tecnologias, sociedade e conhecimento
vol. 4, n. 1, Dezembro/2017

A half-century perspective on Computational Thinking[1]

Ken Kahn (University of Oxford)[2]

Abstract
For more than fifty years people have been exploring how computers might enhance learning and teaching. The malleable nature of computers has enabled suggestions that a computer can act like flash cards, personal tutors, textbooks, reference books, virtual laboratories, quizzes, virtual spaces, lecture halls, and study groups. Perhaps the most radical suggestion has been to see the computer as something learners can creatively mold into something personally meaningful that is dynamic, interactive, and shared. And that the process of constructing such computational artefacts is rich in learning opportunities. These range from a deeper understanding of the subject matter of the constructions to high-level skills in thinking and problem solving. In 2006, Jeanette Wing published an essay on computational thinking that led to over a dozen books and over a thousand articles. It has strongly influenced the national curriculum of many countries. This article addresses the question of how the concepts underlying computational thinking fit into over fifty years of explorations of the role of computers in learning.

Keywords. Computational thinking; Constructionism; Technology-enhanced learning; Seymour Papert; History of computers; Education.

Resumo
Há mais de cinquenta anos que as pessoas exploram como os computadores podem melhorar a aprendizagem e o ensino. A natureza maleável dos computadores permitiu que ele funcione como cartão de memória, tutor pessoal, livro didático, livro de referência, laboratório virtual, questionário, espaço virtual, sala de conferências e grupos de estudo. Talvez a sugestão mais radical seja conceber o computador como algo que aprendizes podem moldar de forma criativa resultando em algo realmente significativo, dinâmico, interativo e compartilhado. E, ainda, que o processo de construção desses artefatos computacionais seja rico em oportunidades de aprendizagem. Estas oportunidades variam desde ganhar uma compreensão mais profunda do assunto até a construir habilidades de alto nível de pensamento e de resolução de problemas. Em 2006, Jeanette Wing publicou um ensaio sobre o pensamento computacional que instigou a publicação de mais de uma

[1] This paper is an extended and revised version of Ken Kahn, A half-century perspective on the role of computers in learning and teaching in Music Learning with Massive Open Online Courses edited by Luc Steels

[2] Contact: Kenneth.kahn@it.ox.ac.uk

KAHN, K. A half-century perspective on Computational Thinking. **Tecnologias Sociedade e Conhecimento**, Campinas, v. 4, Dez. 2017. Disponível em: <http://www.nied.unicamp.br/ojs/>.

dúzia de livros e mais de mil artigos. Esse ensaio influenciou fortemente o currículo nacional de muitos países. Este artigo aborda a questão de como os conceitos subjacentes ao pensamento computacional se encaixam em mais de cinquenta anos de exploração a respeito do papel dos computadores na aprendizagem.

Palavras-chave. Pensamento computacional; Construcionismo; Aprendizagem aprimorada pela tecnológica; Seymour Papert; História dos computadores; Educação.

1. Introduction

The idea that computers can play an important role in learning and teaching is over fifty years old. This paper describes the history of attempts to use computers to support and enhance learning from a personal perspective. Instead of a complete history it attempts to highlight groundbreaking and significant ideas and computer systems that have led to today's efforts to provide technology-enhanced learning. Some systems use the computer to emulate older paper-based technologies. Others attempt to give the computer the role of teacher or tutor. The systems that are described in the most detail here are those that attempt to use the computer to provide novel learning experiences that were impossible or impractical before.

The 1950s through the 1970s were dominated by "computer-aided instruction" systems that attempted to teach in a very didactic and mechanical manner. These were based upon behaviorist theories of learning. Research laboratories at MIT, Xerox PARC, and the University of Edinburgh were exploring a very different approach. Instead of the computer programming the student, the student was given tools for programming the computer. Creativity and exploration were emphasized. Early attempts at computer tutoring systems were made. Programming languages designed specifically for learners were developed.

The 1980s saw the wide-spread dissemination of personal computers and programming languages for children. There were efforts to enhance these languages with new ideas from computer science. Media creation was combined with program creation. Intelligent tutoring systems were demonstrated to work well for a limited number of topics.

The 1990s saw the wide-spread use of "multi-media" to enhance education. Programming languages for children expanded into new territories. Learners were supported in building computer games and programming robots.

The 2000s saw the integration of the web into educational software. Learners were connected by the World Wide Web and able to easily share their constructions. Multi-user three-dimensional virtual spaces became popular places to explore their potential to enhance learning. Many explored the benefits of each learner having their own personal computer.

The 2010s saw the introduction of MOOCs (massive open online courses) and web-based programming environments. Programming environments have been developed data support students making apps or robots that integrate large online databases. Others are integrating AI cloud services and machine learning in student programming tools. This

decade also sees educational software being adapted for tablets and smartphones with touch interfaces.

2. 1950s and 60s

Alan Perlis saw the potential of computer programming for learning by science, mathematics, and engineering students in the mid-1950s. He began teaching the first freshman course on computer programming in 1958. In 1961 in a lecture at MIT he said “The purpose of a course in programming is to teach people how to construct and analyze processes” (Greenberger, 1962). J.C.R. Licklider commented “... I see computer programming as a way into the structure of ideas and into the understanding of intellectual processes that is just a new thing in this world”.

In 1964 Kemeny and Kurtz introduced the Basic programming language, the first programming language designed for learners and beginners (Dartmouth College Computation Center, 1964). It contained many comprises due to the hardware limitations of the day. Variable names, for example, were limited to one letter followed by digits. While initially limited to use in universities, Basic became very popular with schools and hobbyists in the 1970s and 80s.

In 1967 Seymour Papert, Wally Feurzeig, Cynthia Solomon, and Danny Bobrow developed the Logo programming language. Unlike Basic, which was designed to provide the minimal language that can support student programming, Logo was designed to be a rich and powerful language. Logo is the result of “child-engineering” (in other words redesign to improve usability by children) the best ideas in computer science at the time. It borrowed very heavily from the Lisp programming language which was being used by artificial intelligence researchers. Logo was conceived of as both a tool for learners to use to express themselves creatively and an “object to think with” (Papert,1980). Initially the projects created using Logo focused upon word and list processing and mathematics. For example, children constructed programs that generated poetry. By 1969 Logo was enhanced to control “floor turtles”, robots that could be commanded to move forward or turn. This became the basis of the very successful turtle graphics when “screen turtles” were introduced in 1972.

A very different trend that began in 1960 is “computer-aided instruction”. This was pioneered by the Plato system (Plato History, 2017). The Plato system initially focused on presenting multiple-choice or numeric questions and automated responses. Its initial innovations were in computer graphics and display terminals that it pioneered. The Plato system grew with time to include interactive simulations, educational games, and discussion forums. But unlike the efforts around Basic and Logo, Plato was based upon a didactic

teaching method instead of a programming languages' support of learner-centered problem solving and creativity.

Figure 1 - The PLATO system

Source: Author

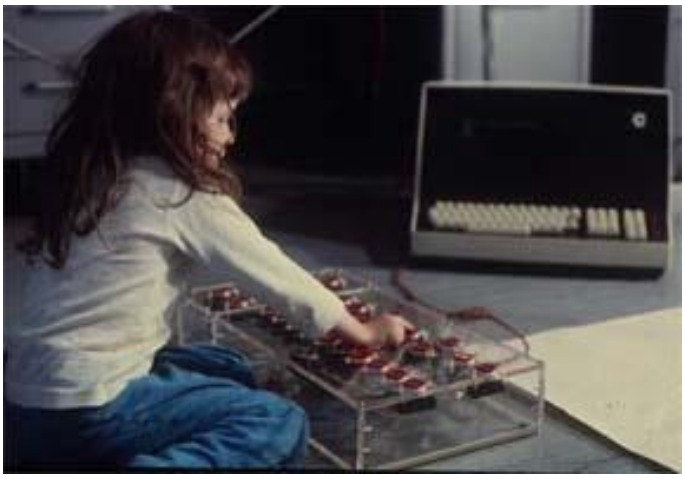
The idea of using computers in education was very radical in a period where computers were few and very expensive. As Hal Abelson, one of the earlier pioneers of Logo programming, said "You really have to try hard to get into the mindset of that time, because a computer in those days was something that cost several million dollars. And the idea that you would take the most advanced computing research equipment around anywhere, and you would let fifth graders … start playing with it, it was just mind boggling. For the first 10 years of that, people just thought we were nuts" (Hardesty, 2010).

3. 1970s

The next decade saw substantial progress in efforts around the Logo programming language. Since the center of this research was the MIT Artificial Intelligence Laboratory it is perhaps not surprising that many efforts attempted to connect Logo and AI. Gerry Sussman (1973) and Ira Goldstein (1974) produced systems that helped debug and teach Logo. Danny Hillis wrote about AI projects that children could do in Logo. Radia Perlman developed

special hardware to provide interfaces appropriate for very young children to construct Logo-like programs (Morgado; Cruz; Kahn, 2006).

Figure 2 - Radia Perlman's Button Box for Preschoolers



Source: Author

This was the decade when the concept of object-oriented programming was incorporated into programming languages for children. Smalltalk 72 and 76 were designed for children and inspired by Logo. (Smalltalk 80, however, was developed as a tool for professional programmers.) Director was another object-oriented language for children that was designed to support the programming of animation (Kahn, 1979).

During the 1970s some versions of Logo were created to support the programming of music, color graphics, three-dimensional graphics, and animations. Implementations of Logo appeared on computers inexpensive enough for schools to acquire and the use of Logo by students expanded beyond the laboratory by the end of the decade.

Researchers on intelligent tutoring systems made substantial progress this decade. A notable example is Buggy (Brown; VanLehn, 1980), which was able to diagnosis students' arithmetic mistakes and respond appropriately.

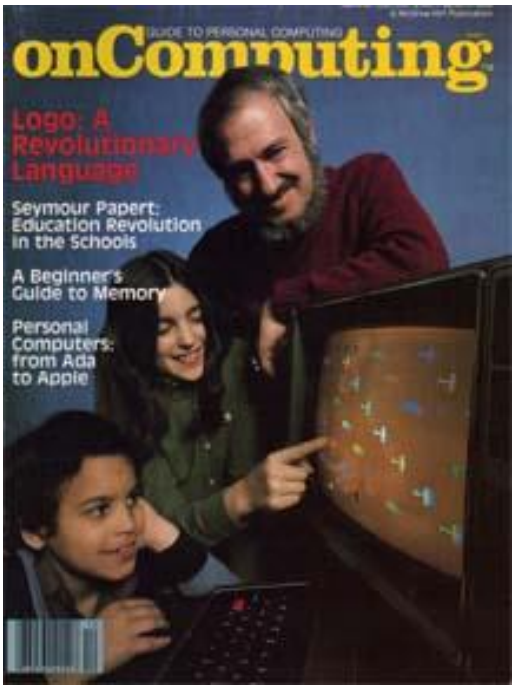
Research on the use of computer games for learning began in this decade as well. Games were developed for educational purposes and researchers explored the educational value of games designed for entertainment purposes (White, 1981; Malone, 1981). The first computer game, MIT Space War, created in 1961, attempted to have accurate positioning of stars and simulation of gravity and hence could be argued to be "educational". Seymour Papert later argued that more serious learning can result from challenging entertainment games than with many "edutainment" games that attempt to be both educational and

entertaining (Papert,1998). Educational games and educational uses of commercial games has continued to be an active area of development and research for nearly fifty years.

4. 1980s

With the spread of relatively inexpensive personal computers, programming languages for children became widespread in schools and the home. This, combined with Seymour Papert's very influential 1980 book, Mindstorms: Children, Computers, and Powerful Ideas, led to an explosion of activities around Logo. Many schools in the US required its teaching. It became part of the UK National Curriculum in 1988. Far too often, however, the spirit of Logo was lost, and children were taught Logo in a way that was far from the creative, exploratory, reflective style it was designed for.

Figure 3 - Logo becomes mainstream



Source: Author

Abelson and diSessa wrote Turtle Geometry: The Computer as a Medium for Exploring Mathematics, a book that explores how advanced mathematics could be explored and taught building upon the turtle geometry of Logo (Abelson; DiSessa, 1981). While this undoubtedly helped counter the misconception that Logo was only for primary school children, it was commonly held that Logo was too childish for use by older students. A three-volume book, Computer Science Logo Style by Brian Harvey (1997), was aimed at high school teaching and was partially successful in countering this. The misconception that Logo

is childish is ironic given that Logo was based upon Lisp, an advanced AI programming language with very powerful primitives for dealing with symbolic information.

This decade saw a flourishing of experimental variants of Logo and other rogramming languages for children. Object Logo (Drescher, 1986) was an object-oriented programming language that contained classical Logo as a sub-language. Multi-Logo (Resnick, 1990) explored Logo running in multiple processes. Boxer (DiSessa, 1997) tightly integrated a powerful Logo dialect with a sophisticated user interface. Efforts were made to take other artificial intelligence languages and adapt them for use by school children (Kahn, 1984; Ennal, 1982).

Intelligent tutoring systems made strong advances but only in a few select subjects such as teaching algebra, geometry, or computer programming (Anderson, et al. 1990).

5. 1990s

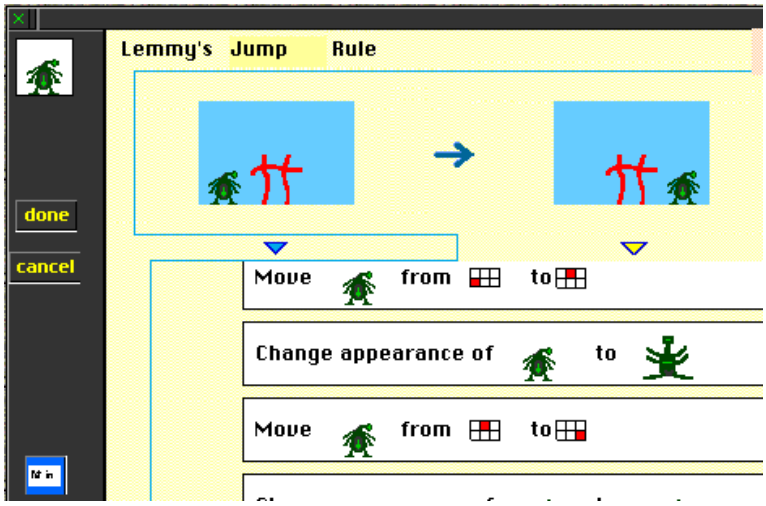
The 1990s saw a good deal of activity around adding concurrency and visual syntaxes to programming languages for children. One of the drivers towards concurrency was agent-based modeling. The idea is that one can learn about complex systems by constructing, observing, and experimenting with simulations of interacting entities. This began with StarLogo (Resnick,1994) to be followed by NetLogo (Wilensky, 1999) and Agentsheets (Repenning, 1991). These efforts to introduce agent-based modeling to school children were described in Mitchel Resnick's book Turtles, Termites, and Traffic Jams (Resnick, 1994). By using these tools, students could acquire a deeper understanding of the underlying processes in scientific phenomena. Topics include those in the physical, biological, and social sciences as well as the humanities including history, philosophy, and language. The educational value of computer programming expanded by providing new ways of learning most school subjects.

Concurrency appeared in other programming languages for children. Stagecast Creator (Smith; Cypher; Spohrer, 1994) was based upon concurrent rewrite rules. ToonTalk (Kahn,1995) followed the design philosophy of Logo to child-engineer the best computer science programming language ideas. Three decades after Logo's design borrowed from Lisp, ToonTalk's design built upon the ideas of concurrent constraint programming (Saraswat, 1993). All of these languages supported programs with multiple simultaneous activities, but only ToonTalk provides general mechanisms for communication and coordination between multiple processes.

The other major trend in the 1990s was to explore graphical syntaxes for programming languages. Agentsheets and Stagecast Creator (at first called KidSim) supported expressing programs as graphical rewrite rules. For example, here is how one

expressed in KidSim that a character should jump over obstacles (Smith; Cypher; Spohrer, 1994):

Figure 4 - A KidSim rule for jumping over fences



Source: Author

These graphical rewrite rules are intuitive and surprisingly expressive but support abstract rules poorly. Agentsheets addresses this by combining graphical rewrite rules with a spreadsheet metaphor and a scripting language for advanced users.

Agentsheets and Stagecast Creator/KidSim also supported program construction by demonstration. ToonTalk took this to the extreme: the only way to construct programs was via demonstration followed by removal of details to obtain abstraction (Kahn, 2000). ToonTalk has no static syntax; programs are created and viewed as animations in a game-like environment. Programming by demonstration in ToonTalk can be successfully performed by preschoolers (Morgado; Cruz; Kahn, 2003). The lack of a static syntax does interfere with scanning and editing programs however. Unlike other programming languages for children, ToonTalk programs can be completely text-free, making them particularly suitable for pre-literate children and internationalization.

In 1996 LogoBlocks (Begel, 1996) pioneered a graphical syntax that subsequently became hugely popular. It introduced shaped blocks that can be dragged and dropped to assemble programs. These blocks correspond to program commands, expressions, data, and control structures. Palettes of blocks enable users to construct programs by selecting the needed parts. Most importantly, these blocks snapped together only when the parts fit together like a jigsaw puzzle. Syntax mistakes are not expressible in such a system. Unlike textual programming languages, the user doesn't need to remember what primitives are

available, but instead can select them from palettes. In the next decade the syntactic ideas of LogoBlocks were integrated with StarLogo TNG (Klopfer et al., 2009), Scratch (Resnick, et al., 2009) Snap! (Harvey; Mönig, 2010), MIT App Inventor (Wolber et al., 2011) and many more (Blockly, 2017).

Another programming language trend of the 1990s was to support robot construction kits. A pioneering example of this was LEGO/Logo (Resnick; Ocko; Papert, 1988).

Idit Harel (1991) and later Yasmin Kafai (1995) explored the idea of children programming educational games for younger children. Children using the Logo programming language designed and implemented games to teach concepts about fractions to younger children. Kafai created a sustainable school culture consisting of three grade levels. The oldest children built the games for the youngest children with assistance from middle children who the next year became the game makers. Studies demonstrated that the children who designed and constructed educational games learned the subject matter of their games very well even if the games themselves were not particularly pedagogically effective for the younger students.

The 1990s also saw the rise of multi-media CD-ROMs. For example, Microsoft's Encarta encyclopedia included much that paper alternatives lacked, including audio, animations, videos, and interactive applications. Subjects could be connected by hyperlinks. Novel interactive books on CD-ROMs where illustrations were animated and reacted to clicks became very popular (Wikipedia, 2017a). So-called "edutainment" games appeared on many CD-ROMs in the 90s.

6. 2000s

The most interesting developments in the first decade of the 21st century were creative and game-changing uses of the Internet. Two early examples of this were the Playground Project (Hoyles; Noss; Adamson, 2002) and WebLabs project (Mor et al.,2004) both large-scale multi-country European projects. Playground was focused on very young children authoring and sharing computer games. The games were exchanged by students in different countries via email enhanced by video conferences. Games were constructed that enabled players in different countries to play together. WebLabs supported children in exploring mathematics and science computationally *and* sharing and discussing their discoveries in web reports. This added extra dimensions to their learning. In publishing on the web students reflected deeply about what they discovered and worked hard to communicate it effectively. The discussions attached to each report often contained constructive criticism and suggestions. The students were not only doing science and exploring mathematics by

constructing computer programs, but were also engaged in the process of academic publication to peers.

The web and increasingly capable web browsers enabled the Modelling4All project (Kahn; Noble, 2010) to build a web-based tool (the *Behaviour Composer*) to support teaching, research, and public engagement with agent-based modelling (ABM). By building on the popular open-source NetLogo agent-based modelling system, the project was able to focus upon higher-level issues of enabling a range of users, including those with no programming experience, to produce open, modular, transparent, sharable models. The Behaviour Composer is web-based both in the sense that one can construct and run models from a modern web browser as well as in supporting sharing models, model components, and interactive tutorials as public web pages.

A major event of this decade was the emergence of the Scratch programming language from MIT. It became very popular after launching its website in 2007. Ten years later, almost 25 million projects have been shared on the website, over 20 million users registered, and over 125 million comments posted. Users, mostly between 8 and 14 years old, support and learn from each other. About 30% of projects are “remixes” where someone makes a variant of another’s project (with attribution maintained) (Kahn; Noble, 2010). As discussed earlier, Scratch’s syntax contributes significantly to its popularity. The website provides support, motivation, millions of sample projects, and a sense of community that accounts for the popularity of Scratch (MIT Media Lab, 2017).

A different trend in the 2000s is exemplified by *Second Life*, a communal three-dimensional virtual world, that became very popular. Thousands of avatars controlled by their “owners” interact in this virtual world. “Residents” of Second Life can earn virtual money, build virtual objects, buildings and spaces, and communicate with other residents. A teen-only *Teen Second Life* was launched in 2005. Educators saw this as potentially a new and effective place for teaching and learning. Many museums opened up Second Life “branches” that exploited the unique capabilities of this virtual world. For example, the US Air and Space Museum built replicas of rockets that visitors could enter and launch. Schools and universities also opened locations. Some uses were recreations of ordinary lecture-oriented teaching while others explored new possibilities. For example, Dr. Peter Yellowlees created virtual hallucinations based upon the experiences of schizophrenia patients. Visitors could experience first-hand what it’s like to have schizophrenia (BBC News, 2016).

Figure 5 - The US Air and Space Museum in Second Life.

Source: Author

Due to the appearance of inexpensive micro-controllers in the beginning of the century, educational robotics kits evolved from being cabled to a controlling personal computer to running programs inside the robot itself. Lego's Mindstorms (inspired by Seymour Papert's book of the same name from 1980) became popular. Robot behaviors were still programmed on personal computers, but once downloaded into a micro-controller, they became autonomous. Students used these kits to make a wide range of interactive gadgets. The Lego Group offered *RoboLab*, a graphical dataflow language, to schools using Mindstorms. Researchers implemented dozens of other languages for controlling Mindstorms bricks.

2006 saw the launch of the One Laptop per Child project by MIT Professor Nicholas Negroponte (One Laptop per Child, 2017). The dream was to support the dissemination of inexpensive laptops to every child in the developing world. Special hardware and software was developed. The laptops were designed to have very low power requirements so that electricity could be provided by other means if electrical power wasn't available. The laptops can easily be connected in a network to share resources and support multi-user applications. Over two million laptops were produced and in a few countries, there were enough to provide a laptop to each child (Uruguay for example). Two million is a significant number, but many fewer than the hundreds of millions initially expected.

Figure 6 - One Laptop per Child laptops in West Papua

Source: Author

7. 2010s

By 2010 web-based technology (JavaScript, CSS, and HTML5) began to be mature enough that serious programming environments could be built to run in any modern browser, including those on tablets and smartphones. Implementations of Logo, ToonTalk, and dialects of Scratch appeared that ran immediately in a browser without any installation or plugins. Programs could be stored seamlessly to cloud storage so that students could move easily between school, home, and libraries as they constructed computational artefacts.

An example is *Snap!*, a more powerful variant of Scratch, implemented as a web application (Harvey; Mönig, 2010). It contains new primitives for supporting first-class functions (functions that can create or use other functions) and lists. Unlike Scratch, it is suitable for an advanced high school or beginning university computer science course. It illustrates a tension between programming languages designed to be easy to learn, such as Scratch, and those designed to support more advanced computational concepts and the construction of larger, more complex programs. A curriculum called the Beauty and Joy of Computing (Kahn, 2014) which includes text books, MOOCs, and lesson plans is based upon Snap!.

ToonTalk was built as a Microsoft Windows application. *ToonTalk Reborn* is a reimplementation and redesign for the web (University of California, 2017). ToonTalk programs can be associated with any browser element, giving them interactivity. Widgets

constructed in ToonTalk can be embedded inside web pages. Programs and widgets can be dragged between browsers. Programs can be published as automatically generated web pages surrounded by editable rich text.

The Khan Academy (2017a) began in 2009 as an online mathematics learning site relying heavily on short videos. It has delivered over 600 million lessons in many school subjects to over 55 million students. It delivers 4 million exercise problems daily. Many teachers use it to “flip the classroom” where watching videos as homework replaces classroom lectures. This frees up classroom time for personal support of students as they attempt to do exercises.

This decade has also seen the rise in online tutorials and puzzles designed to teach programming. Code.org has promoted the “Hour of Code” which has reached more than 400 million people. 600,000 teachers use the online programming courses on the web site. A very impressive online programming tutorial is from the Khan Academy (2017b). Each programming lesson replays the actions of an expert with audio commentary. The web page is split between the coding area and an area displaying the result of running the code. Edits of the code are immediately reflected in the output/visualization area. Students can at any time pause the playback and experiment with their own edits or additions to the code area and receive instant feedback.

Another trend of this decade is the programming of smart phones. The *MIT App Inventor* (Wolber et al., 2011) enables learners to build Android apps in a web browser that can be run either on a phone or in a phone emulator in the browser. It relies upon a variant of the block syntax made popular by Scratch. Pocket Code (Slany, 2014) enables learners to build phone apps on their phones. Its block syntax and interface were designed to work on small screens and touch sensitive devices.

The Internet, large open online databases, and powerful computers has led the possibility that students can incorporate “big data” into the apps they create. NetsBlox (2017) is a Snap! extension that provides easy-to-use access to a wide range of databases. Mathematica is also being used by high school students to programmatically explore a wide variety of real-world datasets (Wolfram, 2016).

Machine learning has become a very hot topic in research and industry in the 2010s. Recent efforts have attempted to use machine learning to provide personalized tutoring (Coughlan, 2016). Researchers are now exploring how school students might use speech recognition, image recognition, and machine learning to construct “intelligent” apps and robots (Wolfram, 2017; Kahn; Winters, 2017; Machine Learning for Children, 2017). Students using these systems learn to use and understand an increasingly important new technology.

And perhaps in the process the students reflect upon their own thinking thereby acquiring a deeper understanding of their own problem solving, learning, and thought processes.

Massive open online courses (MOOCs) became a hot topic in computer-supported learning when in 2011 Stanford University offered a free course *Introduction to AI*. Its enrolment quickly reached 160,000. Since then courses have been offered at hundreds of universities world-wide with total enrolment of many millions (Wikipedia, 2017b). Because of the large numbers of students, MOOCs generate "big data". This data can be mined to continually improve courses based upon solid evidence.

8. How does "Computational Thinking" fit?

In 2006, Jeanette Wing published an essay on computational thinking where she wrote "Computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (Wing, 2006). Her article has been followed by over a dozen books and over a thousand articles that use the expression "computational thinking" in their titles. Computational thinking has recently been incorporated in the national curriculum of many countries.

Tedre and Denning (2016) published a paper describing the sixty-year history of the ideas underlying computational thinking. It often went with other names such as algorithmic thinking but many as early as the late 1950s and the 1960s were writing about ideas remarkably similar to those in Wing's essay. The article also reviews the emergence in the last few decades of computational thinking in various scientific disciplines, for example, computational biology. They document many different definitions of computational thinking but all are much narrower than the ideas that Seymour Papert first started writing about fifty years ago.

The good news is that a consequence of the recent excitement about computational thinking is that millions of children have been introduced to programming with an intent that they learn more than just programming. The goal of the better computational thinking curricula, websites, and learning resources is that the students learn a way of thinking that computer scientists use to solve problems, design systems, and understand the world.

The bad news is that the set of computational thinking concepts is a small subset of those that Seymour Papert and colleagues have been for decades have been claiming could fundamentally change learning and teaching. Papert's concept of powerful ideas is a much broader and older than computational thinking. One can read about this broader view of computational thinking as early as 1971 when Papert wrote "Teaching Children Thinking"

(Papert, 1971). A section is entitled “Computer Science as a Grade School Subject” reads like a modern description of computational thinking. The paper focused on the powerful idea that thinking about thinking (with the right conceptual tools) can lead to many meta-cognitive improvements.

Powerful ideas as described in Papert’s 1980 Mindstorms book includes a long list of ideas beyond those connected with computational thinking including debugging, microworlds, reflection, and the idea of powerful ideas.

Papert and colleagues have been promoting a pedagogic framework called *constructionism* for over twenty-five years (Papert; Harel, 1991). It emphasizes the deep learning often follows from students publicly sharing the results of their substantial personally meaningful projects. Projects involving computer programs are not the only kind of constructionist projects but are particularly well-suited for students effectively constructing knowledge. In contrast, too many computational thinking learning resources lack an emphasis on creativity and student-directed projects.

9. Looking back and forward

The best way to predict the future is to invent it. – Alan Kay (Wikipedia, 2017c)

In the last fifty years great inventions in using computers to support learning have been made. These include programming languages designed for children, intelligent tutoring systems, online courses, shared virtual spaces, robotics kits, and thousands of games. And learning doesn’t stop with software specifically designed for education but includes use of mainstream developments such as *Wikipedia*, *Google Earth* and *Maps*, social media, computer graphics and animation authoring systems, photo and video editing, 3D printing, spreadsheets, presentation tools, and collaborative document editors.

As computational technology becomes widespread and matures and as the price of computational hardware decreases, we may finally see the fulfilment of the dreams of Seymour Papert, Nicholas Negroponte, and many others that learning by every child on the planet can change dramatically for the better. Children increasingly have devices that enable them to creatively express themselves in a medium that brings their ideas and creations to life. The hope is that children worldwide will thereby acquire powerful ideas changing them into better problem solvers, thinkers, and learners.

10. References

ABELSON, H.; DISESSA, A. A. **Turtle Geometry:** The Computer as a Medium for Exploring Mathematics. Cambridge: MIT Press, 1981.

ANDERSON, J. R. et al. Cognitive Modelling and Intelligent Tutoring. **Artificial Intelligence**, Vol 42, pages 7-49, 1990.

BBC NEWS **What it's like to have schizophrenia.** Available in: <http://news.bbc.co.uk/1/hi/health/6453241.stm>. Accessed in oct. 12, 2017.

BEGEL, A. **LogoBlocks:** A Graphical Programming Language for Interacting with the World, MIT Advanced Undergraduate Project. May 1996. Available in: <http://research.microsoft.com/en-us/um/people/abegel/mit/begel-aup.pdf>. Accessed in oct. 12, 2017.

BLOCKLY **Try Blockly**. Available in: <https://developers.google.com/blockly/>. Accessed in oct. 10, 2017.

BROWN J. S; VANLEHN, K. Repair Theory: A Generative Theory of Bugs in Procedural Skills, **Cognitive Science**, vol. 4, no. 4, pp. 379-426, 1980.

COUGHLAN, S. Could robots be marking your homework?, **BBC News**, December 14, 2016. Available in: <http://www.bbc.com/news/business-38289079>. Accessed in oct. 10, 2017.

DARTMOUTH COLLEGE COMPUTATION CENTER. **A Manual for BASIC,** the elementary algebraic language designed for use with the Dartmouth Time Sharing System. 1964. Archived from the original on 2012-07-16. Available in: <http://www.bitsavers.org/pdf/dartmouth/BASIC_Oct64.pdf>. Accessed in oct. 10, 2017.

DISESSA, A. Twenty reasons why you should use Boxer (instead of Logo). In: SZABÓ, M. T. (Ed.) **Learning & Exploring with Logo:** Proceedings of the Sixth European Logo Conference. Budapest Hungary, 7-27, 1997.

DRESCHER, G, L. Genetic AI: Translating Piaget into LISP. **Instructional Science**, Volume 14, Issue 3-4, pp 357-380, May 1986.

ENNAL, R. Teaching logic as a computer language in schools. **Proceedings of the First International Logic Programming Conference**, Marseille, France, September 1982.

GOLDSTEIN, I. **Understanding simple picture programs**, 1974 Thesis (doctoral in Artificial Intelligence) Massachusetts Institute of Technology, 1974.

GREENBERGER, M. **Computers in the World of the Future**, Cambridge, MA: MIT Press, 1962.

HARDESTY, L. The MIT roots of Google's new software. **MIT News Office**, August 19, 2010. Available in: <http://newsoffice.mit.edu/2010/android-abelson-0819>. Accessed in oct. 13, 2017.

HAREL, I. **Children Designers:** Interdisciplinary Constructions for Learning and Knowing Mathematics in a Computer-rich School. New Jersey: Ablex Publishing, 1991.

HARVEY, B. **Computer Science Logo Style**, Volumes 1, 2 and 3, Cambridge, MA: MIT Press, Second edition, 1997.

HARVEY, B; MÖNIG, J. Bringing 'No Ceiling' to Scratch: Can One Language Serve Kids and Computer Scientists?, **Constructionism 2010 Proceedings**, Paris, 2010.

HOYLES, C; NOSS, R; ADAMSON, R. Rethinking the Microworld Idea, **Journal of Educational Computing Research**, Volume: 27 issue: 1, p. 29-53, 2002.

KAFAI, Y. B. **Minds in Play:** Computer Game Design as a Context for Children's Learning, Mahwah, NJ, Lawrence Erlbaum, 1995.

KAHN, K. Director Guide, **Technical Report** 482B, MIT AI Lab, December 1979.

KAHN, K. A grammar kit in Prolog. In: YAZDANI, M. editor, **New Horizons in Educational Computing**. Ellis Horwood Ltd., 1984. Also in **Instructional Science and Proceedings of the AISB Easter Conference on AI and Education**, Exeter, England, April 1983.

KAHN, K ToonTalk -- An Animated Programming Environment for Children. **Proceedings of the National Educational Computing Conference**, Baltimore, Maryland, June 1995. Extended version in the Journal of Visual Languages and Computing, June 1996.

KAHN, K. Generalizing by Removing Detail**. Communications of the ACM**, 43(3), March 2000. Extended version in LIEBERMAN, H., editor, **Your Wish Is My Command**: Programming by Example. Massachusetts, MA: Morgan Kaufmann, 2001.

KAHN, K. ToonTalk Reborn, Re-implementing and re-conceptualising ToonTalk for the Web. **Proceedings of Constructionism 2014**, Vienna, August 2014.

KAHN, K; NOBLE, H. The Modelling4All Project -- A web-based modelling tool embedded in Web 2.0. **Proceedings of Constructionism 2010**, Paris, August 2010.

KAHN, K; WINTERS, N. Child-friendly Programming Interfaces to AI Cloud Services, **Proceedings of the EC-TEL 2017 Conference**, Tallinn, Estonia, September 2017.

KHAN ACADEMY **Khan Academy site**. Available in: <https://www.khanacademy.org/>. Accessed in oct. 10, 2017a.

KHAN ACADEMY, **Computer programming**. Available in: <https://www.khanacademy.org/computing/computer-programming>. Accessed in oct. 12, 2017b.

KLOPFER, E. et al. "The Simulation Cycle: combining games, simulations, engineering and science using StarLogo TNG", **E-Learning and Digital Media**, Volume 6 Number 1, 2009.

MACHINE LEARNING FOR CHILDREN. **Teach a computer to play a game**. Available in: <https://machinelearningforkids.co.uk>. Accessed in oct. 12, 2017.

MALONE, T. Toward a theory of intrinsically motivating instruction. **Cognitive science** 5.4, p. 333-369, 1981.

MIT MEDIA LAB, **Scratch statistics**. Available in: <http://scratch.mit.edu/statistics/>. Accessed in oct. 10, 2017.

MOR, Y. ET AL. Thinking in Progress. **Micromath**, The Association of Teachers of Mathematics, 20(2), pp.17-23, 2004.

MORGADO, L; CRUZ, M; KAHN, K. Working in ToonTalk with 4-and 5-year olds. **Proceedings of the IADIS International Conference, e-Society,** Vol. II, IADIS, 2003. NETSBLOX. Available in: <https://netsblox.org>. Accessed in oct. 12, 2017.

MORGADO, L; CRUZ, M; KAHN, K. Radia Perlman – A pioneer of young children computer programming. Current developments in technology-assisted education, **Proceedings of m-ICTE**, 2006.

ONE LAPTOP PER CHILD. **Site one laptop per child.** Available in: <http://one.laptop.org/>. Accessed in oct. 12, 2017.

PAPERT, S. Teaching Children Thinking. **MIT AI Memo 247**, 1971.

PAPERT, S. **Mindstorms**: Children, Computers, and Powerful Ideas. New York: Basic Books, 1980.

PAPERT, S. "Does Easy Do It? Children, Games, and Learning", **Game Developer**, June 1998.

PAPERT, S; HAREL, I. **Constructionism**. New Jersey**:** Ablex Publishing Corporation, 1991.

REPENNING, A. Creating User Interfaces with Agentsheets. **1991 Symposium on Applied Computing**, Kansas City, MO, IEEE Computer Society Press, Los Alamitos, pp. 190-196, 1991.

PLATO HISTORY **Plato History**: remembering the future. Available in: <http://www.platohistory.org/blog/timeline/>. Accessed in oct. 12, 2017.

RESNICK, M. MultiLogo**:** A Study of Children and Concurrent Programming. **Interactive Learning Environments**, 1:3, 153-170, 1990, DOI: 10.1080/104948290010301.

RESNICK, M. **Turtles, Termites, and Traffic Jams:** Explorations in Massively Parallel Microworlds. Cambridge, MA: MIT Press, 1994.

RESNICK, M. et al. Scratch: Programming for All. **Communications of the ACM**, Vol. 52 No. 11, Pages 60-67, 2009.

RESNICK, M; OCKO, S; PAPERT, S. LEGO, Logo, and Design**. Children's Environments Quarterly**, vol. 5, no. 4, 1988.

SARASWAT, V. **Concurrent Constraint Programming**. Cambridge, MA: MIT Press, 1993.

SLANY, W. Tinkering with Pocket Code, a Scratch-like programming app for your smartphone. **Proceedings of Constructionism 2014**, Vienna, August 2014.

SMITH, D, C; CYPHER, A; SPOHRER, J. KidSim**:** Programming Agents without a Programming Language. **Communications of the ACM**, 37(7), pp. 54 – 67, July 1994.

SUSSMAN, G. **HACKER:** A model of skill acquisition. Thesis, 1973 (PhD in Artificial Intelligence), MIT, 1973.

TEDRE, M; DENNING, P. J. The Long Quest for Computational Thinking. **Proceedings of the 16th Koli Calling Conference on Computing Education Research**, November 24-27, 2016, Koli, Finland: pp. 120-129.

UNIVERSITY OF CALIFORNIA, **Berkeley**. Available in: <http://bjc.berkeley.edu/>. Accessed in oct. 12, 2017.

WHITE, B. **Designing Computer Games to Facilitate Learning.** Thesis, 1981 (PhD in Artificial Intelligence) MIT, 1981.

WIKIPEDIA **Living Books series**. 2017a. Available in: <https://en.wikipedia.org/wiki/Living_Books_series>. Accessed in oct. 13, 2017.

WIKIPEDIA **Massive Open Online Course**. 2017b. Available in: <https://en.wikipedia.org/wiki/Massive_open_online_course>. Accessed in oct. 12, 2017.

WIKIPEDIA **Alan Kay**. 2017c. Available in: <https://en.wikipedia.org/wiki/Alan_Kay>. Accessed in oct. 12, 2017.

WILENSKY, U. NetLogo, **Center for Connected Learning and Computer-Based Modeling,** Northwestern University. Evanston, IL, 1999. Available in: <http://ccl.northwestern.edu/netlogo>. Accessed in oct. 12, 2017.

WING, J. M. Computational Thinking**. CACM** Viewpoint, March 2006, pp. 33-35.

WOLBER, D. et al. **App Inventor**. California: O'Reilly Media, 2011.

WOLFRAM, S. **How to Teach Computational Thinking**, September 7, 2016 blog post. Available in: <http://blog.wolfram.com/2016/09/07/how-to-teach-computational-thinking/>. Accessed in oct. 12, 2017.

WOLFRAM, S. **Machine Learning for Middle Schoolers**, May 11, 2017 blog post. Available in: <http://blog.stephenwolfram.com/2017/05/machine-learning-for-middle-schoolers/>. Accessed in oct. 12, 2017.