.
- Describe how the Internet facilitates global communication.
- Describe how various types of data are stored in a computer system.

- Explain the basic components of computer networks (e.g., servers, file protection, routing, spoolers and queues, shared resources, and fault-tolerance).
- Compare and contrast client-server and peer-to-peer network strategies.
- Describe the major applications of artificial intelligence and robotics.

**Community, Global, and Ethical Impacts**
The student will be able to:

- Compare appropriate and inappropriate social networking behaviors.
- Compare the positive and negative impacts of technology on culture (e.g., social networking, delivery of news and other public media, and inter-cultural communication).
- Describe ethical issues that relate to computers and networks (e.g., security, privacy, IP, public domain software, information sharing, attribution, open source development, and creative commons licensing).
- Discuss the impact of computing technology on business and commerce (e.g., automated tracking of goods, automated financial transactions, e-commerce, cloud computing).
- Describe strategies for determining the reliability of information found on the Internet.
- Discuss the social and economic implications associated with hacking and software piracy.
- Describe how different kinds of software licenses can be used to share and protect intellectual property.
- Differentiate between information access and information distribution rights.
- Identify ethical issues that relate to computers and networks.
- Describe the role that adaptive technology can play on the lives of people with special needs.
- Explain the impact of the digital divide on access to critical information.

### *5.3.B Computer Science Principles*

*Computer Science Principles* is a follow-up course to Computer Science in the Modern World. It is designed to harness the interests of those students wishing to further enhance their studies in the computing fields. In this course, students will have the opportunity to develop higher-level computing skills and apply them to a variety of subjects and disciplines. Students will learn how computer science impacts society and promotes change. Through the analysis of global issues, students will explore how computer science can help solve real-world problems using innovation, collaboration, and creativity. This course will also provide students with an opportunity to explore Computer Science as a potential career interest at the collegiate level.

In both its content and pedagogy, this course aims to appeal to a broad audience. In this course, will begin to understand the central ideas of computing and computer science, focusing on the concepts and practices of computational and critical thinking and engaging in activities that show how computer science is helping to change the world. This rigorous course also engages students in the creative aspects of computer science.

*Computer Science Principles* is designed as an elective course geared towards students in grades 10 through 12. Students enrolled in this course should have previously completed *Computer Science in the Modern World*. Due to its mathematical content, students should also have completed at least Algebra I. This course should nevertheless be accessible to all students. Students with special interests in a concentrated area of computer science (such as networking, programming, game design, etc.) can continue their exploration through the courses outlined in Level 3.C: *Topics in Computer Science*.

**Computational Thinking**
The student will be able to:

- Discuss the interpretation of binary sequences in a variety of forms (e.g., instructions, numbers, text, sound, image).
- Compare and contrast simple data structures and their uses (e.g., arrays, lists, stacks).
- Evaluate algorithms by their efficiency, correctness, and clarity.
- Use data analysis to enhance understanding of complex natural and human systems.
- Explain the value of heuristic algorithms to approximate solutions for intractable problems.
- Critically examine classical algorithms and implement an original algorithm.
- Classify problems as tractable, intractable, or computationally unsolvable.
- Analyze data and identify patterns through modeling and simulation.
- Use models and simulations to help formulate, refine, and test scientific hypotheses.

- Demonstrate concurrency by separating processes into threads and dividing data into parallel streams.
- Decompose a problem by defining new functions and classes.

**Collaboration**
The student will be able to:
- Demonstrate the software life cycle and agile development process through participating on a software project team.
- Use project collaboration tools, version control systems, and IDEs while working on a collaborative software project.
- Evaluate programs written by others for readability and usability.

**Computing Practice and Programming**
The student will be able to:
- Use advanced tools to create digital artifacts (e.g., Web design, animation, video, multimedia).
- Use object-oriented principles in program design.
- Use functions and classes to decompose a large-scale computational problem.
- Explore principles of system design in scaling, efficiency, and security.
- Classify programming languages based on their level and application domain
- Deploy principles of security by implementing encryption and authentication strategies.
- Anticipate future careers and the technologies that will exist.
- Deploy various data collection techniques for different types of problems.
- Use data analysis to enhance understanding of complex natural and human systems.

**Computers and Communication Devices**
The student will be able to:
- Discuss the impact of modifications on the functionality of application programs.
- Identify and select the most appropriate data format based on trade-offs (e.g., accuracy, speed, ease of manipulation).
- Identify and describe hardware in abstract ways (e.g., physical layers, logic gates, chips, components).
- Describe the issues that impact network functionality (e.g. latency, bandwidth, firewalls, server capability).
- Explain the notion of intelligent behavior through computer modeling and robotics.

**Community, Global, and Ethical Impacts**
The student will be able to:
- Demonstrate ethical use of modern communication media and devices.
- Analyze the beneficial and harmful effects of computing innovations.
- Summarize how financial markets, transactions, and predictions have been transformed by automation.
- Summarize how computation has revolutionized the way people build real and virtual organizations and infrastructures.
- Differentiate among open source, freeware, and proprietary software licenses and their applicability to different types of software.
- Analyze the impact of government regulation on privacy and security.
- Identify laws and regulations that impact the development and use of software.
- Relate issues of equity, access, and power to the distribution of computing resources in a global society.

### *5.3.C Topics in Computer Science*

At this level, interested and qualified students should be able to select one from among several electives to gain depth of understanding or special skills in particular areas of computer science. All of these electives will require the Level 3A course as a prerequisite, while some may require the Level 3B course as well. Most important, these courses provide students with an opportunity to explore topics of personal interest in greater depth, and thus prepare for the workplace or for further study at the post-secondary level.

These electives include, but are not necessarily limited to:

- Advanced Placement (AP) Computer Science A,
- A projects-based course in which students cover a topic in depth,
- A vendor-supplied course, which may be related to professional certification.

These alternatives are discussed in more detail below.

#### *5.3.C.1 AP Computer Science A*

The Advanced Placement Computer Science curriculum is well established (AP, 2010), and is offered at many secondary schools for students planning to continue their education in a two- or four-year college or university, possibly in computer science, business, or a related field. The AP Computer Science A course emphasizes problem solving and algorithm development, and introduces elementary data structures. Students who complete this course and score well on the exam may qualify for one-semester of college credit. Students taking the AP Computer Science A course should have completed Level 1, Level 2, and Level 3A. That is, they need to be familiar with the computational/algorithmic concepts introduced at those levels. The Level 3B course provides an excellent foundation in computer science principles and may also be useful for students intending to take the AP Computer Science A course.

#### *5.3.C.2 Projects-Based Courses*



A projects-based course would be available to all students who have completed the Level 1 and Level 2 courses. Most project-based courses will also require completion of the Level 3A course. Some variants of this course would also require completion of the Level 3B course. A project-based course can be either a half-year or a full-year course.

The projects in this kind of course will naturally reflect diverse student interests and specific faculty expertise. The specific projects that are chosen from year to year will also evolve to reflect the ever-changing characteristics of computer science and information technology. Ideally, each project should build upon basic computer science concepts and help students develop professional skills in the application of technology. Schools should also consider offering project-based courses in conjunction with a local college or university to ensure currency and tap outside expertise.

While some of the project-based courses may be more skills-based, they must still be tied to the "behind-the-scenes" activities of the software and other computer science principles in general. Making such connections enables student to problem solve when software does not perform as anticipated.

Here are some project-based courses that could meet the requirements of a Level 3C course. See the Appendix for more details.

***Example: Desktop Publishing***: This course introduces planning, page layout, and the use of templates to create flyers, documents, brochures, and newsletters. Word processing and graphical editing fluency (Level 2 will help ensure student success. Methods of distribution of these documents in both written and electronic formats should be included. This will necessitate understanding of Internet concepts and network connectivity (Level 3A).

***Example: Technical Communications***: The ability to communicate and share ideas should be a core requirement for all high school graduates. This type of project focuses on end-user documentation and researching and presenting technical information to non-technical individuals in oral, written, and multimedia form. Fluency with word processing and presentation software and an understanding of computer science and technology (Level 3A) is required.

***Example: Multimedia***: The use of multimedia has increased steadily at the user level, fueled by more efficient hardware and the availability of digital cameras and digital audio equipment. However, multimedia is often abused when incorporated into programs, Web pages, and presentations. This project will provide instruction in the use of digital audio and video equipment and related editing software. A major focus will be deploying multimedia in a responsible fashion. Basic software skills (Level 2) and an understanding of multimedia concepts (Level 3A) are required.

***Example: Graphics*****:** This class explores bitmap and vector-based graphics. The discussion includes benefits and limitations of each type of software and hands-on experience with both. CAD, CAM, and 3-D design software should be explored as well as bitmap software for creating and editing of graphics. Availability of a digital camera and scanner is required. Responsible deployment of graphics including style and legal issues needs to be investigated. The discussion of vector-based graphics will be facilitated by completion of Level 3A—limits of computers and design for usability.

***Example: Game Programming*****:** This course helps students understand the creativity needed to program effectively and reinforces the software development cycle. Students plan, design, code and test computer games. Basic programming skills and an understanding of media (level 3B) are required.

***Example: Computational Modeling*****:**
This course explores the computational modeling of complex systems. Using agent-based techniques, locally relevant issues such as the spread of disease, ecosystems, and traffic patterns can be modeled and investigated and efforts to ameliorate negative impacts can be designed and tested virtually. In this course students will come to understand how interactions between individual elements (for example, people, animals, or cars) and individuals and their environment can give rise to emergent, often unpredictable, patterns. Student project work includes abstracting a real-world issue or scenario, implementing a computational model by specifying the agents, interactions and environment in the model, and using automation to perform multiple runs of the simulation as an experimental testbed. Analysis of the model to itself and the data it produces determine if and how the simulated world relates to the real world.

***Example: Web Development:*** At several places in the curriculum students are exposed to Internet concepts and HTML. This course includes CSS and presents a more in-depth view of the design and development issues that need to be considered for a multi-platform international implementation. The standardization of Web page development using the recommendations of the WWW Consortium is one focus issue. Web page development will include coding HTML and CSS using a text editor and utilizing simple scripts to enhance Web pages.

***Example: Web Programming:*** Students who have successfully completed Level 3A and Level 3B but do not wish to take an AP course might nevertheless enjoy applying their programming skills to the WWW. To be successful, a solid understanding of Internet concepts, Web development issues, and basic programming concepts will be required. Topics in this course can include client-side and server-side scripting languages. Students will need to write scripts and deploy them within Web pages or on the Web server.

***Example: Emerging Technologies:*** This project can include several distinct topics, and its content is expected to change on a regular basis. Curriculum and materials for this topic would need to be developed from current resources on the Web, perhaps in conjunction with local colleges and universities, and with input from the professional sector of the Business Community.

***Example: Free and Open Source Software (FOSS) Development*****:** Students who have successfully completed Level 3A may enroll in a course where they can contribute to an ongoing FOSS software project. Here, they might read code written by others, contribute suggestions for new features, identify bugs, write user documentation, and learn to use modern collaborative technologies. Students would actively participate in project discussion threads. Examples of FOSS projects that are accessible to students are identified at http://hfoss.org.

A sample of some other topics (along with their prerequisites) includes:

- The computer and animation (Level 3A)
- Networking technologies (Level 3A)
- Programming simulations (e.g., a computer-controlled chemistry experiment) (Level 3A, Level 3B)
- Object-oriented design and coding (Level 3B)
- Effective use of computer applications (Level 1, Level 2)

#### *5.3.C.3 Courses Leading to Industry Certification*

Such a course is primarily geared toward students planning on entering the workforce, continuing their education in a post-secondary technical school, or entering a two-year college AAS program. Students taking this course should have completed the Level 1 and Level 2, and typically the Level 3A courses.

Industry certification provides a standard that is useful to potential employers in evaluating a candidate who has no prior work experience. Industry certifications are either vendor-neutral or vendor-sponsored. Vendor-sponsored curricula need to be evaluated carefully. While rich in content, some of these courses are structured to emphasize proprietary products rather than general concepts. Students who complete certification courses should be encouraged to take the corresponding exam as proof of acquired knowledge. Here are a few examples of vendor-neutral certification programs.

***Example: A+ Certified Technician*: “**The CompTIA A+ certification is the industry standard for computer support technicians. The international, vendor-neutral certification proves competence in areas such as installation, preventative maintenance, networking, security and troubleshooting. CompTIA A+ certified technicians also have excellent customer service and communication skills to work with clients (http://www.comptia.org/certifications/listed/a.aspx). Two different exams are available: CompTIA A+ Essentials and CompTIA A+ Practical Application. The use of critical thinking skills to problem-solve is necessary to troubleshoot and resolve problems. These skills reinforce and extend the concepts presented in Levels 1, 2, and 3A.

***Example: Certified Internet Webmaster (CIW)*:** CIW’s core curriculum focuses on the foundational standards of the Web, including Web design, Web development and Web security. CIW certifications verify that certified individuals have the skills necessary to master a technology-driven world (http://ciwcertified.com/About_CIW/index.php). CIW curriculum is state-endorsed in various areas of the country. (http://ciwcertified.com/About_CIW/Why_CIW/highschools.php) The CIW Web Foundations exam requires competency in Internet business, Web design, and networking fundamentals. Many of these concepts are introduced in Levels 1, 2, and 3A.

More detailed information about these and other certification programs, both vendor-specific and vendor-neutral, can be found by searching the Web.



## 6. Implementation Challenges

We understand that many obstacles lie in the way of the ideal of a K–12 computer science education for all students. How will room be found in the jam-packed curriculum? How will qualified teachers be recruited, trained, and credentialed? In the world of standards-centric evaluation of schools, should computer science support existing standards, or should new ones be designed for computer science? These and other questions and challenges are significant, but so are the benefits—to students and to society—of computer science becoming as much a part of a high-quality education as other core disciplines.

Teaching any subject effectively depends on the existence of a sound learning standards for students, explicit teacher certification standards, appropriate teacher training programs, and effective curricular materials and a core of teachers who are willing, able, and empowered to deliver the curriculum. K–12 computer science education faces unique challenges along all of these lines.

The challenge of improving computer science education is significant and will require attention and interventions from multiple institutions. Professional organizations in computer science can make an important contribution. CSTA, for example, is a professional organization that supports and promotes the teaching of computer science and other computing disciplines. CSTA provides a large number of programs that include the development and dissemination of learning resources, the provision of professional development, and advocating for state and federal level policies to improve computer science education. Other organizations such as ACM, the IEEE Computer Society, institutions of higher education, and national and local teacher organizations can also work to addressing these issues in K–12 computer science education. Industry is also deeply affected by pipeline issues and the need to produce workers who have the skills to support and build the technology tools of the future. It is therefore in their best interest to contribute significantly to improving access to the quality of computer science courses at the K–12 level.

For schools to widely implement these standards, work is needed in three important areas: *teacher preparation*, *state-level content standards, and curriculum materials development*. In addition, persons in leadership positions must acknowledge the importance of computer science education for the future of our society. States and accrediting organizations should make this a factor in their overall school accreditation process. Some states have begun to establish computer science content standards, define models for teacher certification, provide in-service training in computer science, and experiment with developing new curricular materials. However, a much wider effort and commitment are now required. The following sections discuss these needs in more detail.

## 6.1 Teacher Preparation

Two levels of teacher preparation must be considered for this curriculum to be successfully implemented. Since the Level 1 curriculum will be integrated into existing curricula, all individuals striving for teacher certification should be required to demonstrate a basic understanding of computer science, the related computing tools, and the related computing tools, and methods for presenting and integrating the content. Individuals who will teach the computer science curriculum at higher levels must acquire both a depth mastery of computer science and the pedagogical skills that will allow them to present the material to students at appropriate levels.

Beyond the mastery of core subject matter, teachers need to be able to convey this material to others accurately and reliably, to provide perspective, and to cultivate the student's interest and curiosity. In other words, they need to possess three distinct kinds of knowledge:

- subject matter knowledge,
- pedagogical content knowledge, and
- curricular knowledge.

This requires a rigorous, well-established teacher preparation program, defined by certification requirements and supported by professional development opportunities. Ensuring that we have a sufficiently large pool of teachers prepared to teach computer science in K–12 is particularly challenging, however, because the path to becoming a computer science teacher differs significantly from that for teachers in many other disciplines.

One of the key challenges to improving computer science teacher preparation and certification centers on the current lack of consistency between educational and licensure bodies in different states. The terms used to describe various levels of teacher licensure can be confusing and different licensure bodies often use the same terms in slightly different ways. In some states, for example, teachers must become certified to teach at a specific educational level (for example elementary or secondary). They then must meet additional qualifications to receive an endorsement to teach a specific subject area. In other states, the original teacher certification can include both their educational level and their subject area (Khoury 2007). Whatever the state logistics are, the necessary elements for quality teacher preparedness are the same: academic requirements in the fields of computer science and education, and a comprehensive understanding of methodology and pedagogy in the teaching of computer science.

In most instances there is no specific computer science certification, so computer science teachers must first meet the certification requirements in some other discipline. In some cases, teachers may be able to receive an additional endorsement to teach computer science even though the content they are required to master may have no more than a tangential relationship to what is needed to teach in a computer science classroom. A survey conducted by the Computer Science Teachers Association (Khoury, 2007) indicated that approximately 53% of the states required an endorsement to teach computer science at some grade level (K–12) but the courses that the endorsed teachers could teach vary from keyboarding and computer applications to programming. Finally, teachers may find it very difficult to prepare to become computer science teachers because so few schools of education provide programs with rigorous and relevant computer science training. In the absence of clear and specific requirements for computer science, these schools, whose primary mission is to prepare teachers to meet discipline-based requirements for certification, have little or no incentive to address the needs of computer science teachers.

In the present educational infrastructure, the majority of K–12 computer science teachers come from one of the following constituencies:

- new teachers: presently college or university students working towards their first teacher certification,
- veteran teachers with a certification in another area that have never taught computer science,
- veteran teachers with a certification in another area that have experience teaching computer science, and
- professionals coming from business with a computer science background and no teaching experience.

To meet the learning needs of this diverse group of teacher candidates and current practicing teachers, CSTA has developed a multi-level model that includes distinct requirement for each of these four sets of teacher constituencies. The document *Ensuring Exemplary Teaching in an Essential Discipline* (http://csta.acm.org/ComputerScienceTeacherCertification/sub/CertificationResearch.html) describes this model in detail. The following chart lists the requirements for each of the four types of potential computer science teacher.

| **Certification for** | **Eligibility Requirements** | **License/Endorsement Requirements** |
|---|---|---|
| **New Teachers** | **Degree (or presently working for degree)**<br><br>Bachelor's degree or higher in computer science **or** a minor in computer science | **Academic Work and Field Experience**<br><br>Academic requirements in the field of computer science:<br>* Major or minor in computer science<br>* A seminar-type course that includes the history of computer science, the nature of the field and its relationship with other disciplines, the various computer science curricula on both high school and college/university levels.<br>* Writing a research paper in the field of computer science education<br><br>Academic requirements in the field of education:<br>* Curriculum design and development<br>* Educational Psychology<br>* Technology in the classroom<br><br>Methodology and field experience:<br>* Methods Course<br>* Class observations and a minimum of 10 weeks practice teaching<br><br>Praxis Exam:<br>* Praxis II: Principles of Learning and Teaching Exam or equivalent satisfactory performance on a similar assessment to document proficiency in general pedagogy |
| **Veteran Teachers with NO Computer Science Teaching Experience** | **Degree**<br>Bachelor's degree or higher in a field other than computer science<br><br>**Certification**<br>Certification in an academic discipline other than computer science | **Academic Work and Field Experience --- (to be completed within 3 years)**<br>Academic requirements in the field of computer science include advanced coursework in the following areas:<br>* programming<br>* object-oriented design<br>* data structures and algorithms<br>* computer hardware and organization<br>* computer applications<br><br>Methodology requirements can be documented by the completion of at least **one of the following:**<br>* Methods Course<br>* Auditing of two complete K–12 computer science |

| | | |
|---|---|---|
| | | courses |
| **Veteran Teachers WITH Computer Science Teaching Experience** | **Degree**<br>Bachelor's degree or higher in a field other than computer science<br><br>**Certification**<br>Certification in an academic discipline other than computer science<br><br>**Computer Science Teaching Experience**<br>* Teaching an Advanced Placement Computer Science course (or the equivalent) for at least two years, **and/or**<br>* Teaching International Baccalaureate HL Computer Science (or the equivalent) for) for at least two years, **and/or**<br>* Teaching a rigorous introductory computer science course (equivalent to the Level 3 course described in the CSTA Computer Science Standards) for at least two years | **Academic Work and Field Experience --- to be completed within 3 years**<br><br>Academic requirements in the field of computer science can be documented by completing **one of the following:**<br>* Completion of a minimum of 40 hours of professional development workshops designed for teachers of computer science.<br>Advanced coursework in the following areas:<br>* programming<br>* object-oriented design<br>* data structures and algorithms<br>* computer hardware and organization<br>* computer applications<br><br>Methodology requirements can be documented by the completion of at least **one of the following:**<br>* Completion of a minimum of 40 hours of professional development workshops designed for teachers of computer science.<br>* Methods Course<br>* Auditing of one complete K–12 computer science course<br>*Creating a portfolio that documents pedagogy in the computer science classroom |
| **Individuals Coming From Business With A Computer Science Background** | **Degree**<br>Bachelor's Degree or higher in Computer Science<br><br>Bachelor's Degree or higher in Related Field, for example:<br>* Bioinformatics<br>* Computer Information Systems<br>* Computer Programming<br>* Computer Systems Engineering<br>* Database Systems<br>* Electrical Engineering<br>* Information Science<br>* Information Systems Design<br>* Information Technology<br>* Mathematics<br>* Networking<br>* Robotics<br>* Software Engineering<br><br>Undergraduate minimum 2.5 GPA on a scale of 4.0<br><br>Undergraduate minimum 3.0 GPA in major field on a scale of 4.0<br><br>**Work Experience**<br>A minimum of two years related work experience within past five years. | **Academic Work (a total of 18 college/university credits) and field experience --- to be completed within 3 years**<br><br>Academic requirements in the field of computer science include advanced coursework in the following areas (minimum of 3 credits):<br>* programming<br>* object-oriented design<br>* data structures and algorithms<br>* computer hardware and organization<br>* computer applications<br><br>Academic requirements in the field of education (minimum of 9 credits):<br>* Curriculum design and development<br>* Educational Psychology<br>* Technology in the classroom<br><br>Methodology and field experience:<br>Methods Course (3 credits)<br><br>Class observations and a minimum of 10 weeks practice teaching<br><br>Praxis Exam:<br>Praxis II: Principles of Learning and Teaching Exam or equivalent satisfactory performance on a similar assessment to document proficiency in general pedagogy |

The development of teaching certification requirements and content standards for K–12 computer science education by the various states should, in turn, prompt districts and schools to implement relevant computer science programs. But most importantly, this step should also motivate schools of education to introduce pre-service programs in computer science education. With computer science becoming a recognized academic discipline in K–12 schools of education will become more motivated to set up such pre-service programs.

Some states already offer certification or an endorsement for teaching computer science. But the majority of states do not require any computer science credentials for teaching this subject. For those states that offer teachers an endorsement in computer science, their requirements vary widely; some require teachers to have a background in data processing while others require them to have a business background. Another concern is that some states' endorsements cover a very narrow aspect of computer science while others bury the subject inside and endorsement in technology education.

It is essential that the states' departments of education review their licensing standards for professional educators so that they recognize and support computer science as a distinct discipline. The meaning of the term "teacher of computer science" requires clear definition. As the requirements for such a certification and its pre-service preparation programs are developed, states should also maintain the view that the field of computer science is evolving rapidly. (For a comprehensive exploration of issues relating to computer science teacher certification see *Ensuring Exemplary Teaching in an Essential Discipline: Addressing the Crisis in Computer Science Teacher Certification* at http://csta.acm.org/ComputerScienceTeacherCertification/sub/TeacherCertificationRequi.html.)

As with other subjects, in-service education is important to help current teachers adopt and integrate new computer science curriculum elements and to keep their computer science knowledge and skills up-to-date. In-service programs therefore deliver continuing professional development for the educators who will teach computer science courses. In-service program must provide relevant on-going learning opportunities for teachers already in the school systems, so that they maintain currency with computer science practices and acquire the skills and knowledge necessary to obtain new certifications as needed.

An in-service program can take many forms. In addition to school-wide and district-wide workshops, state and regional events can be organized to bring teachers together as a community to learn and exchange ideas. These events can be used to disseminate to the teachers and school administrators new curricular content and guidelines as they evolve. College and university computer science and education faculty can also play an important role through the provision of intensive multi-day workshops focused on key curriculum elements and new teaching methodologies.

Professional recognition is also essential for the current cohort of teachers of computer science, regardless of the nature of their original teaching certification. Almost all of these teachers have original credentials in mathematics, science, business, or English, but they have since self-educated to teach many different types of computing courses, including AP computer science. One way to provide such recognition is for states to develop standard core competencies for computer science teachers and endorse those teachers who already have these competencies, thus recognizing teachers who have achieved the requisite skills and knowledge in computer science by their own efforts. Thus, major new in-service training initiatives for computer science are urgently needed in all 50 states.

### 6.2 State-Level Content Standards

Recently, efforts have increased to develop national and state content standards for computer science. Curriculum standards serve to define the skills and knowledge of the discipline to be acquired by every student. Content standards for computer science education within states must be developed and adopted in a way that parallels what has occurred in disciplines such as science, mathematics, and language arts. Curriculum content that is aligned with these standards can then be developed for the classroom.

In the design of state standards, it is important to ensure the distinction between the teaching of IT skills and the teaching of computer science itself. That is, computer science must be viewed as a distinct subject area and technology should be viewed as a tool that cuts across all subject areas. Existing technology standards, where present, should not be substituted for computer science standards. (For a comprehensive discussion on the ways in which states have failed to incorporate computer science standards into state standards in this way, see *Running on*

*Empty: The Failure to Teach* K–12 *Computer Science in the Digital Age* at http://csta.acm.org/Communications/sub/Documents.html.)

## 7. Conclusions

Computer science is a mainstream discipline that can no longer be ignored by public schools in the 21st century. The learning standards detailed in this document provide a basis by which states, schools of education, and individual school districts can begin to implement a coherent computer science curriculum that is available to all students.

Much work needs to be done to translate these standards into teaching and laboratory materials that are pedagogically viable and widely accessible. We hope that alongside state and federal departments of education, corporations, foundations, and other external stakeholders will support this work by providing appropriate incentives that will enable such a massive curriculum development effort to succeed.

**References**


*2008–09 Taulbee Survey*. (2009).Downloaded February 2011 from http://www.cra.org/resources/taulbee/

*AP Course Description: Computer Science*. (2011) Downloaded February 2011 from http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/4483.html

*Computer Science Unplugged*. (2011). Downloaded February 2011 from http://csunplugged.org/

Deek, F. and H. Kimmel. (1999). Status of Computer Science Education in Secondary Schools. *Computer Science Education* 9 (2), 89–113, August 1999.

Ericson, B, Armoni, M.. Gal-Ezer, J., Seehorn, D., Stephenson, C., Trees, F. (2008) *Ensuring Exemplary Teaching in an Essential Discipline: Addressing the Crisis in Computer Science Teacher Certification.* New York: Association for Computing Machinery.

Friedman, T.L. (2007). The World is Flat 3.0: A Brief History of the Twenty-first Century. Picador/Farrar Straus, and Giroux. New York, NY.

Gal-Ezer, J. and D. Harel. (1999). Curriculum for a high school computer science curriculum. *Computer Science Education 9*(2), August 1999.

Khoury, G. (2007). *Computer Science State Certification Requirements*. CSTA Certification Committee Report. Retrieved February 2011 from http://csta.acm.org/ComputerScienceTeacher

Lapidot, T. and Hazzan, O. (2003). Methods of Teaching a Computer Science Course for Prospective Teachers. *Inroads—The SIGCSE Bulletin*, *35*(4), 29–34.

National Research Council Committee on Information Technology Literacy, *Being Fluent with Information Technology*, National Academy Press, Washington, DC, May 1999.

National Council for Accreditation of Teacher Education. *Program for Initial Preparation of Teachers of: Educational Computing and Technological Literacy, and Secondary Computer Science Education*. http://www.ncate.org/standard/programstds.htm

Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas* (1980).

Simard, C., Stephenson, C., Kosaraju, D. (2010). *Addressing Core Equity Issues in K-2 Computer Science Education: Identifying Barrier and Sharing Strategies*. The Anita Borg Institute and the Computer Science Teachers Association. Palo Alto, CA.

State of Maine Learning Results. Downloaded February 2011 from http://www.state.me.us/education/lres/lres.htm

Task Force of the Pre-College Committee of the Education Board of the ACM. ACM Model High School Computer Science Curriculum. *Communications of the ACM*, May 1993.

Tucker, A., McCowan, D., Deek, F., Stephenson, C., Jones, J., & Verno, A. (2003, 2006). *A Model Curriculum for K–12 Computer Science: Report of the ACM K–12 Task Force Computer Science Curriculum Committee*, New York, NY: Association for Computing Machinery.

Wilson, C., Stephenson, C., Stehlik, M., Sudol, L. (2010). *Running on Empty: The Failure to Teach Computer Science in the Digital Age*. Association for Computing Machinery and the Computer Science Teachers Association, New York, NY.

## Acknowledgments

**Appendices**

In these appendices, we illustrate the viability of this curriculum model by providing example activities for courses and modules that are now being taught in various schools throughout the world. Many of these are directly adapted from the Ontario Computer Science Curriculum discussed in this report. The layout of these activities (also adapted from the Ontario curriculum) is explained below:

*Activity*: Name of the activity
*Time*: Number of in-class hours to complete the activity
*Description*: Brief description of the subject and goals of the activity.

*Level*: 1, 2, or 3, as defined in this report
*Topics:* The topics at this level that are covered by this activity (see the topic lists in Sections 5.1, 5.2, and 5.3 of this report)
*Prior Knowledge*: What students should know before beginning this activity.

*Planning Notes*: Suggestions to teachers for preparing this activity
*Teaching/Learning Strategies*: Organization of the in-class presentation and the particular student tasks
*Assessment and Evaluation*: Formative and summative assessments of in-class and laboratory work

*Accommodations*: Additional supporting materials (e.g., scaffolding labs, example programs, challenging problems)
*Resources*: Links to the source of this activity, as well as other related activities

## A.1 Sample Activities for Level 1

***Activity*: A Mystery Play**

*Time*: 8 hours

*Description*: Students learn and act out a short mystery play. Other students use logic to solve the mystery.

*Level*: 1

*Topics:* Understand the fundamental ideas of logic and its use for solving real-world problems

*Prior Knowledge*: Elements of logic (statements, truth and falsity), reading and speaking skills.

*Planning Notes*:

- Students will need time to learn their parts and rehearse the play in advance.

*Teaching/Learning Strategies*:

- The play is about 20 minutes long.
- The setting is a classroom, so no special props or scenery are needed.

*Assessment and Evaluation*:

- Students discuss the mystery and the reasoning they used to solve the mystery.

*Accommodations*:

Students receive copies of the play, which is 3 scenes and 4 pages long.

*Resources*:

See http://www.c3.lanl.gov/mega-math/menu.html to learn more about this activity.



***Activity*: Battleships**

*Time*: 4 hours

*Description*: Computers are often required to find information in large collections of data. They need to develop quick and efficient ways of doing this. This activity demonstrates three different search methods—linear search, binary search, and hashing—using numbered cards and the game of battleships as vehicles.

*Level*: 1 (Grades 3–5)

*Topics:* Basic understanding of a search algorithm

*Prior Knowledge*: Mathematics; greater, less, and equal relationships, geometry (coordinates)

*Planning Notes*: Finding information efficiently—linear search, binary search, hashing

*Teaching/Learning Strategies*:

- 15 children have cards with different numbers on them, arranged randomly and hidden from one of the children who tries to guess who holds a mystery number. The game is repeated after the 15 numbers are rearranged in order.
- Children are grouped in pairs, and each pair is given two battleship game cards. The game is played using a simple hashing technique to locate the column of a ship on the card.

*Assessment and Evaluation*:

- Discussions should explore the scores children achieved in each game.
- Discussions should also explore the advantages and disadvantages of each search strategy.

*Accommodations*:

Each child will need a battleships game card (copied from masters)

*Resources*:

See **http://csunplugged.org/** to learn more about this activity.

***Activity*: Beat the Clock**

*Time*: 4 hours

*Description*: There is a limit to how fast computers can solve problems. One way to speed them up is to use several computers to solve different parts of a problem. This activity uses sorting networks to do several comparisons at the same time.

*Level*: 1 (Grades K–2)

*Topics:* Understanding how to arrange (sort) information into useful order

*Prior Knowledge*: Grade 2 mathematics; greater than, less than

*Planning Notes*:

Some tasks can be done faster using fewer steps, while others can be done faster using parallel computation. Sorting networks is a good example of the latter.

*Teaching/Learning Strategies*:

- Six students hold one number each and arrange themselves on the left-hand side of the court. They move forward to the next circle in the network and wait for someone else to arrive.
- The circle is a decision point from which the student with the smaller number goes left and the larger number goes right.
- At the end, the numbers are sorted.

*Assessment and Evaluation*:

- Students successfully sort the numbers using the given network.
- They also discuss the use of other networks for sorting.

*Accommodations*:

This is an outdoor group activity. Chalk, two sets of six cards with numbers on them, and a stop watch are needed. A master is used to draw a sorting network on the sidewalk.

*Resources*:

See **http://csunplugged.org/** to learn more about this activity.

***Activity*: Color by Numbers**

*Time*: 3 hours

*Description*: The computer stores drawings, photographs, text, and other pictures using only numbers. This activity demonstrates how that is done.

*Level*: 1 (Grades K–2)

*Topics:* Using 0s and 1s to represent information

*Prior Knowledge*: Grade 2 geometry (exploring shapes, counting, graphing)

*Planning Notes*:

- Motivational discussion questions include, “What does a fax machine do?”
- “In what situations would a computer want to store pictures?”
- “How do computers store pictures when they can only use numbers?”

*Teaching/Learning Strategies*:

- A 5x6 rectangular grid is used as a basis for representing different images (such as letters) by coloring in some of the squares (pixels).
- Coding of the image is done by scanning the sequences of 1s (shaded squares) and 0s (non-shaded squares) in each row of the grid and recording the length of each sequence.

*Assessment and Evaluation*:

Worksheet activities.

*Accommodations*:

No computers are required; students use two worksheet activities, called “Kid fax” and “Make your own picture”

*Resources*:

See **http://csunplugged.org/** to learn more about this activity.

***Activity:* Dancing Cat**
*Time*: 60 minutes
*Description*:  Students construct a Scratch program to make a sprite dance.
*Level*: 1
*Topics*:

- Recognize that software is created to control computer operations in a sequential manner.

*Prior Knowledge*: Students should have a basic familiarity interacting with a computer and its input devices.
*Planning Notes*:
In preparing for this activity, teachers should:

- Decide how he/she would like the cat sprite to "dance". A sample dancing cat program is explained in the Getting Started Guide: http://scratched.media.mit.edu/resources/getting-started-guide
- Decide what concepts he/she would like to emphasize in his/her modeling of the activity. For example, if students are fairly new to programming and Scratch, the teacher can present how to drag and click blocks to make the cat sprite perform actions like moving 10 steps.

**Teaching/Learning Strategies:**
This activity involves the following steps:

- The teacher models the Dancing Cat activity for students. For example, the teacher can:
    - Present how Scratch looks like when it first opens, with the Blocks Palette on the left, the Scripts Area, and the Stage with the cat sprite.
    - Drag a *Move* block from the Blocks Palette to the Scripts Area.
    - Click on the *Move* block in the Scripts Area. Point out how the cat sprite moves 10 steps on the Stage.
    - Drag a *Play Drum* block from the Sound category and connect it to the *Move* block to create a stack of blocks or a script. Point out the white highlight that appears as you connect the two blocks. When a block is being dragged in the Scripts Area, the white highlight indicates where that block can make a valid connection.
    - Click anywhere on the stack to run the script. Point out how the cat sprite moves 10 steps and then plays a drum sound. Scratch runs a script starting from the top of a stack.
    - Build the rest of the dance script with the *Move* and *Play Drum* blocks. To mimic the back and forth movement of dancing, enter *-10* for the second *Move* block. After adding each new block, click on the stack and point out the new actions that the cat sprite performs on the stage.
    - Drag a *Forever* block from the Control category and connect it to the top of the stack, enclosing all the blocks.
    - Click on the stack again and point out how the sequence of actions repeat over and over again.
    - Drag the *When Green Flag Clicked* block from the Control category and connect it to the top of the stack.
    - Click on the Green Flag above the Stage. Point out how clicking on the Green Flag now runs the script.
- Students build their own Dancing Cat projects. The teacher should encourage them to experiment, finding new ways to express dancing or extending their program such as creating their own sprite.
- If there is time, students walk around and see each other's work. Students may also do a show and tell, explaining their projects to their peers.
- After students complete their projects, the teacher facilitates a discussion around some of the ways students programmed their sprite to dance.

*Assessment and Evaluation*:

- If there is time for students to walk around or do a show and tell, students can describe their projects and their process in creating their scripts. They can also evaluate the scripts of their peers.
- In the closing discussion, students can articulate what they learned in creating their Scratch program.

*Accommodations*:
While this activity can be carried out without any supplemental materials, the teacher may want to have the Getting Started Guide available for students to support them in the activity.
*Resources*:

- Getting Started Guide: http://scratched.media.mit.edu/resources/getting-started-guide
- ScratchEd: http://scratched.media.mit.edu/

***Activity*: Ice Cream Stand Problem**

| | |
|---|---|
| *Time*: | 4 hours |
| *Description*: | A graph is used to represent the map of a city. An ice cream company wants to build ice cream stands at different intersections, so that it is easy for people to get to them but not too many stands have to be built. |
| *Level*: | 1 (Grades 6–8) |
| *Topics:* | Understand the graph as a tool for representing problem states and solutions |
| *Prior Knowledge*: | Elementary map reading |

*Planning Notes*:

- Pass out copies of a map of a town, where lines represent streets and circles represent intersections.

*Teaching/Learning Strategies*:

- Children must determine the smallest number of stands to build so that no person has to walk to more than one intersection to buy ice cream.

*Assessment and Evaluation*:

- Children discuss different strategies for placing the stands, and they evaluate each other's solutions.

*Accommodations*:
Copies of different city maps need to be handed out.
*Resources*:
See http://www.c3.lanl.gov/mega-math/menu.html to learn more about this activity.

***Activity*: The Orange Game**

| | |
|---|---|
| *Time*: | 4 hours |
| *Description*: | This activity uses a simple game with oranges to illustrate Internet traffic management (routing) and deadlocks. |
| *Level*: | 1 (Grades 3–5) |
| *Topics:* | Simple algorithms for network routing |
| *Prior Knowledge*: | Math; logic and reasoning |
| *Planning Notes*: | This is a group activity, requiring five or more children sitting in a circle and having different letters on their shirts. There are two oranges for each child's letter except one, for which there is only one orange. |

*Teaching/Learning Strategies*:

- Every child is given an orange in each hand (except that one child has one orange) randomly.
- Oranges are passed between children until everyone has two oranges with his/her own letter.
- Only an empty hand can receive an orange, and only from an adjacent hand.

*Assessment and Evaluation*:

- Children should learn that holding onto one's own orange as soon as it is received may prevent the whole group from achieving its goal.

*Accommodations*:

- A bag of oranges

*Resources*:
See **http://csunplugged.org/** to learn more about this activity.

***Activity*: You Can Say that Again**

| | |
|---|---|
| *Time*: | 4 hours |
| *Description*: | Text can be compressed by taking advantage of patterns in words and linking repeating patterns to each other without rewriting them. For instance, "pitter patter" can be encoded by replacing the last instance of "tter" by a link to the first instance. |
| *Level*: | 1 (Grades 3–5) |
| *Topics:* | Develop a simple understanding of an algorithm |
| *Prior Knowledge*: | English, recognizing patterns in words, copying written text; basic familiarity with computers |
| *Planning Notes*: | Images and text containing millions of pieces of information are transmitted on the Internet every day. To save time and space, they are compressed into ZIP or GIF format before they are transmitted. |

*Teaching/Learning Strategies*:

- Students successfully encode and decode text, using worksheets.
- They also discuss the kinds of texts and images that compress best/worst using this algorithm.

*Assessment and Evaluation*:

- Students' completed worksheets are evaluated.

*Accommodations*:

Four different worksheets are used to facilitate this activity; a transparency is used to present the compression algorithm.

*Resources:*

See **http://csunplugged.org/** to learn more about this activity.

### A.2 Sample Activities for Level 2

***Activity*: Connections Inside and Out**

*Time:* 3-2/3 hours

Description: Students view the video *The Journey Inside The Computer* (from Intel Corporation (http://secure.wesweb.com/intel/form.htm) and examine the individual internal components of the computer. Using resources available to them, students discover the importance of each component and its impact on the computer's operations. The activity culminates with a series of problems that students must solve using the new knowledge. Finally, students use this information to suggest an alternative placement of computers within the school environment that makes a positive impact on the school community and demonstrates wise use of resources.

*Level*: 2

*Topics*: Students will gain a conceptual understanding of the principles of computer organization and the major components (input, output, memory, storage, processing, software, operating systems).

Students will gain a conceptual understanding of the basic components of computer networks (servers, file protection, queues, routing protocols for connection, communication, spoolers and queues, shared resources, and fault-tolerance).

*Prior Knowledge*: the differences between hardware and software;
ability to record findings from observation,
familiarity with the operating system they are using and the term network, and
familiarity with internal components and their uses.

*Planning Notes:*

- Request permission for students to visit certain areas of the school during class time—plan this as an in-school field trip.
- Think of visiting a music midi lab, communication lab, front office, and any specialized resources specific to your local environment.
- Check with the site administrator if you are not sure of network type(s) available in the school.
- Prepare checklist of terms for student use during video.
- Arrange to have a computer site administrator from the school or board office or a computer technician speak to the class about networks and operating systems
- Have a school map available for students to take on tour and an overhead of the map for review.
- Check for materials from *The Journey Inside The Computer* kit available from Intel (Intel Corporation. *The Journey Inside*. Part of *The Journey Inside Education* kit.)

*Teaching/Learning Strategies*:

- Show *The Journey Inside The Computer* video, Unit 4 on Microprocessors, then Unit 6 on Networking, with the purpose of reviewing computer components and extending student knowledge of networks and operating systems.
- Take up terms sheet and have students complete definitions for words they are unfamiliar with (teachers may introduce students to the online dictionary at www.dictionary.com).
- Share information on networks with students.
- Indicate type(s) of networks currently used in the school environment.
- Share information on operating systems with students.
- Deliver short test on networks and operating systems.
- Provide each student with a map of the school and explain tour route and any special routines required for secure areas.
- Give students a simple key for marking on map (e.g., C = stand alone computer, L = lab, SL = specialized lab, S = server room, P = printer resource).
- Return to the classroom and review the map on an overhead with input from students.
- Encourage a discussion of how improvements that have been made in network and operating systems make a difference in a computer community such as a school.
- Ask them to reflect on why they think the computer resources have been placed in the school the way they are.
- Ask students to prepare a written brief of changes they would like to see in the school computer environment.

- dDirect students to include positive impact(s) their suggestions have on the school environment and incorporate their knowledge of networks.
- Facilitate student pair/square and share of suggestions.

*Assessment/Evaluation Techniques*:

- A formative assessment in use of the review terms sheet, and
- An evaluation of test on networks and operating systems.

*Accommodations*:

- Give an oral test if appropriate.
- Provide students with physical disabilities assistance if required.
- Assist students with special needs with terms sheet.

*Resources*:

Course Profile: Computer and Information Science, Grade 10 Unit 4: The Computer and Society (page 116), Ontario Ministry of Education (www.acse.net/resources.htm)

***Activity*: Culturally Situated Design Tools—Weaving**

*Time*: 4 hours

*Description*: In this lesson, students learn how computers can be used as a tool for visualizing data, modeling and design, and art in the context of culturally situated design tools. Connections between the design of the tools and mathematics will be explored.

*Level*: 2

*Topics*: Interact with content-specific models and simulations (e.g., ecosystems, epidemics, molecular dynamics) to support learning and research; The connection between elements of mathematics and computer science, including binary numbers, logic, sets and functions; Collaboratively design, develop, publish, and present products (e.g. video, podcasts, wikis, etc.) using technology resources that demonstrate and communicate curriculum concepts.

*Prior Knowledge*: Understanding of Cartesian coordinate system.

*Planning Notes*:

- Ensure that csdt.rpi.edu is available to all students
- The bead loom tutorial is online; the other two are not. You can adapt the other two from the bead loom tutorial and provide printed copies.
- Create rubric for project and provide it to students in advance
- Review Cartesian coordinate system

*Teaching/Learning Strategies:*

- Post the possible design tools:
  - Virtual Bead Loom
  - Pacific Northwest Basket Weaver
  - Navajo Rug Weaver
- Display the first page of each tool in order to give students an idea of what each does. (http://csdt.rpi.edu )
  - Students divide into groups to work on the tool of their choice. Group sizes will depend on the size of the class. You may need to have more than one group per tool.
- Each member of the group should go through the entire cultural background section individually.
  - Answer any questions posed in the section in their journal.
  - Look for and write down the mathematical connections.
- All group members discuss the section.
  - Resolve answers to questions and mathematical connections.
- Each member of the group completes the tutorial.
  - Students should go through the tutorial at their own pace, but discuss with other member as questions arise.
  - Encourage students to record in their journal points that they want to remember.
- Groups create designs using the design tool software.
  - Each person should choose one of the goal pictures for practice and discuss any issues with the other group members.
  - Groups decide whether they want to create one design as a group or have multiple designs for their presentation.
  - Groups work on design/designs—these should be their own creations rather than a mimic of one of the preloaded designs.
- Edit designs with Photoshop Express.
  - Have students watch the online tutorial and create an account.
  - Edit the design.
- Prepare presentations to include:
  - Culture

- Math connections
- Demo of software
- Display of designs

- Groups deliver presentations.

*Assessment/Evaluation Techniques*:

- Students are assessed on their projects, including design, description of culture, math connections, and demo. Students should also be assessed on their work as a team.

*Accommodations*:

- Use extensive visual aids and demonstrations to assist students as needed.
- Provide appropriate adaptive devices or implementation accommodations for identified students.
- Construct student groups to enable assistance where necessary

*Resources*:

- Culturally Situated Design Tools—http://www.csdt.rpi.edu (site and adaptations of tutorials courtesy Ron Eglash)
- Virtual Bead Loom Tutorial
- Pacific Northwest Basket Weaver Tutorial
- Navajo Rug Weaver Tutorial
- Culturally Situated Design Tools Project Sample Rubric
- http://photoshoponline.com/

***Activity*: Number Systems**

*Time*: 4 hours

*Description*: Students develop an understanding of the relationship between the binary number system and computer logic. Also, students learn how to convert Base 10 numbers into binary and vice versa. Character representation of binary codes is explored. Students have the opportunity to experiment in writing their own message and decoding.

*Level*: 2

*Topics*: The connection between elements of mathematics and computer science, including binary numbers, logic, sets and functions.

*Prior Knowledge*: Understanding of the decimal number system and place value

*Planning Notes*:

- Review how programming software handles character representations.
- Have eight pennies for each pair of students and either a handout and/or overhead of bit information
- Review binary and base 10 conversions.
- Prepare coded messages for the students to decipher.
- Have copies of ASCII code available (both standard and extended).

*Teaching/Learning Strategies:*

- Show segment 3 of The Journey Inside video (8 min 25 sec—Intel Corporation. The Journey Inside. Part of The Journey Inside Education kit), or any other video that shows how computers turn pictures and colors into codes. Students gain an understanding of how information is communicated through the use of codes.
- Hand each pair of students eight pennies and work through the questions on bit information. Ask students what pattern they can see forming in the right column (numbers double).
- Students are challenged to count as high as they can on one hand and told the answer is greater than 10. While students ponder the challenge, teachers demonstrate, with the aid of a simple series circuit, the binary logic states of ONE and ZERO (TRUE and FALSE, HIGH and LOW) by equating them to series circuit lamp ON and OFF condition.
- Binary numbers are introduced by initiating finger counting on one hand—no fingers up is zero, thumb up is a one, index finger up is two, middle finger up is a four, ring finger is eight, and pinkie finger represents sixteen. Students demonstrate counting to 31 on one hand.
- This sets the stage for demonstrating how to convert numbers from Base 10 to Base 2 (binary). Work through several examples with students.
- Give students a quiz on binary conversion to assess their grasp of the concept.
- Handout the ASCII conversion information. Since computers cannot think like we do, they need a code to translate our language into data that they can process and then convert that data back into recognizable language.
- Students complete conversion exercises.

*Assessment/Evaluation Techniques*:

- Formative assessment of quiz at the end of the binary conversion exercise to prompt students on progress and show changes required for success of conversion application.
- Summative assessment of conversion exercises.

*Accommodations*:

Use extensive visual aids and demonstrations to assist students as needed.

Provide an enlarged copy of conversion methodology in classroom as well as ASCII character chart.

Use a variety of teaching styles to accommodate learning styles.

Provide appropriate adaptive devices or implementation accommodations for identified students.

*Resources*:

**Adapted from the course profile for Computer Engineering Technology, Grade 10, Unit 2: Integrated Circuits (page 53) Ontario Ministry of Education (www.acse.net/resources.htm)**

***Activity*: Pass-it-on**
*Time*: 90 minutes
*Description*: In this activity, students collaboratively and incrementally construct projects using Scratch by passing the projects from student to student.
Level: 2
*Topics*:

- Collaborate with peers, experts, and others using collaborative practices such as pair programming, working in project teams, and participating in group active learning activities (CS unplugged, KLAs, etc.).
- Design, develop, publish, and present products (e.g. Webpages, mobile applications, animation, etc.) using technology resources that demonstrate and communicate curriculum concepts.

Prior Knowledge: Students should have at least a basic familiarity with Scratch and its mechanism of snapping blocks together to specify program instructions.
Planning Notes:
In preparing for this activity, teachers should:

- Select a theme for the pass-it-on project. Some themes that we have tried before and have been popular include pass-it-on stories and pass-it-on dance parties. The content of pass-it-on projects can also be connected to various curricular areas.
- Decide which Scratch features and computational concepts will be demonstrated in the opening introduction.

*Teaching/Learning Strategies*:
This activity involves the following steps:

- The teacher models the activity by starting a pass-it-on project. For example, the teacher might demonstrate how to start a dance party project, by adding music, a background, and a party-goer. The teacher explains the pass-it-on process and, as an example, makes (or solicits) two or three suggestions for elements that could subsequently be added to the model project (e.g. additional party-goers, more costumes).
- The students (either individually or in pairs) each use a computer to start their pass-it-on projects. They have 15 minutes to work on their projects.
- After 15 minutes, the students rotate to a new computer and continue building the project they find at the new computer. They have 15 minutes to work on their projects.
- After 15 minutes, the students rotate a final time and work on a third project. They have 15 minutes to work on their projects.
- At the end of this final 15 minutes, the students return to their original computer to see how the project they started has evolved.
- Depending on the number of projects, students can walk around to view all (or some subset) of the other pass-it-on projects.
- The teacher facilitates a discussion about the concepts and features students learned as they worked on the projects and looked at others' code.

*Assessment and Evaluation*:

- The closing discussion - The discussion can be used to make explicit the different concepts and features students learned and the challenges they experienced.
- The pass-it-on Scratch projects - The projects can be evaluated for process (e.g. the nature of the collaborative activity, how students supported each others' learning) and product (e.g. the feature and concepts that were included in the project, the ways in which the theme was creatively appropriated). This evaluation can include self-evaluation, peer-evaluation, and instructor-evaluation.

*Accommodations*:
Although this activity can be completed without supplemental materials, the concepts and features demonstrated in the teacher's initial modeling can be supported by making handouts available that suggest things to try. For example, the following handouts could be accessible nearby as students work on their creations:

- Scratch cards: A set of 12 code excerpts. http://scratched.media.mit.edu/resources/scratch-cards
- Monkey business: A one-page suggestion of introductory blocks to experiment with. http://scratched.media.mit.edu/resources/monkey-business

*Resources*:

- Video documenting the "Pass-it-on story" activity, as done with a group of teenage girls. http://scratched.media.mit.edu/stories/csed-week-day-4-wise-dance-party
- ScratchEd online community. http://scratched.media.mit.edu/

***Activity*: Setting up a Computer**

*Time*: 2-1/2 hours

*Description*: Students set up a computer including installing available software and an operating system. Students connect, configure, and test all peripherals. Finally, students troubleshoot any problems that arise. All students set up a PC.

*Level*: 2

*Topics*: Students will gain a conceptual understanding of the principles of computer organization and the major components (i.e. input, output, memory, storage, processing, software, operating systems).

*Prior Knowledge:* Components of a computer system, correct terminology

*Planning Notes:*

- Prepare available samples of micro-controllers and PCs of various types.
- Determine the most effective use of existing hardware within the recommended time allotment (e.g., two to three students per computer).
- Open an older discarded hard drive for demonstration purposes.
- If resources are limited, a single system may be set up several times to accommodate all students.
- This activity is done with stand-alone machines to not interrupt a networked environment.
- The teacher should review the procedures in the attached appendices. This activity assumes that the computer system hard drive has been configured prior to the installation of the operating system.
- The actual system installation can be performed as a class “walk through”. The teacher can modify the process to have the individual groups perform the set-up task.
- The teacher should review the disk partitioning, formatting, scandisk operations, and information available in the Help files of the operating system (see Resources).
- Inventory the operating system CD-ROM and software key.
- Ensure all software is available for the full installation including operating systems, device drivers, and application software.

*Teaching/Learning Strategies:*

- Teachers and students review safety with static electricity and the importance of keeping contacts clean as they apply to components. Review the safety considerations when setting up a desktop computer (grounded plugs, using power bars, dangling cords, eliminating the danger of static electricity, and unplugging power supply before opening a PC, etc.).
- The teacher explains how hard drives work so that students can understand the utility functions they are required to complete by the end of the activity.
- Students use the equipment they require to complete the task, including the monitor, CPU, keyboard, mouse, and a printer, if available. The teacher explains any special considerations they need to know (e.g., positioning of computers for plugs in the room). Students use this information to create a checklist for the activity.
- Depending on the resources available, divide students into the appropriate number of groups. Students connect all the parts of their computer system. Circulate to help with troubleshooting and use questioning techniques to assist with problem solving.
- Once all components are connected, students load the operating system software. Students complete their personal checklist to keep in their portfolios.
- All groups must then test their software to ensure their system is working and that all peripherals connected are functioning properly.

*Assessment/Evaluation Techniques:*

- A formative assessment through student discussion and observation, encouraging students to assess their thinking for successful completion of task.
- Assess student-created checklists. Provide students with written/oral feedback, to assist their success in upcoming related activities.

*Accommodations:*

- Provide step-by-step instructions.
- Provide a glossary of terms.
- Provide visuals of different computer types.

*Resources*:

- Course Profile: Computer Engineering Technology, Grade 10, Unit 3: Networking (page 72) Ontario Ministry of Education (www.acse.net/resources.htm

**A.3 Sample Activities for Level 3**

***Activity*: Careers in Computer Engineering**

*Time:* 3-3/4 hours

*Description*: A guest speaker is invited to share information about his/her job/career with the students. Students expand on their computer industry knowledge. Students look at degrees and certifications available and opportunities they have at the high school level and beyond to move them toward careers in the computer industry.

*Level*: 3A, 3B (could also be used for Level 2 with some modification)

*Topic*: Students will gain a conceptual understanding of the identification of different careers in computing and their connection with the subjects studied in this course (e.g. information technology specialist, Web page designer, systems analyst, programmer, CIO).

*Prior Knowledge*: Word-processing skills

*Planning Notes*:

- Guest speakers may include the school sysop, board technician, or someone from the local community.
- Collect information from a local university or community college, including school course calendars and college/university catalogues.
- Gather copies of recent computer trade magazines.
- Arrange ahead of time for a student to introduce guest speakers and another student to thank them.
- Collect newspaper advertisements for jobs in the computer industry.
- Distribute a sample certification worksheet

*Teaching/Learning Strategies*:

- Teachers introduce the expectations of the activity.
- Teachers review with students (ahead of time) questioning techniques for the guest speaker.
- One student may introduce the guest speaker. Students take brief notes in order to ask relevant and interesting questions. One student may thank the guest speaker.
- Discuss the speaker information with the students, after which they write their personal views on the information.
- Students look through trade magazines to see advances in the computer industry. Each student picks one article from a magazine to summarize or review using a word processor.
- Finally, students look at opportunities for different computer designations ranging from MCSE (Microsoft Certified Engineer) to computer engineering at the university level. Students use newspaper advertisements to explore what skills and designations are requested by potential employers.
- Students retrieve the certification chart file (either electronically or via handout) and, using designations discovered in the advertisement exercise above, they complete the chart and add it to their portfolio.
- Students create a plan on how to pursue a computer career, beginning with the completion of this course, and save the information in their portfolio (long-term goal).

*Assessment and Evaluation*:

- Review of student portfolio to provide written/oral feedback on completion and comprehension of tasks given.
- Evaluate the article review using the rubric provided.

*Accommodations*:

- Allow flexible timelines for due date of report.
- Use career center videos if available.
- Invite the Student Services resource personnel into the classroom.
- Videotape the guest speaker(s) presentation to allow students an opportunity to watch it again.

*Resources*:

Course Profile: Computer Engineering Technology, Grade 10, Unit 3: Networking (page 93), Ontario Ministry of Education (www.acse.net/resources.htm)

***Activity*: Culturally Situated Design Tools—Cornrow Curves**

*Time*: 3 hours

*Description*: This lesson serves as a reinforcement of the problem-solving process and connections between mathematics and computation within the context of a culturally situated design tool.

*Level*: 3A, 3B

*Topics*: Explain how sequence, selection, iteration, and recursion are building blocks of algorithms; Describe how computation shares features with art and music by translating human intention into an artifact; Describe how computing enhances traditional forms and enables new forms of experience, expression, communication, and collaboration.

*Prior Knowledge*: Understanding of Cartesian coordinate system.

*Planning Notes*:

- Ensure that csdt.rpi.edu is accessible to all students
- Create rubric for project and provide it to students in advance
- Review Cartesian coordinate system

*Teaching/Learning Strategies:*

- Cultural background of cornrow braiding
  - Students read the cultural background and how to braid sections (csdt.rpi.edu, Cornrow Curves).
- Group discussion on cultural background of cornrow braiding
  - Divide students into groups of 3-4 and ask each group to reflect on one of the following sections:
    - African Origins
    - Middle Passage
    - Civil War to Civil Rights
    - Hip Hop
  - Each group shares their response with the rest of the class.
- Cornrow curves design tool tutorial
  - Individual students complete Part I of the tutorial following all instructions and checking their work with their elbow partner.
  - Discuss any issues as a class before proceeding to Part II.
  - Complete Part II of the design tutorial.
  - Stress mathematics and structured inquiry.
- Cornrow curves project
  - Students create their own design.
  - Describe each step of the problem-solving process used.
  - Highlight the mathematical concepts used and where used.
  - Reinforce the strategy of finding a similar problem that has already been solved to help solve the new problem.
- Gallery walk of designs
  - Students share their solutions.

*Assessment/Evaluation Techniques*:

- Students are assessed on their projects, including design, description of process used, and description of mathematical concepts used.

*Accommodations*:

- Use extensive visual aids and demonstrations to assist students as needed.
- Provide appropriate adaptive devices or implementation accommodations for identified students.
- Construct student groups to enable assistance where necessary

*Resources*:

- Culturally Situated Design Tools Cornrow Curves—csdt.rpi.edu (courtesy Ron Eglash)

***Activity:* New Solutions for Old Problems**

*Time:* 5 hours

*Description*: Students examine problems that can be solved using more than one algorithm (e.g., determining the factorial value of a number). Using brainstorming or other group problem-solving techniques, students develop alternative algorithms using recursive and non-recursive techniques. Students identify the components of a recursive algorithm and develop criteria for recognizing when a recursive algorithm may be applied.

*Level*: 3A, 3B

*Topics*: Fundamental ideas about the process of program design, and problem solving, including style, abstraction, and initial discussions of correctness and efficiency as part of the software design process.
Simple data structures and their uses.

*Prior Knowledge and Skills*: use of problem-solving models, the ability to develop appropriate algorithms to solve problems, and the ability to write pseudocode.

*Planning Notes*:

- Review the nature of recursion.
- Gather examples of problems that can be solved using more than one method, including recursion, and determine which problems may be solved using a recursive algorithm.

*Teaching/Learning Strategies*:

- Divide the class into groups of two or three students.
- Review the brainstorming problem-solving technique.
- Present a problem that can be solved using a familiar but complex algorithm and may also be solved using a less familiar but simpler algorithm (e.g., determining the quotient and remainder of the division of two integers).
- Students, in their groups, develop more than one algorithm for the solution.
- The teacher facilitates a class discussion to develop criteria for the evaluation of algorithms, including the efficiency of the solution and the complexity of the required coding. Both processing and user interface efficiencies are considered.
- Groups evaluate the algorithms using the developed criteria and share their algorithms and evaluations with the class.
- The teacher introduces the recursive method of problem solving and illustrates a recursive algorithm for the solution to a different problem (e.g., calculating the factorial value of a number).
- Groups develop a recursive algorithm to the initial problem and evaluate its efficiency.
- The teacher facilitates a class discussion to establish criteria for determining if a recursive algorithm is an appropriate solution and identifies additional problems that may be solved using recursion.
- Working in groups, students develop recursive and non-recursive algorithms for additional, assigned problems.

*Assessment and Evaluation*:

- A formative assessment of the assigned in-class work in the form of roving conferences, and
- A summative assessment in which students complete an assignment requiring the development of both a recursive and a non-recursive algorithm.

*Accommodations*:

- Provide print copies of examples of algorithms using recursive and non-recursive methods, including graphic illustrations, and use models to illustrate the algorithms.

***Activity:* Planning a Solution**

*Time:* 6 hours

*Description:* Students work in groups to analyze complex problems (e.g., Towers of Hanoi) and to develop appropriate algorithms using recursive and non-recursive techniques. Students create pseudocode and design charts to assist them in planning a solution and assess these representations of code as problem-solving tools.

*Level:* 3A, 3B

*Topics:* Fundamental ideas about the process of program design, and problem solving, including style, abstraction, and initial discussions of correctness and efficiency as part of the software design process and

Principles of software engineering: software projects, teams, the software life cycle.

*Prior Knowledge and Skills*: students can apply the steps in the software design life cycle; use pseudocode, diagrams, and charts to summarize program design; and develop appropriate algorithms to solve problems.

*Planning Notes*:

- Review top-down problem solving.
- Select a problem to use in developing a model solution and prepare the appropriate models.

*Teaching and Learning Strategies*:

- The class is divided into groups of two or three students and each group is assigned a problem.
- Groups investigate the problem, using a variety of problem-solving techniques to analyze it.
- Each group uses brainstorming or other group problem-solving techniques to develop an algorithm for the solution to the problem. In a class discussion, groups present and share their algorithms.
- Students compare the effectiveness and efficiency of the algorithms presented and then the groups refine their algorithms.
- Each group develops a flow chart, structure chart, and/or pseudocode to represent the application of the algorithm. The teacher conferences with each group to discuss and assess the solution design.

*Assessment and Evaluation*:

- A formative peer assessment of the presented algorithms.
- A formative assessment of the design for the solution to the problem.

*Accommodations*:

- Provide print sample algorithms similar to the one studied.
- Use graphical models to illustrate the problem.
- Selectively pair/group students to assist problem solving.
- Provide problems of varying complexity to provide an appropriate challenge.

***Activity*: Role Playing Helper Functions/Recursion**

*Time:* 1 hour

*Description*: Students role play various objects of simple programs to understand parameter passing and recursive calls

*Level:* 3B**,** 3C.1

*Topics:* Methods (functions) and parameters, recursion

*Prior Knowledge and Skills*: compile and run simple programs; write code using parameters.

*Planning Notes*:

- Prepare or obtain from resources scripts for role playing objects in a small program with several nested (and usually also recursive) calls.
- Gather colored markers and poster board as needed.

*Teaching/Learning Strategies:*

- Review the concepts of constructors, parameters, and calling helper methods.
- Students read code for the program to be used for the role play.
- Select a student to role play the main function. If desired, give student a large name tag to wear. Select another student to be the code monitor whose job is to keep a record of the current line of code being executed.
- Assist class as they act out the script, each time an object is constructed the student calling the constructor function picks a classmate to play the role of the object; if using name tags, be sure to give each object-player a name tag. Using different sized, shaped, or colored tags for different classes is helpful.
- Frequently pause the play and ask audience members to identify who the next actor will be.

*Assessment and Evaluation*:

- A formative assessment of the role play in the form of roving interviews.
- A summative assessment can be administered asking students to indicate the number of objects in existence as the play progressed and similar questions.

*Accommodations*:

- Let pairs of students play a role.
- Assign roles yourself giving simpler roles to students who are struggling.

*Resources*:

- Several role playing exercises are available at: http://cs.colgate.edu/APCS/Java/RolePlays/JavaRolePlays.htm

***Activity*: Short Programming Challenges in Scratch**
*Time*: 90 minutes
Description: Students construct solutions to short programming challenges that explore interactivity in Scratch and discuss the various solutions and possible applications of these challenges in their programs.
*Level*: 3
*Topics*:

- Apply algorithm problem solving to solving computational problems.

Prior Knowledge: Students should have some familiarity with the Scratch environment, connecting blocks to create programs, creating sprites, managing program flow, and triggering events.
Planning Notes:
Before the activity, teachers should select challenges to go through with their students based on what Scratch features and computational concepts they would like to explore. For example, the teacher may consider the following challenges:

- Whenever two sprites collide, one of them says: "Excuse me."
    - Scratch features: If, Touching blocks
    - Computational concepts: Condition, Events
- Whenever you click on a sprite, all other sprites do a dance.
    - Scratch features: Broadcast, When Sprite clicked block
    - Computational concepts: Events, Parallelism, Synchronization
- When the score reaches 10, the scene (background) changes.
    - Scratch features: If, Operators, Variables blocks
    - Computational concepts: Condition, Operators, Variables

More challenges can be found at this link: http://scratched.media.mit.edu/resources/short-scratch-programming-challenges
*Teaching/Learning Strategies*:
For each challenge, teachers should:

- Present students with a challenge. Encourage them to work with their peers to come up with a possible solution.
- After 15 minutes, facilitate a discussion around the diversity of solutions that students developed to address the given challenge. Ask students how they might use their solutions to create other Scratch projects. For example, the challenge "Whenever two sprites collide, one of them says: "Excuse me." can be adapted into a project with a story that involves the interaction of multiple sprites.

*Assessment and Evaluation:*
In the group discussion, students can articulate their solutions and evaluate the solutions of others to address each challenge.
*Accommodations*:
Teachers may want to print out the challenges for students to have on hand.
Resources:

- Description and video of Scratch Challenges Activity with a group of educators http://scratched.media.mit.edu/resources/short-scratch-programming-challenges
- ScratchEd website http://scratched.media.mit.edu/

**A.4 Sample Activities for Level 3C: Topics in Computer Science**

***Activity*: Introduction to Object Oriented Design**

*Time*: 3 hours

*Description*: Students are introduced to the initial steps of applying Object Oriented Design to a programming problem and practice applying those steps to a problem which may later be used as a significant programming project

*Level:* 3C.1 AP Computer Science, 3C.2 Projects

*Topics:* Fundamental ideas about the process of object oriented program design

*Prior Knowledge and Skills*: none

*Planning Notes*:

- Prepare blank diagrams for use as CRC cards and/or object diagrams
- Select examples of problems that can be solved by using object oriented techniques.

*Teaching/Learning Strategies:*

- Explain the differences between an object oriented and a functional approach to the design of a computer program.
- Have a student explain how a card game (like blackjack) is played.
- Working with the class, identify potential objects involved in the game.
- Working with the class, identify possible operations that might be done by or to a card or one possible object. Students in small groups brainstorm operations for other objects identified as part of the problem.
- Illustrate how the two notations can be used to summarize the analysis so far.
- Discuss possible relationships between objects.
- Show notations for relationships.
- Describe a possible scenario in the game and use developed notations to represent that scenario.
- Have students in pairs describe a second scenario and represent it.
- Share class scenarios and consider whether they collectively represent the range of possibilities especially extreme scenarios.
- Students read the possible problems and select one on which to do an OOD. Students work individually on the first two phases (identify objects and operations).
- Students working on the same problem share results and agree on a "best" set of objects and operations.
- Students individually complete the last two steps.

*Assessment and Evaluation*:

- A formative assessment of the assigned in-class work in the form of roving conferences.
- A summative assessment applying OOD to a new problem

*Accommodations*:

- Provide copies of problem descriptions, with important nouns and verbs indicated using a different font or type size. Provide written scenarios for each problem.

*Resources*:

- Wirfs-Brock, Wilkerson, and Wiener, *Designing Object-Oriented Software*, Prentice Hall, 1990
- Fowler, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, 1997.
- Overview of lesson and sample exercises are available at:
  http://max.cs.kzoo.edu/AP/OOD/OODPresentation/

http://max.cs.kzoo.edu/AP/OOD/OODSpecifications/

**A.5 Additional Resources for Level** 3C.2 and 3C.3:

A wide range of resources is available for supporting a Level 3 computer science curriculum.

**AP Computer Science course**
The curriculum for all AP CS courses is governed by the topic outline available from The College Board at http://www.apcentral.collegeboard.com/. Two example activities to enhance learning for the AP curriculum are shown earlier in this Appendix.

**Courses Leading to Industry Certification**
Many of the certification courses provide a prepared curriculum that details the content and order of topics. While implementation of this type of course may be simplified by the information provided, careful evaluation is needed with regard to proprietary content versus general concepts. Information about the content of some certification courses can be found online by searching for the specific certification.

**Projects-Based Courses**
Projects-Based courses can provide targeted education geared toward specific student interests in Computer Science. While these courses can be offered by the local school district, enrolling students in a college-based course should be considered. Computer projects-based courses are often offered in a college or university department of computer science, information technology, or information systems. Computer-based programs of study at the college level are typically well established and provide a variety of current topics. Additionally, a student participating in a college computer course may be eligible for college credit. The secondary school will need to investigate the content of each course to determine if it is appropriate for Level 3C study.

Courses for college credit can be offered on the college campus, at the high school, or via distance education. A course offered on the college campus will place less burden on the high school for supporting special hardware, software, and faculty resources.

Online courses can be taken off-hours or during school time. A student who chooses to take an online course outside of school hours will need access to specified hardware, software, and the Internet. When a college-delivered online course is offered during the school day, it may be possible to schedule several online courses during the same class session, allowing better utilization of faculty time. To support this, the school must also provide access to the required hardware, software, and the Internet.

A course for college credit offered at the secondary school permits the student and high school faculty member to interact in a traditional manner. The secondary school will need to ensure that the curriculum is sanctioned by the college and students are officially enrolled in the college. The college may wish to participate in the selection of the faculty member delivering the course.

Some typical methods by which high school students can achieve college credit are described below.

*Tech Prep and Articulation Credit*—The student takes a course at the high school and receives high school credit. The course is also pre-approved for college credit at a particular college, and is typically taught by a high school faculty member. Credit is awarded for the course upon matriculation at that college. The college may require that the student pass a competency exam before applying the credit to the student's transcript.

*Dual Credit / Concurrent Enrollment*—The student is enrolled in a course for which s/he simultaneously receives high school and college credit. The student must meet all college requirements for entrance into the course.

*Challenge Exams*—The student may be able to prove proficiency and receive credit by exam. This method is useful for a student who completes a high school course that is not articulated, or who believes s/he has independently gained knowledge of all topics covered by a specific course. The student may be charged a fee to sit for the exam.

*Advanced Placement / CLEP Test*—A student planning college study toward a career in Computer Science, Information Technology, or Engineering may wish to take an AP Exam in Computer Science. A student planning college study toward a career in Business may wish to take the CLEP Exam in Information Systems and Computer

Applications. It is recommended that the student determine how credit will be granted for success on the exam from his/her targeted institution.

***Programs That Provide College Credit to High School Students:*** A number of programs provide college credit to high school students. Here are three examples:

*The University of Pittsburgh's College in High School* is one example of high schools interacting with a college. The "College in High School (CHS) program has offered qualified high school students the opportunity to earn University of Pittsburgh college credits during their regular school day." (Pittsburgh). Students are required to pay a reduced tuition to participate in the program. Financial Aid may be available. The program offers "14 courses to more than 2,900 students in over 100 high schools with more than 200 faculty" (Pittsburgh). At the end of the course, students receive a University of Pittsburgh college transcript. Information about this program is available at http://www.as.pitt.edu/undergraduate/offices/chsp/.

*The University of Cincinnati Clermont College* also offers high school students the opportunity to earn college credit by providing a Post-Secondary Enrollment Options Program (PSEOP). This program permits the student to enroll in the college for college credit only, or to enroll for both high school and college credit. In some semesters, the cost of tuition and books may be paid by the high school and/or college. For more information see \http://www.ucclermont.edu/documents_cms/Future%20Students/2011_2012_PSEOP.pdf

*The University of Northern Colorado* provides "High School Concurrent Credit, a program for Colorado Residents enabling high school Juniors and Seniors to earn college credit while in high school" (Colorado). Three options are provided under the program:

> "Option I—Fast Track Program: For the student who is a high school senior and has met high school graduation requirements.
>
> Option II—Post-Secondary Enrollment Options Program: For the student who is a high school junior or senior and has not met high school requirements.
>
> Option III—College Acceleration Program: For the student who is a high school junior or senior and wants to accelerate his or her college program whether or not the graduation requirements have been met." (Colorado)

***Resources:***


eHow.com, How to Get College Credit Before Attending College, Downloaded 1/29/2011 http://www.ehow.com/how_4424237_get-college-credit-before-attending.html

Arthur R. Greenberg, ERIC Clearinghouse on Higher Education Washington DC, BBB27915, George Washington Univ. Washington DC. School of Education and Human Development., ERIC Identifier: ED347956, Publication Date: 1992-03-00, http://www.ericdigests.org/1992-2/high.htm

Comparison of Methods to Receive College Credit for Courses Taken in High School, http://www.tea.state.tx.us/Cate/teched/collegecreditinhs.pdf

Dawn Fuller, University of Cincinnati, High School Students: Do You Qualify for UC College Credit?, March 7, 2003, http://www.uc.edu/news/NR.asp?id=300

The College Board, AP Central, Downloaded 1/29/2011, http://apcentral.collegeboard.com/apc/Controller.jpf

University of Cincinnati Cleremont College, Post-Secondary Enrollment Options Program. Downloaded 1/29/2011 http://www.ucclermont.edu/documents_cms/Future%20Students/2011_2012_PSEOP.pdf

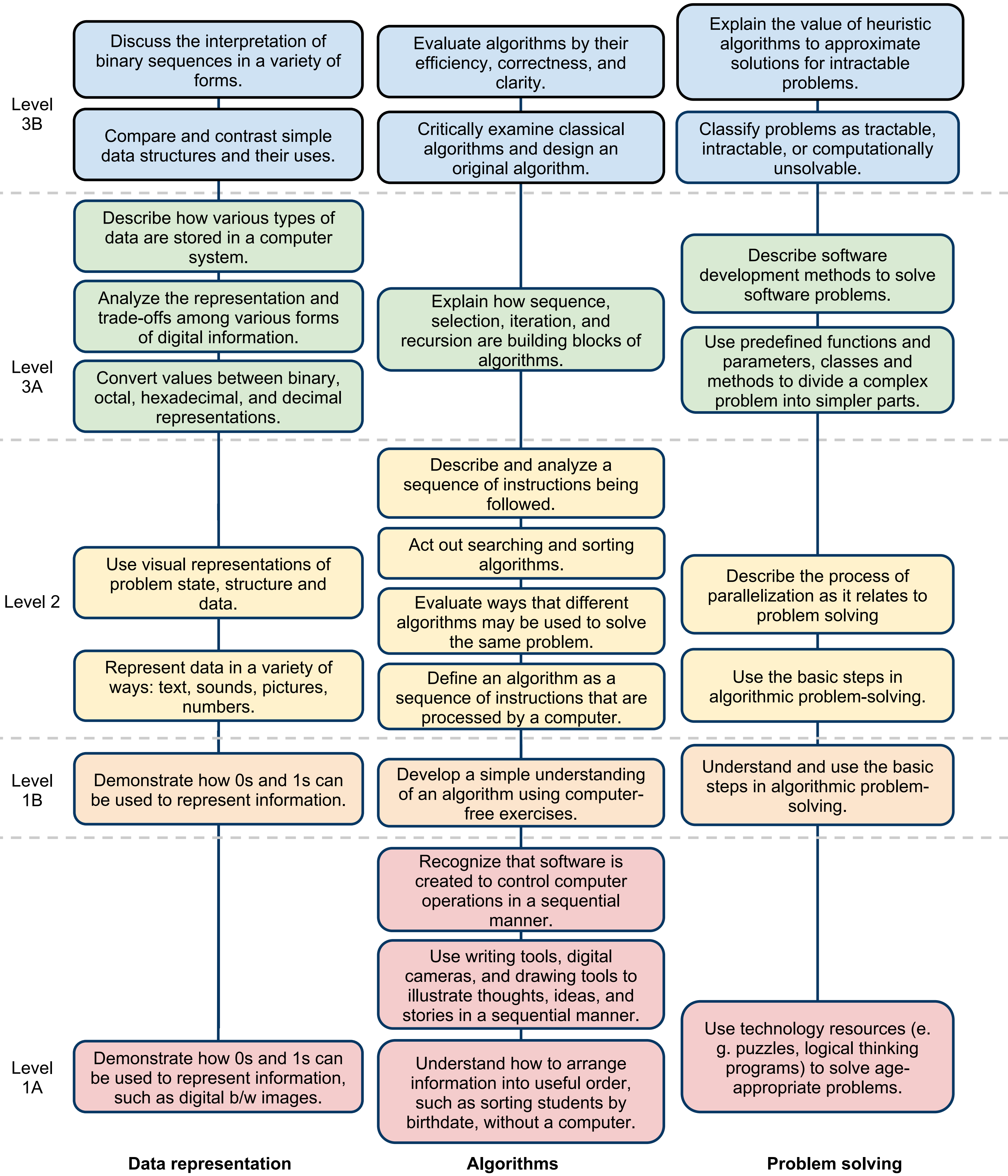
University of Northern Colorado, Undergraduate Admissions, High School Concurrent. Downloaded 1/29/2011,http://www.unco.edu/admissions/special_populations/highschoolconcurrent.asp

University of Pittsburgh, College in High School, Downloaded 1/29/2011, http://www.as.pitt.edu/undergraduate/offices/chsp/.

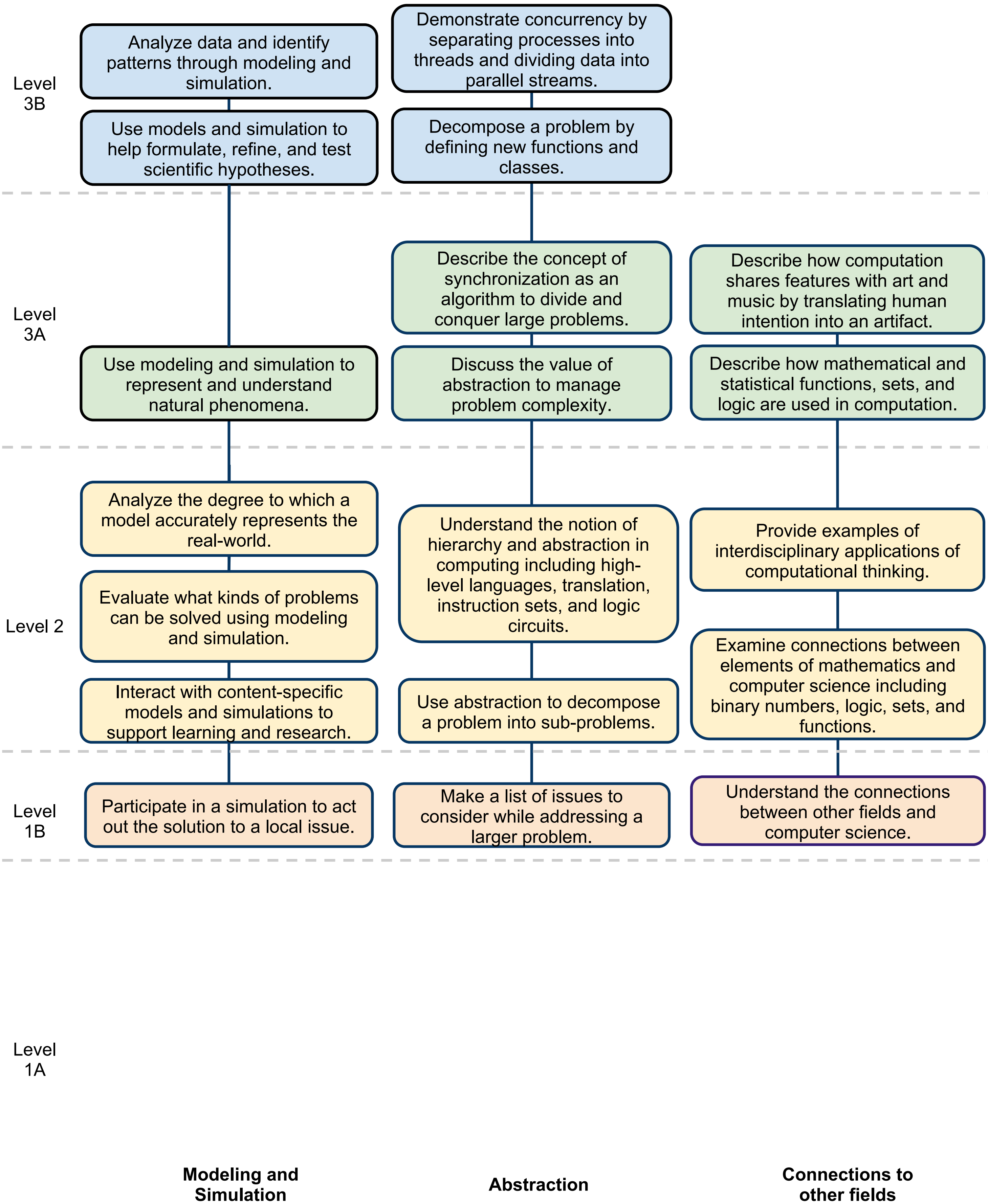
U.S. Department of Education, Community College and High School Partnerships by Elisabeth Barnett and Katherine Hughes, Community College Research Center,. Downloaded 1/29/2011, http://www2.ed.gov/PDFDocs/college-completion/09-community-college-and-high-school-partnerships.pdf

U.S. Department of Education, Tech Prep Education. Downloaded 1/29/2011, http://www2.ed.gov/programs/techprep/index.html

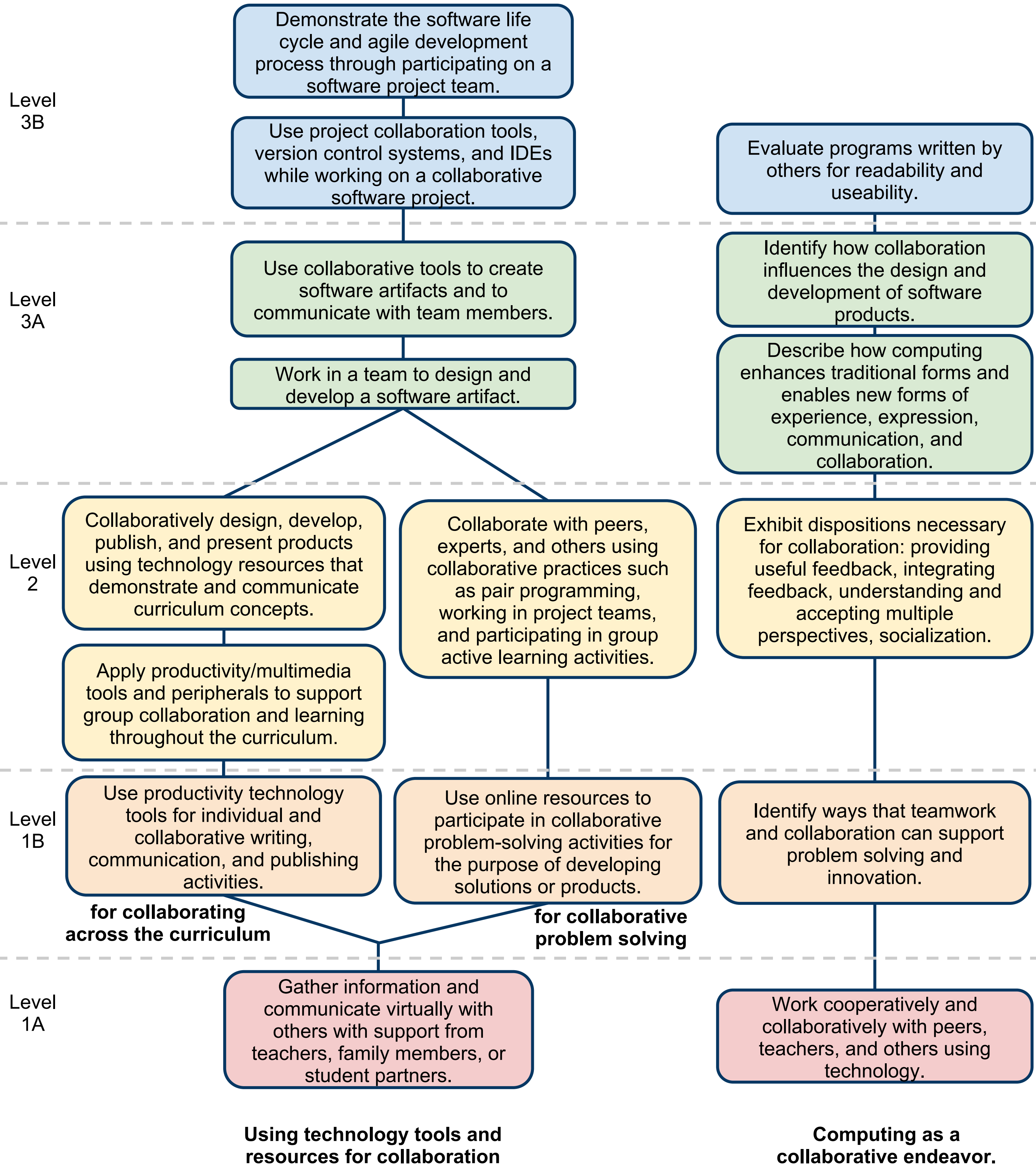
# Computational Thinking



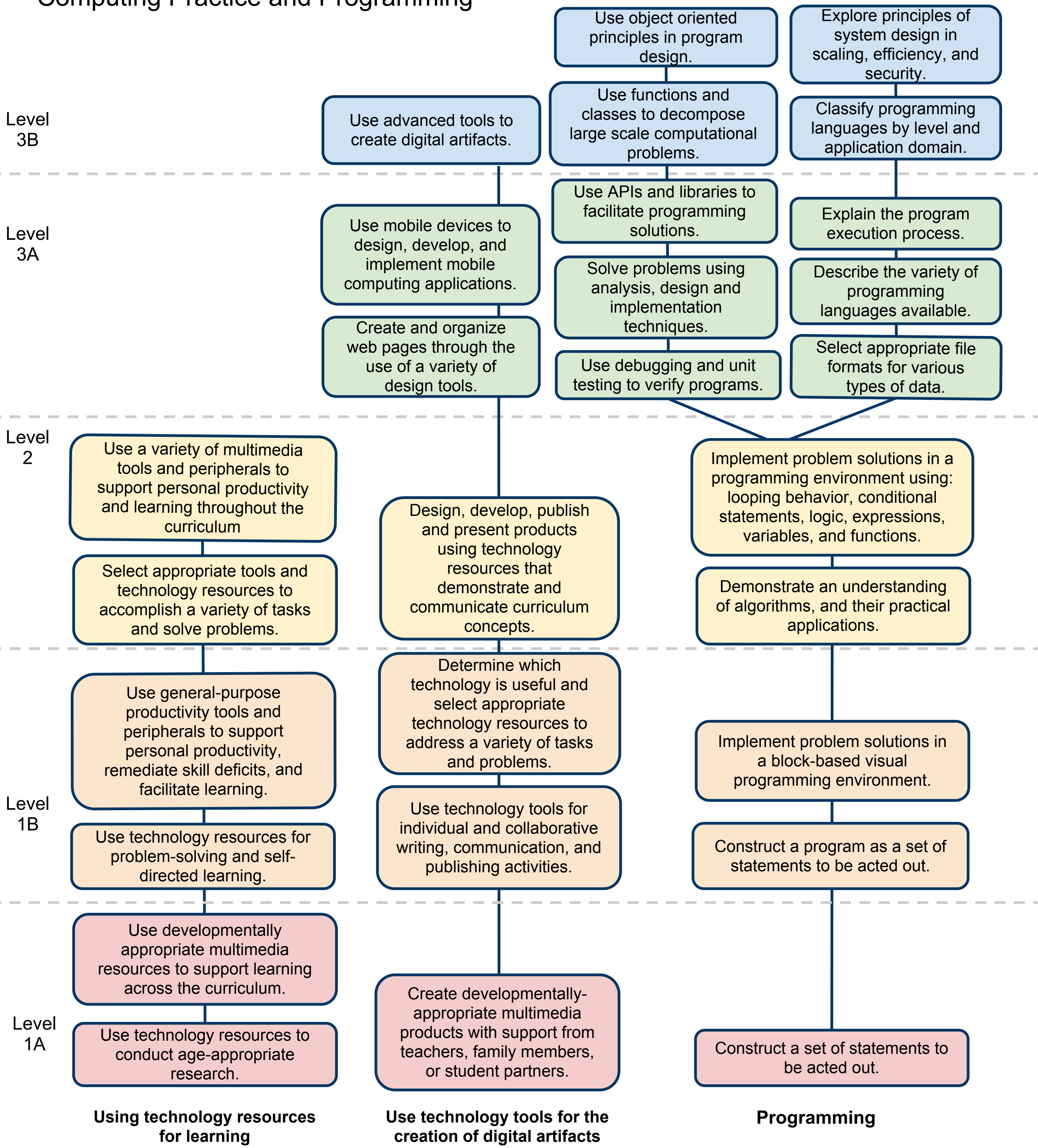
# Computational Thinking



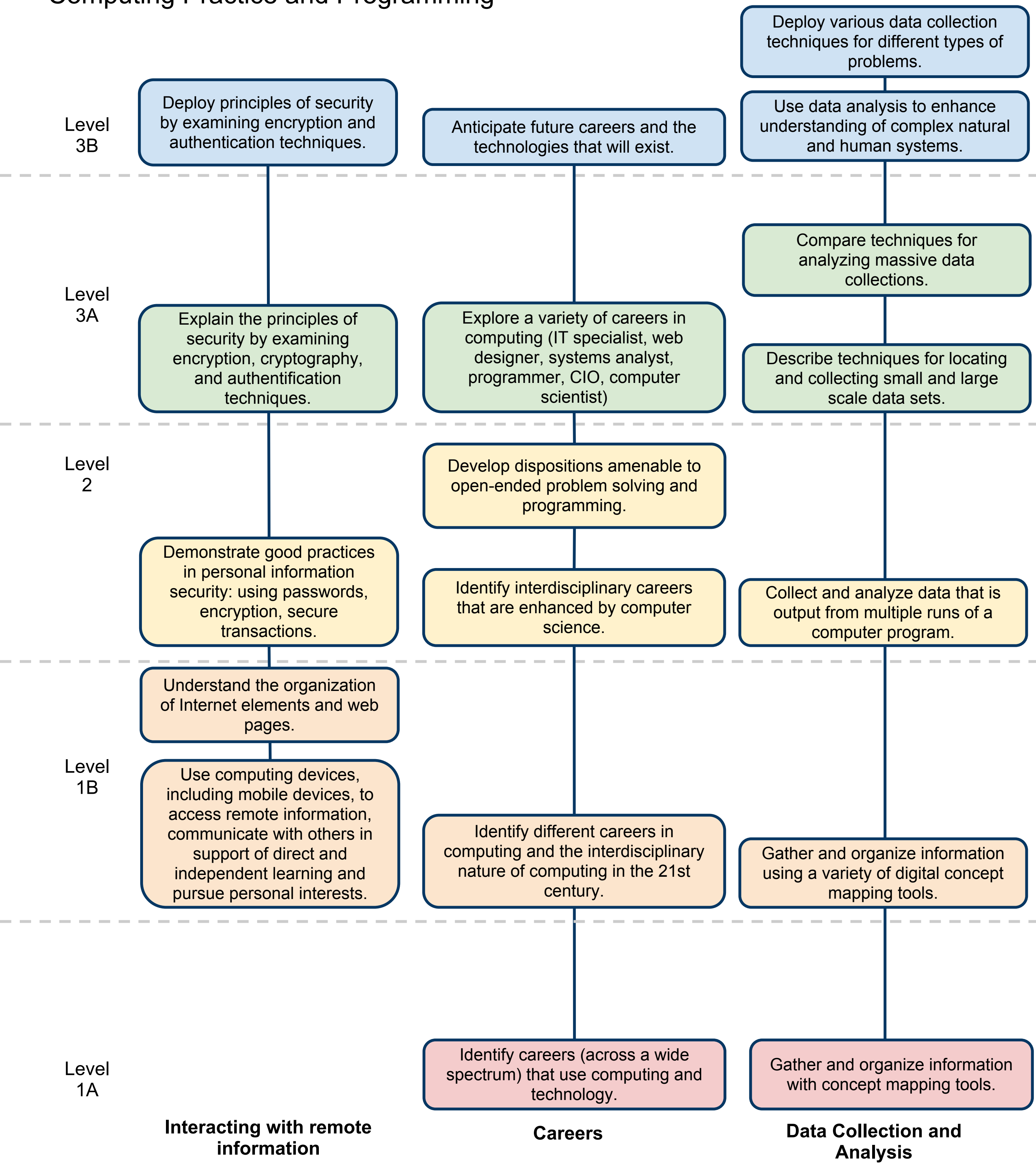
# Collaboration



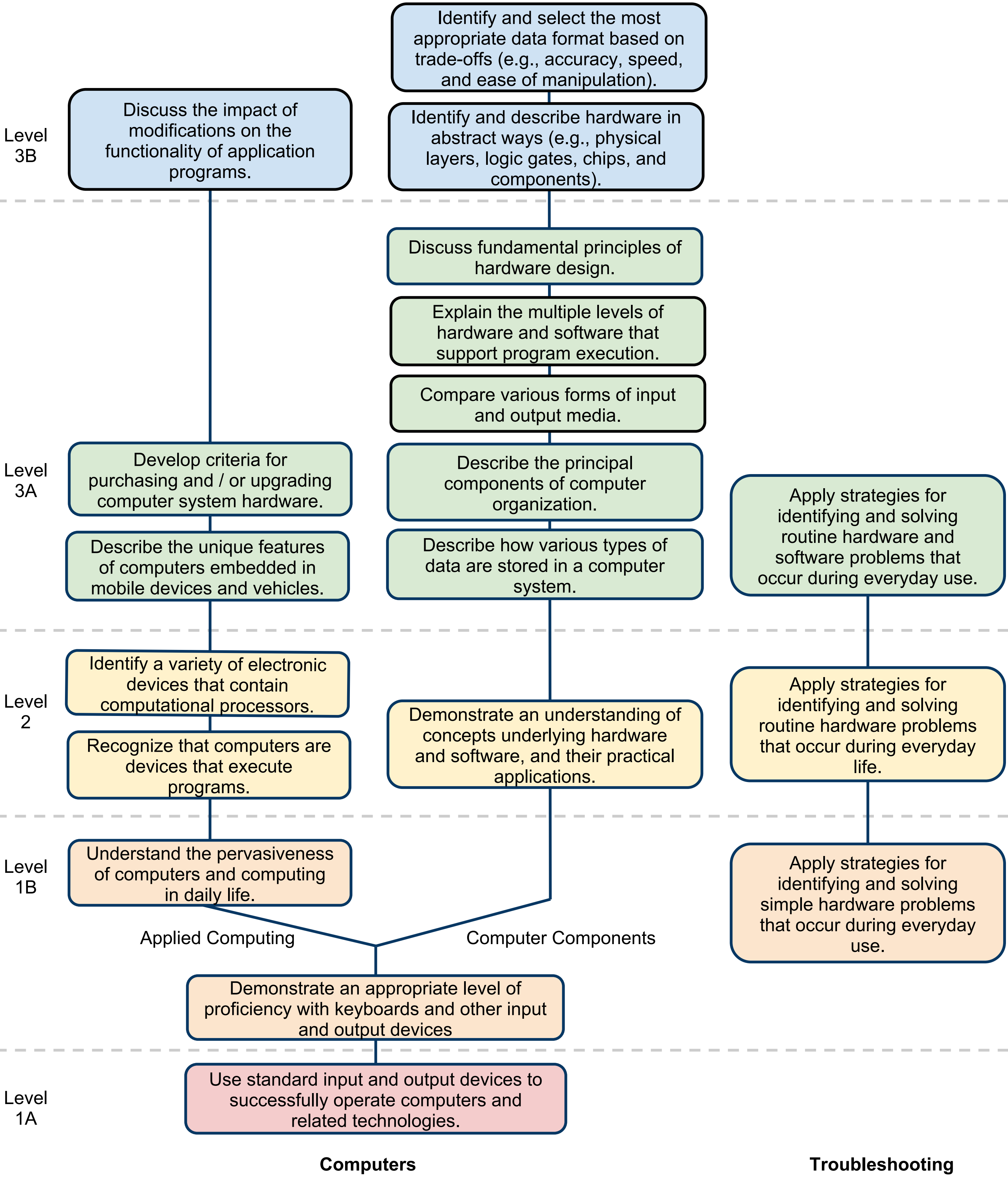
# Computing Practice and Programming



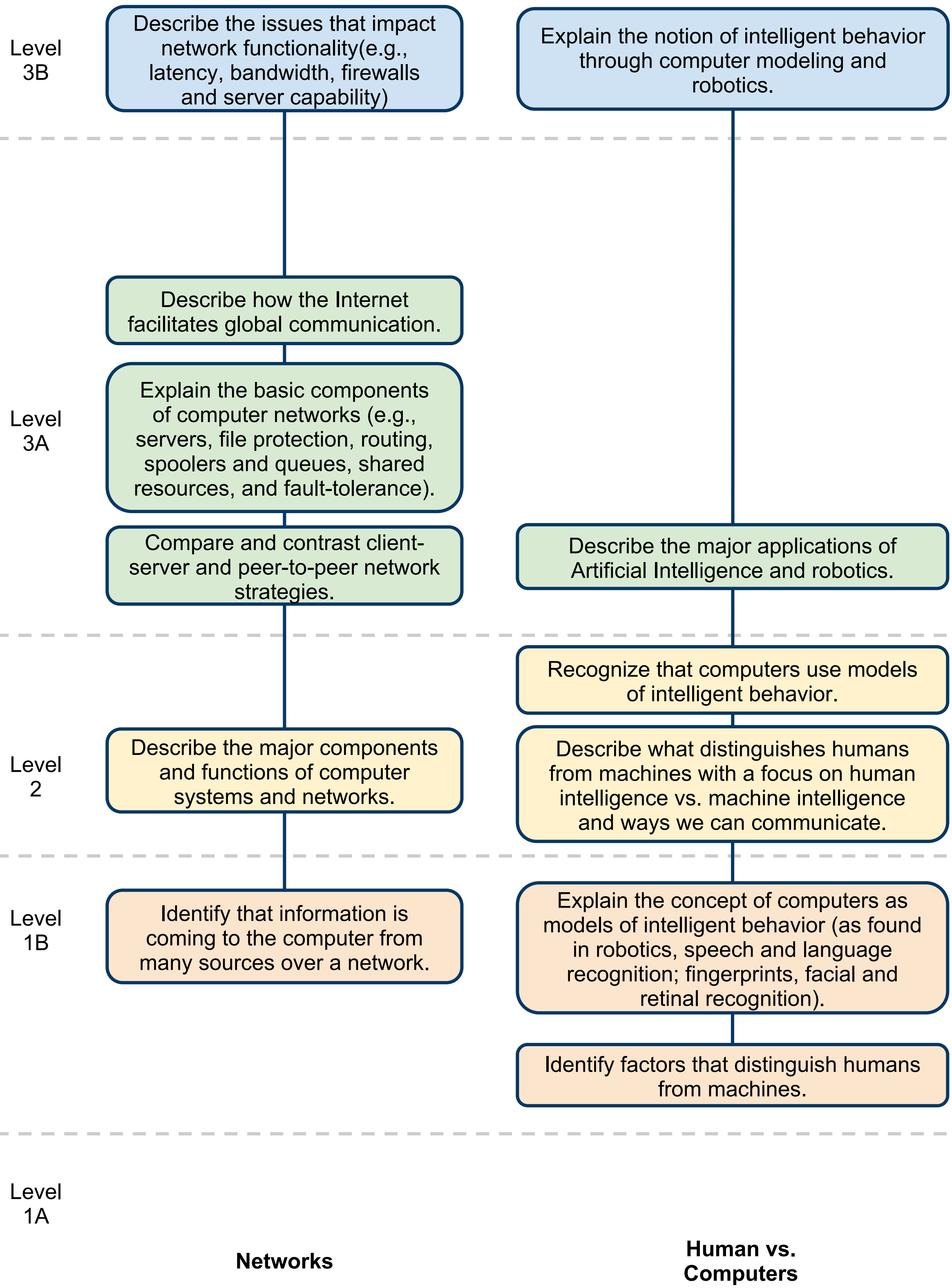
# Computing Practice and Programming



# Computers & Communication Devices



# Computers & Communication Devices



## Community, Global, and Ethical Impacts

| Level | Responsible use | Impacts of technology | Information accuracy |
| --- | --- | --- | --- |
| Level 3B | Demonstrate ethical use of modern communication media and devices. | Summarize how computation has revolutionized the way people build real and virtual organizations and infrastructures.<br>Summarize how financial markets, transactions, and predictions have been transformed by automation.<br>Analyze the beneficial and harmful effects of computing innovations. | |
| Level 3A | Compare appropriate and inappropriate social networking behaviors. | Describe the role that adaptive technology can play in the lives of people with special needs.<br>Discuss the impact of computer technology on business and commerce.<br>Compare the positive and negative impacts of technology on culture. | Describe a variety of strategies for determining the reliability of information found on the internet. |
| Level 2 | Exhibit legal and ethical behaviors when using information and technology and discuss the consequences of misuse. | Analyze the positive and negative impacts of technology on human culture.<br>Demonstrate knowledge of changes in information over time and the effects those changes may have on education, the workplace, and society. | Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and bias of electronic information sources concerning real-world problems. |
| Level 1B | Discuss basic issues related to responsible use of technology and information, and the consequences of inappropriate use. | Identify the impact of technology (e.g. social networking, cyberbullying, mobile and web technologies, cyber-security, and virtualization) on personal life and society. | Evaluate the accuracy, relevance, appropriateness, comprehensiveness, and biases that occur in electronic information sources. |
| Level 1A | Practice responsible digital citizenship in the use of technology systems and software. | | |

## Community, Global, and Ethical Impacts