The Big Ideas in Computer Science for K-12 Curricula

Tim Bell
University of Canterbury, NZ
tim.bell@canterbury.ac.nz

Paul Tymann
Rochester Institute of Technology, USA
paul.tymann@rit.edu

Amiram Yehudai
Tel Aviv University, Israel
amiramy@tau.ac.il

Abstract

When teaching computer science it can be easy to focus on details and lose sight of the bigger picture. This is particularly concerning with new pre-tertiary curricula being adopted in many countries as teachers grapple with a bewildering array of topics to teach. This paper steps back and introduces a list of ten big ideas of computer science that have been distilled based on input from curriculum designers and computer science education experts around the world. The big ideas are presented in a way that a classroom teacher will be able to engage with, so that they can use them to relate topics that they teach to the context of a bigger picture.

1 Introduction

As computer science appears as a subject in K-12 (pre-tertiary) curricula around the world, often from the first year of schooling (for example, [2, 3, 5, 6, 8]), it is important to articulate the big ideas of the subject to inform curriculum design, and more importantly, to enable teachers to understand the subject that they are being asked to deliver in the classroom. This is particularly important at the K-8 (elementary/primary) level where teachers are often generalists, and are being asked to incorporate ideas that are new to them into their classroom.

Focusing on a big picture view of a subject makes it easier for teachers to understand what the subject is about. It helps them to see how abstract ideas like algorithms, binary numbers and coding underpin things that deeply affect the modern digital world. Without a big picture view, teachers can perceive the material as an unwelcome imposition on their limited classroom time, and teachers who have not studied the subject previously can see the topics as a collection of esoteric facts and jargon. A big picture view also helps curriculum designers to make sure that it is not based on low-hanging fruit such as topics for which resources are already available for teaching them. For example, the plethora of introductory coding websites might give the impression that programming is the only topic of relevance in computer science.

Another concern amongst teachers as computer science appears in the curriculum is that things will change so fast that the curriculum will soon be out of date. A big picture view of a subject, one that focuses on the big ideas of the subject, paints a picture of a discipline with longevity, rather than something that needs to change every time a new technology develops. The big ideas for computing education should be stable. As Armoni points out "if complex and major changes [to a curriculum] seem unavoidable, the program probably emphasises technological or trendy aspects more than necessary, or the topics constituting its core should have been selected better at the start" [1]. These big ideas are intended to help curriculum designers to focus on core computer science topics, and to help teachers see the long-term value of the subject. By focusing on big ideas educators can distinguish relevant knowledge from skills. Skills (such as programming) are best built up over time, whereas foundational knowledge affects a student's view of the topic (such as how it is possible to search billions of items quickly, or that new systems need to be designed with security in mind from the start).

The big ideas presented in this paper are not meant to be general principles, discipline areas, or curriculum topics, but rather ideas that capture the essence of the discipline. They are not intended to cover every idea that comes up in the study of the topic (although they do have broad coverage, since curriculum content will lay the foundation for students to encounter these ideas). They should also have longevity, and not focus on a specific technology. There are already many overviews of what the subject of Computer Science is, but many are either directed at university level education (for example, [9]), or are somewhat abstract and hard to follow for a non-expert, or are more comprehensive than is appropriate for K-12 education.

The "Great principles of computing" by Peter J. Denning and Craig H. Martell [4], articulates six principles (communication, computation, recollection, coordination, evaluation and design) that are used as "windows" to view the space of computer science, and are presented as "cosmic" principles; that is, they should be ideas that will be true at all times in all parts of the universe. The approach used

by Denning and Martel provides a valuable new window on the subject, and is complementary to the approach presented here. In fact, Denning and Martell's principles are considered in several domains in the chapters of their book [4], and these expanded topics have elements of overlap with the big ideas presented in this paper.

Our goal here is to follow a format familiar to curriculum designers and teachers outside of computer science. The approach taken here was initiated from a request of a curriculum designer with no background in computer science, who asked for the equivalent of a paper called "Big ideas of science education" [7] that had the goal of identifying:

> ... the key ideas that students should encounter in their science education to enable them to understand, enjoy and marvel at the natural world.

Such a definition does not necessarily require an exhaustive list of everything covered by the topic, but is focused more on what a younger student should take away from their education, regardless of whether they will specialise in the subject, or use the knowledge to be a more informed citizen. This makes the focus more on concepts that are fundamental, but not obvious to the general public. For example, many people would assume that a program (algorithm) that is given twice as much data to process would take twice as long, whereas in reality it is sometimes much worse than this, and other times is much better. Computer science is full of paradoxes and surprises that provide opportunities for students to understand, enjoy and marvel at the digital world, and we are particularly interested in ensuring these are captured so that the ideas are in line with the intention of the "Big ideas of science education" [7].

Schwill considered how to develop a set of "fundamental ideas" in 1994 [10]. Our process and goal has much in common with Schwill's guidelines, which suggest considering the vertical and horizontal applicability of ideas (vertical means that they can be make sense to students at a variety of levels, and horizontal means that they have broad applicability), wanting the ideas to be relevant in the longer term, and ensuring that the ideas can be observed in everyday life.

Our work has focused on the subject of computer science. In K-12 curricula computer science may appear under headings such as Computational Thinking, Computing or Digital Technologies. While such curricula are likely to include topics outside of computer science (such as how to use particular types of software as a tool), our interest is in the parts of the topic that may not be so obvious because computer science has not previously been a subject in schools, and teachers and curriculum designers are unlikely to have studied it themselves, particularly those working at the grade school level. This contrasts with curriculum topics relating

to teaching *with* computers (e.g. e-learning), or teaching how to use computing devices, which is sometimes referred to as ICT [6].

We also note that the ideas don't necessarily reflect the weight that might be given to a topic. For example, programming is just one of our ten big ideas, but in many curricula considerable time is spent on this, partly because it is a skill that can require some time to acquire. Conversely, there will be other topics around learning to use or configure computers that appear in curricula but not the big ideas because students need to learn to do these things, even though they aren't necessarily a fundamental concept from computer science.

2 The Big Ideas

The ten big ideas presented in this paper start by looking at data and algorithms, then the idea of programming to enable the two to interact, then how humans are involved in the digital world, and concludes with wider ideas and challenges that come up when implementing real systems.

The description of each of the ten big ideas in this paper is limited to a two-paragraph summary that uses language that is intended to be accessible to an informed lay person. An expanded version is available online.[1] The online version describes in more detail the meaning and implications of each idea, and presents examples that are intended to make them more accessible to those without a strong background in the subject.

1. **Information is represented in digital form.**

   A huge variety of information is stored as data on digital devices, and shared between them; the data may be as simple as the number of steps counted on a fitness tracker, or as complex as the details of every transaction going through an international organisation; it includes text, images, video, sound and scientific readings. The remarkable thing is that all of this information is reduced to bits (binary digits), which are the fundamental element that makes digital devices so useful.

   Digital representations lead to versatile devices because the same hardware can be used for quite different purposes: a smartphone can play music, take photos, send email and show videos, because all these things are represented as bits, which are easily stored, copied, manipulated and transmitted on the same hardware. This is in contrast to non-digital (i.e., analogue) devices, which by nature are specialised (phones connect to a phone line, a TV gets a signal from a TV antenna, music is played from a vinyl disc, and video

[1] `http://www.cosc.canterbury.ac.nz/research/RG/CSE/big-ideas/`

is recorded on videotape). Digital data can also be shared without loss of quality, whereas analogue devices tend to reduce the quality if the material is copied or re-transmitted.

2. **Algorithms interact with data to solve computational problems.**

   An algorithm is a well-defined process that acts on data to solve some problem, for example, finding the shortest route on a map, matching two strands of DNA, or changing the brightness of a photo.

   An algorithm can only include steps that a conventional computer could do; for example, you couldn't just put in a step that says "find the most efficient solution". Remarkably, the full power of a conventional digital device can be realised by an algorithm using just three structures to control program flow: sequencing (putting instructions one after the other), selection (choosing which part of the algorithm to execute based on some values, usually using an "if" statement), and iteration (repeating part of the algorithm with a loop). Apart from these three basic types of instruction, a computer is able to read in information (input), give out information (output) and store data to use later on. These basic components, or their equivalents, can be used to define every algorithm, as they define exactly what can, and cannot, be done on conventional devices.

3. **The performance of algorithms can be modelled and evaluated**

   The main resources that an algorithm uses are time and space (memory). Time is a key factor because slow programs are annoying to users, and if a program is going to take decades to complete a calculation, it is better to determine this before implementing it! Using an unnecessarily inefficient algorithm might lead to devices wasting power, needing cooling, or running batteries flat in mobile devices. Some algorithms also need a lot of spare memory or storage while they are running. This may make the algorithm infeasible in some cases, while in other cases it might be an excellent trade-off if the algorithm is faster.

   The time taken by an algorithm is usually estimated based on the size of the input (such as the number of items being searched through, the number of streets in a map, or the number of pixels in an image). In some cases, when the input size is doubled, the algorithm takes twice as much time (we call this a linear time algorithm). But for some algorithms the time grows much faster than the size of the input, while for others it grows much slower. It is important to at least estimate the time it will take an algorithm to solve a problem before implementing it, as it might be very sensitive to the size of

the input; perhaps a program works satisfactorily in tests, but with a larger input it might take a lot longer.

4. **Some computational problems cannot be solved by algorithms.**

   There are some computational problems that we can prove will never have programs written to solve them (these problems are not computable). For example, it can be proved that no one will ever be able to write a general app that can determine whether or not another app will freeze your smartphone (this is more formally known as the halting problem).

   In addition to non-computable problems, there are many practical problems for which all known algorithms to find the optimal solution are "intractable," which means that no machine currently exists that has the resources required to execute the algorithm once the size of the input gets fairly large. For these problems we need to consider algorithms that find an approximate solution rather than pursue optimal solutions that could potentially take billions of years to evaluate even on the fastest computer. Some problems have mathematical proofs that they are intractable, but there are many problems for which an algorithm has not been found that runs in a reasonable amount of time, despite decades of research; yet we also have not proved that the algorithm cannot exist. Resolving this issue is widely regarded as one of the biggest questions in computer science!

5. **Programs express algorithms and data in a form that can be implemented on a computer.**

   Programming involves taking algorithms (which might exist only in the programmer's head, or may have been designed by a team of people) and turning them into program instructions that can be executed by a computer. Program instructions are written in a programming language which is precisely defined. These instructions manipulate data on the computer, so the form and meaning of the data is dictated by the program. Programming languages let the programmer represent complex data as various data structures that allow for efficient access and manipulation by the program.

   Because the capabilities needed to fully control a general-purpose computer (which covers most digital devices) can be defined by six properties, consisting of three control structures (often expressed as sequence, selection, iteration), and three ways to deal with data (input, output and storage), these properties (or their equivalent) are also the key elements of writing programs. Consequently, any programming language that has all of these elements can be used to write any computation that any other full programming language could be used for, and the differences between languages

is largely to do with how well they suit a particular situation (e.g. for processing files, running on a smartphone, teaching programming, or running an enterprise system), or how suitable their built-in capabilities are for the domain.

6. **Digital systems are designed by humans to serve human needs.**

   This is the driver for all the ideas above; digital devices must be fast, reliable and match a human need appropriately if people are to use them, and because they are designed by people, the process for designing them needs to enable the developers to efficiently turn creative ideas into working products.

   This means that there are (at least) three broad areas concerned with human factors: creating interfaces that are easy to use in the situation they are intended for, developing software on a large scale (potentially with thousands of people contributing to it), and making sure that the product meets the needs of the user (the product is reliable, does what is intended, and is completed in a timely fashion). It is important that both users and developers of digital systems, understand the impact of technology on humans, the responsibilities of those who work on it, and possibly even whether or not the systems should be constructed. All of these concerns require some understanding of human behaviour (psychology), interaction (sociology), and capability (physiology).

7. **Digital systems create virtual representations of natural and artificial phenomena.**

   Computer simulations and virtual systems are used to create virtual versions of processes in the physical world, and also to create imagined scenarios. Simulations can be used to reduce cost (e.g. simulating a physical structure to fine tune it before building it, or simulating different financial scenarios to choose the most effective strategy) and to reduce risk (e.g. simulating dangerous situations to give aircraft pilots experience, or simulating the spread of a disease to determine how best to prepare for an epidemic). Virtual worlds can be created by generating images and sounds artificially, to make the user feel that they are in a world imagined by the designer (including computer games, virtual reality and augmented reality). Virtual machines provide a simulation of a computer, and this approach is widely used because it protects the physical hardware and the software from each other, which can provide a safer and more flexible environment. Artificial Intelligence uses computational techniques to make the kind of decisions that are normally attributed to human intelligence.

These virtual representations take advantage of the unique properties of digital devices that enable computing systems to work on vast amounts of data, and if something goes wrong, they can be re-started in full working order by simply restoring some data, which is considerably simpler than reinstating physical objects that have been damaged or placing humans in a dangerous situation.

8. **Protecting data and system resources is critical in digital systems.**

   Modern computing systems provide access to data and resources that if used inappropriately could breach privacy, provide unauthorised access to financial data or other resources, or even bring about physical harm. Security professionals often say that the weakest link in the security of a computer system is the user, and so it is essential that all computer users understand basic computer security principles. These principles include confidentiality (not allowing unauthorised access to information), availability (legitimate users can access their information), and integrity (the information is accurate).

   Everyone in computing needs to be aware of and understand the tools and techniques that they can use to make their computing environment more secure. These tools include encryption methods, detecting and blocking attacks, authenticating who is accessing a system, and allowing users to recover from damage, whether malicious or accidental.

9. **Time dependent operations in digital systems must be coordinated.**

   Digital systems have many components that can run independently; these components can be working in parallel, and on independent schedules. Parallelism occurs at many levels in digital systems, from instruction execution on a CPU, to multi-core systems in a laptop, to data being transmitted over a network through multiple routes, to large *big data* systems that process vast quantities of data in small chunks and combine the results.

   When a computational task is being spread over several independent parts of the system, considerable care is needed to make the most of the ability to spread the work over multiple devices. Problems need to be broken up into as many parts as possible that can be processed independently and recombined, and the dependency between these operations can restrict how easily a problem can be broken into parts.

10. **Digital systems communicate with each other using protocols.**

    Very few digital devices are an island — most are connected by wired or wireless networks. The goal is to get data through the networks as quickly

as possible while being resource efficient. Networks are prone to errors from faulty components or transmission interference, and are also vulnerable to attack from people wanting to eavesdrop on the data or prevent it getting through.

Techniques are available to minimise these issues, to the extent that people use wireless data and the Internet for sending important and private information without being overly concerned about reliability and security. Protocols that ensure that the data has arrived safely and efficiently are essential for almost any situation: personal communications, commercial transactions, or military control all need to be sure that the data gets through reliably.

3 Conclusion

The big ideas presented above serve to introduce those who are involved with the design, development, implementation and delivery of computer science curricula with the wide range of individual topics in the subject. Understanding what the major landmarks are in the computing landscape will help individuals to see the bigger picture.

Clearly, a list such as the one presented in this paper is subjective, but to maintain objectivity the list was developed by discussing it in detail with a number of computer science education colleagues, at both the higher education and K-12 school level. Early versions were created by showing colleagues the equivalent list for science, and asking what they thought should be on a list for computer science. This input was reviewed for commonality, and synthesised into a single list. In order to ensure that the has been articulated in a way that is meaningful to teachers the list has been shared with teachers who were relatively new to the subject. The big ideas presented in this paper are the result of several such reviews.

We believe that we have converged on a list that incorporates practically all of the ideas that our colleagues regard as the key ideas in the discipline, and that they have been articulated in a meaningful way. We also believe that the list can continue to be improved by continuing to solicit and incorporate feedback from our colleagues. The list above is only an introductory articulation of the big ideas. An online version expands each idea into examples that are intended to be meaningful to teachers and non-specialists[2]. Having the online lists also allows more detail to be added if necessary to include important ideas that may not have been captured in our process, and to keep the list relevant to new developments in the field.

To this end, we welcome feedback from readers of this article. The list to date

[2] `http://www.cosc.canterbury.ac.nz/research/RG/CSE/big-ideas/`

has been built up by consulting many members of the CS community around the world, but we are particularly interested to hear if there are *big* ideas from Computer Science relevant to K-12 curricula that we may have overlooked through the process, if there are better ways to express the ideas that highlight their fundamental nature, or if there are compelling examples that will help non-specialists to "understand, enjoy and marvel" at the ideas behind our digital world. Readers are invited to contact the authors with suggestions.

Acknowledgements

We are greatly indebted to a number of colleagues who spent some time commenting on the ideas, including attendees at the 2016 CSMC workshop and the 2016 combined ISSEP/WIPSCE conference. We would particularly like to acknowledge input from Ira Diethelm, Mike Fellows, Jens Gallenbacher, Juraj Hromkovič, Tobias Kohn, Wiebke Kothe, and Regula Lacher.

References

[1] Michal Armoni. Computing K-12 curricular updates. *ACM Inroads*, 4(4):20–21, 2013.

[2] Tim Bell, Peter Andreae, and Anthony Robins. A case study of the Introduction of Computer Science in NZ schools. *ACM Transactions on Computing Education (TOCE)*, 14(10):10:1–10:31, 2014.

[3] Neil C C Brown, Sue Sentance, Tom Crick, and Simon Humphreys. Restart: The Resurgence of Computer Science in UK Schools. *Trans. Comput. Educ.*, 14(2):9:1–9:22, jun 2014.

[4] Peter J. Denning and Craig H. Martell. *Great Principles of Computing*. MIT Press, 2015.

[5] Katrina Falkner, Rebecca Vivian, and Nickolas Falkner. The Australian Digital Technologies Curriculum: Challenge and Opportunity. *Proc. Sixteenth Australasian Computing Education Conference (ACE2014)*, pages 3–12, 2014.

[6] Steve Furber, editor. *Shut down or restart? The way forward for computing in UK schools*. The Royal Society, London, 2012.

[7] Wynne Harlen, Derek Bell, Rosa Devés, Hubert Dyasi, Guillermo Fernández, De Garza, and Pierre Léna. *Big Ideas of Science Education*. Science Education Programme (SEP) of IAP, 2015.

[8] Richard Kick and Frances P Trees. AP CS Principles: Engaging, Challenging, and Rewarding. *ACM Inroads*, 6(1):42–45, feb 2015.

[9] Mehran Sahami, Steve Roach, Ernesto Cuadros-Vargas, and Richard LeBlanc. ACM/IEEE-CS computer science curriculum 2013: reviewing the Ironman report. In *Proceeding of the 44th ACM technical symposium on Computer science education*, SIGCSE '13, pages 13–14, New York, NY, USA, 2013. ACM.

[10] Andreas Schwill. Fundamental ideas of computer science. *Bulletin – European Association for Theoretical Computer Science*, 53:274, 1994.