Teaching Computer Programming to Post-millennial Kids: Overview of Goals, Activities and Supporting Tools

Christophe Ponsard
CETIC Research Centre, Gosselies, Belgium

Keywords: Education, Coding, Computer Science, Computational Thinking, Robotics, Flipped Classrooms STEM.

Abstract: Post-millennial kids have experienced Internet technologies from their early age both at home and at school. Although this familiarity can trigger interest in learning how to engineer software, the learning itself needs teaching instruments that are adapted to their generation. In this paper, we focus on early steps where the teenager is mainly in a discovery phase. Starting from key education goals and skills to develop as children of the 21st century, we look how they relate with computer programming and computational thinking. We also look how synergies can be established with other matters like Sciences, Technology, Engineering, Arts and Mathematics (STEAM). We analyse how such goals can be met using different kinds of activities and supporting tools. Our work is illustrated by practical experiences carried out in Belgium. Based on this, we also propose a general roadmap for setting up education programs targeting those kids.

1 INTRODUCTION

In the last twenty years, Internet and computer technologies have grown and matured to the point of becoming ubiquitous in our daily lives and also in the lives of our children. Post-millennial kids (i.e. born after 2000, also known as ”Generation Z”) are now growing and are about to reach their majority. They are totally familiar with using computers, smartphones, tablets and can quickly find and process information from the web and social networks. This generation is different from the previous generation in terms of multi-tasking, entrepreneurship and expectations. However this also comes at a price of less skills for hand written work and more chances to be affected by attention deficit disorder (Promethean, 2017).

Both primary and secondary school levels (also referred to as ”K12”) have adopted digital technology to support education by providing Internet access, using specific learning software or hardware like interactive boards. In contrast with ”learning through digital means”, ”learning what digital means” is far less developed and this fact is largely recognised: in the US only 40% of U.S. schools offer C.S. classes with computer programming (Gallup, 2016). In Europe, the situation varies a lot depending on country specific policies. A survey carried out in 2014-2015 across 21 European countries showed that coding was part of the curriculum in 16 of them (about 75%) under some form which can be optional or integrated within other courses. It is integrated at upper secondary school in about half of the countries and compulsory in one third of the countries at specific (vocational) level (Balanskat and Engelhardt, 2015).

Figure 1: A few initiatives for computer science education[1].

In reaction to this situation, a number of complementary initiatives in computer science education have emerged, e.g. CoderDojo (Liao et al., 2011), Devoxx4Kids (Devoxx4Kids, 2010), First LEGO League (FLL) (Kamen, 1999) as depicted in Figure 1. Some are relying on a form of competition either in teams like FLL or individually like Informat-

[1] All trademarks, product names and logos appearing on the figure are the property of their respective owners and are shown for education information purposes only.

Ponsard, C.
Teaching Computer Programming to Post-millennial Kids: Overview of Goals, Activities and Supporting Tools.
DOI: 10.5220/0007755104740480
In *Proceedings of the 11th International Conference on Computer Supported Education (CSEDU 2019)*, pages 474-480
ISBN: 978-989-758-367-4

ics Olympiad (IOI, 1989). In addition, many online resources and tutorials for learning are also available like Scratch (MIT, 2002), CodeCombat (Saines et al., 2013) or Hour of Code (and beyond) (Partovi and Partovi, 2013). They rely on simple to learn languages based on visual block-based paradigms or high level scripting languages like Python. Specific target domains may be used as concrete support for developing coding skills like robotics (used by FLL, Devoxx4kids) or mobile application programming, e.g. MIT App Inventor (MIT, 2012).

The ICT industry is also sponsoring those initiatives in order to encourage students to engage in a computer science curriculum given the current shortage of skilled workers in this domain (D-Soul, 2013), even though it might be in the first place for their own interest (Tarnoff, 2017). Of course every kid will not turn into a computer scientist. The goal of education should be centred on providing kids with a consistent education relying on the use of computers both as an education support but also as a tool that can be programmed for meaningful purposes, e.g. in relation with some specific courses or a transverse project mixing Scientific, Technology, Engineering, Mathematics (STEM) and which can also include artistic dimensions (STEAM, with the extra ”A”).

The purpose of this paper is to look a bit more into details in those kinds of goals and analyse how the available activities and tools sketched hereabove fit with them. Our work does not claim to be exhaustive but to give what we believe is a representative overview together with a useful roadmap. It is also anchored in the author personal experience (both as parent and volunteer) in Belgium where the evolution of secondary school teaching is being discussed thoroughly, including about learning computational thinking and computer programming.

This paper is structured as follows. First Section 2 recaps the main goals and skills developed at secondary school in our 21st century. Section 3 explores how Software Engineering can be taught in this context. Section 4 discusses some related work. Finally, Section 5 concludes and discusses some next steps.

2 EDUCATIONAL CONTEXT

2.1 General Goals and Skills

Defining education goals and skills is not an easy task but this work has been tackled by different groups considering the needs of our present world. For example, the P21’s Framework (Trilling and Fadel, 2009) organises the basic skills as follows:

- core subjects including native/foreign language skills, arts, geography, history, mathematics, science, civics.
- more specific 21st century themes like global awareness, economics, health literacy, entrepreneurial literacy

In addition, many transverse skills have been identified. They cover innovation, ICT and ”soft” skills. Those are summarised into ”7C” skills that come on top of the above core skills:

- Critical thinking and problem solving
- Creativity and innovation
- Collaboration, teamwork, and leadership
- Cross-cultural understanding
- Communications, information, and media literacy
- Computing and ICT literacy
- Career and learning self-reliance

2.2 Rationale for Teaching Kids to Code

The importance of learning to code is often emphasised nowadays because of the fourth industrial revolution and its high impact on the computerisation and automation of our personal and professional lives. Computer programming has become a highly demanded skill and it can gives students a competitive advantage when applying to colleges or for a job.

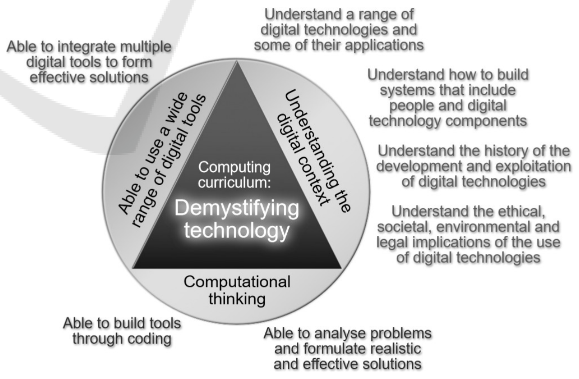

Understand a range of digital technologies and some of their applications
Able to integrate multiple digital tools to form effective solutions
Understand how to build systems that include people and digital technology components
Understand the history of the development and exploitation of digital technologies
Understand the ethical, societal, environmental and legal implications of the use of digital technologies
Able to use a wide range of digital tools
Understanding the digital context
Computing curriculum:
Demystifying technology
Computational thinking
Able to build tools through coding
Able to analyse problems and formulate realistic and effective solutions

Figure 2: Typical goals of a computer curriculum.

Of course the perspective of professional opportunities should not be reduced to job marketing. From an educational point of view, several reasons have been identified, e.g. (Bradford, 2016). Actually “programming” is a complex activity involving the following dimensions that can have educational benefits:

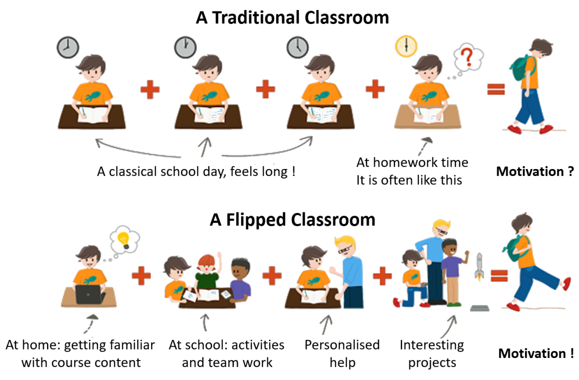
- Computational thinking is a set problem-solving methods that involve expressing problems and their solutions in ways that a computer could execute (Wing, 2006). It covers concepts like decomposition, pattern recognition, abstraction and algorithms. This helps in developing mathematical, analytical as well as organisational skills.

- Coding is the practical implementation of an algorithm into a programming language which can be a visual programming language like Scratch. The early exposure to a computer language with its own syntax and semantics is as rewarding as learning a foreign language. In addition, computer programming requires a high level of rigour in order to be correctly executed.
- Many soft skills are also involved as coding is creative process requiring a lot of communication to understand the problem and collaboration to find a solution. The process is also iterative and includes the need of identifying and correcting errors without falling into "trial an error" mode.

Those high level objectives can be further refined into finer grained goals of a computer curriculum as depicted in Figure 2 from (Baker, 2012). They can be achieved in computer science classes directly or in the context of other matters, e.g. through STEAM activities. Note also that many activities such as computational thinking, collaboration, communication relates to the "7C's" presented in Section 2.1.

2.3 Teaching in the Digital Age

Traditional (or "victorian") teaching has relied on the "I Do" (teacher), "We Do" (in-class exercises), "You Do" (homework) model for years. It does not foster active participation and collaboration. The teacher is also seen as the reference source of information which is no longer true with several sources of information (of various quality) available through Internet.


A Traditional Classroom
A classical school day, feels long !
At homework time It is often like this
Motivation ?
A Flipped Classroom
At home: getting familiar with course content
At school: activities and team work
Personalised help
Interesting projects
Motivation !
Figure 3: Traditional classroom vs flipped classroom.

In the last 20 years, different models have emerged to address those issues and to use teaching possibilities enabled by our digital age:

- Inverted classrooms, depicted in Figure 3, completely reverts the paradigm by using a "You Do", "We Do", "I Do" approach, i.e. students need to discover instructional content outside of the classroom, possibly through digital means like videos. Then they collaborate through online discussions and within classroom under the personalised guidance of the teacher. This requires more engagement but results in higher motivation and positive reinforcement (Baker, 2000).
- Problem based learning is one possible trigger for an inverted classroom. It anchors the learning process in an open problem. Teamwork is used to identify and acquire the required skills to solve the problem. It fosters autonomy and provide strong learning rationales (Amador et al., 2006).
- Online Courses either Massive Online (MOOC) or Small Private (SPOC) can support a blended form of education by providing high quality online content together with collaboration means (Kaplan and Haenlein, 2016).

3 A JOURNEY THROUGH CODE RELATED ACTIVITIES

This section reviews different types of activities supporting the development of programming skills in the large from early phase of discovery to more specific activities relating to computational thinking, coding and project-oriented challenges. Those activities can be roughly structured according to the roadmap sketched in Figure 4. We will illustrate some of them with local experiments.

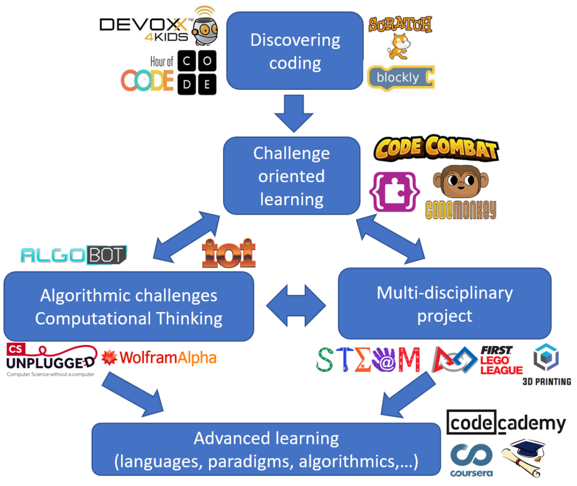


Figure 4: Roadmap for Teaching Coding.

3.1 Discovery Activities

The main goal is to demystify what it means to program in connection with some purpose, e.g. controlling a small robot, producing a simple animation or game. The format is short, with no prerequisite, so it relies on very intuitive tools and the presence of tutor to help in this discovery.

Hour of code and more generally **Coder Dojo** sessions propose various challenges of progressive difficulty and at some point the learner can come with its own project. This can be a small video game but it can also be related to some science project, e.g. Figure 6. Volunteers are present to help in progressing (typically 1 for 6 kids).

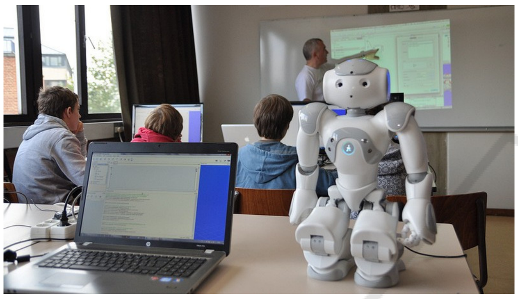


Figure 5: Nao programming session at Devoxx4Kids.

Devoxx4Kids are short activities quite directed but exploring different technologies like Minecraft (java code), Nao/EVC3 (both robotics, block-based) or Scratch (possibly interactive using the webcam). A volunteer is overseeing each activity. Figure 5 shows a programming session with Nao.

3.2 Learning to Code

Visual programming using block-based languages is widely used to teach coding to children because blocks are easy to recognise through their shape, colour and position in a palette. Those shapes are also designed to only allow valid assembly which rules out a large class of syntactic errors that are a main problem for beginners with a textual programming language. Such languages come in different flavours but most of them are now web-based, and Open Source. Many also rely on the Blockly javascript framework (Google, 2012). The most common are:

- Scratch (MIT, 2002) emphasises on sharing, reusing and combining code through its large community. A large repository of code is available. An example of aquaponics simulation (discussed in section 3.3) is depicted in Figure 6.
- MIT App Inventor (MIT, 2012) is directed to the generation of mobile application. In addition to blocks, it also includes a quite powerful yet simple screen designer. It also supports project sharing.
- Microsoft MakeCode (Microsoft, 2018) is a framework for creating domain-specific programming experiences for beginners. It brings computer science to life for all students with fun projects, immediate results, and both block and text editors for learners at different levels.

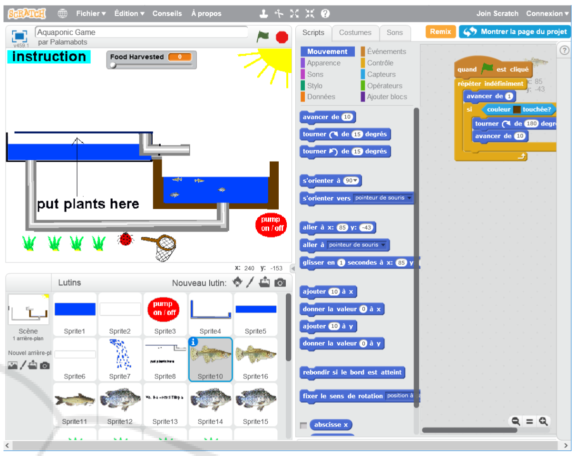


Figure 6: An aquaponic serious game with Scratch.

Many robotics toolkits such as LEGO Mindstorms or Nao also come with a block-based language for programming a robot through specific blocks mapped to the available sensors and actuators. Although often closed, Open Source alternative exists, e.g. Mindstorms blocks are available in Scratch, App Inventor and MakeCode. Figure 7 shows a typical FLL mission and the corresponding Mindstorms code at the top. Note that in parallel to code the robot also needs to be designed as part of a wider project activity (see next section).

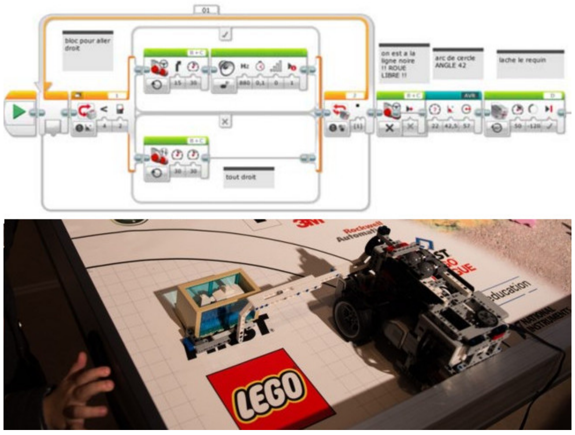


Figure 7: Shark mission: robot and corresponding EV3 code.

Learning textual programming language is proposed by frameworks like CodeCombat (Saines et al., 2013) or CodeMonley (Schor et al., 2016) typically for languages like Python and Javascript variants.

Both those visual and textual languages provide progressive, usually game-based, challenges to teach basic concept such as objects, function calls, arguments, variables, arrays, loops, conditions and various operators. They also typically rely on event-driven programming for managing user interactions, driving animations or controlling hardware (e.g. robot).

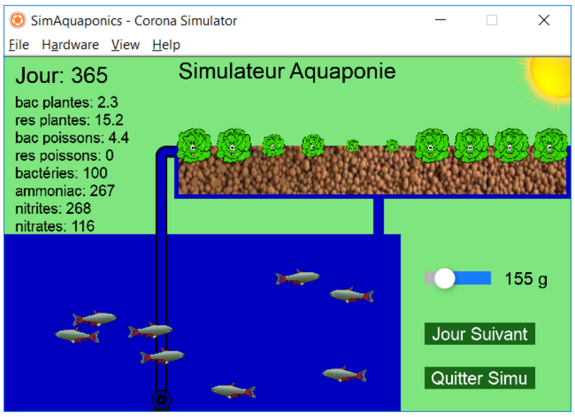
3.3 Project-driven Activities

Projects aim at making pupils work together on a problem and on the design of its solution. In the process, they will learn both technical and non-technical skills. Such projects can be proposed in the scope of a specific course (e.g. programming the robot in computer course) but they are often best carried out in a multi-disciplinary STE(A)M context even if this requires more coordination between teachers.

Project-driven activities are proposed by FLL challenges under two forms, both interesting from the programming perspective:

- the main FLL activity is to program a LEGO robot to achieve a maximum of tasks (or missions) on a thematic mat (e.g. help animal, manage rubbish, living in space...) within 2:30 minutes. This requires to program the EV3 LEGO brick controller but prior to that, it is mandatory to design the robot that will solve a mission so this requires far more that programming skills.
- a STEAM project related to the annual theme must also be developed. A possible project, related to the water theme, is to build an aquaponic system combining fishes, plants and bacteria. It is primary related to biology and the nitrogen cycle but analysing how such a system evolves requires some modelling which can then be studied using an existing simulator or by coding a simplified version. A nice simulator interface will also mobilise artistic skills as depicted in Figure 8.

Actually proposing pupils to try to solve a problem will lead them across a very rich journey starting by defining the problem, then analysing it, investigating possible solutions and finally producing some code as part of the solution (see Section 3.3). This process will trigger creativity and innovation but also some research about what might have been done by others. Different talents can express themselves within such a team. Some might focus on understanding the problem, coding, designing user interface, testing and enhancing, advertising,... But in the end, everybody should bring its contribution to the solution. So having the opportunity to organise a project will exercise far more skills than just learning a programming language to code a well defined problem.



Jour: 365
bac plantes: 2.3
res plantes: 15.2
bac poissons: 4.4
res poissons: 0
bactéries: 100
ammoniac: 267
nitrites: 268
nitrates: 116

Figure 8: Serious game showing an aquaponics simulator.

3.4 Computational Thinking Activities

Purely algorithmic challenges can be part of learning challenges proposed as part of code learning activities but some more specific web-sites also propose dedicated activities such as the AlgoBot serious game (Fishing Cactus, 2018) or CSunplugged (U.Canterbury, 2018) which has to specificity to perform activities without the use of a computer. Algorithmic challenges are also proposed by the Informatics Olympiad which are individual, although a specific coaching is organised in preparation and on-line material is also available (IOI, 1989).

Project driven challenges require to analyse the problem in order to elaborate a solution which can take the form of a code or with some coding is involved in the design of a solution. Looking back at the robot design activity stated previously, it has to meet a number of constraints such as the need to accomplish many tasks in a short time, hence to combine different tasks along a path in a given area. Moreover, the availability of limited sensors and motors requires to design and code the tooling in a modular and efficient way, e.g. using an arm to grab different things or use some clever passive mechanism triggered by contact. A number of useful patterns can be learned and then reused when designing new robot mission.

Most of the time, software is not the central part of a project but using a computer is useful to understand how the designed solution will behave given its complexity. Typically, simulations can be carried out based on a model. Modelling is an abstraction activity in computational thinking. It can rely on different formalisms depending of the kind of system being designed. Going back at the aquaponic example, the biological system is driven by the biochemistry of the nitrogen cycle and can be modelled using system dynamics. A more naive simulation can be developed using a few simple evolution rules using discrete time.

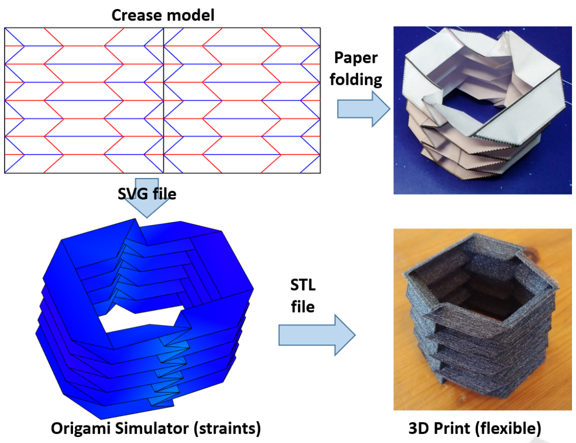


Figure 9: Modelling, simulating and 3D printing an origami.

Figure 9 shows another project relating to the design of foldable solids that could be used in a spatial context. A first abstraction with no thickness can rely on the use of origami paper models. Those can actually be simulated using computer models very simple to design as SVG files modelling the crease patterns. Then using a simple online simulator can help to understand the folding behaviour and reveal the presence of distortions (Ghassaei, 2017). Some good candidates can then be checked further by building them with ticker material or by using 3D printing.

4 DISCUSSION AND RELATED WORK

Our work does not claim to be exhaustive but rather representative of the kind of activities that can be explored to teach various aspect of computer science (including coding) to children and which makes sense with other educational goals. The (Taccle3, 2015) project has carried out a more extensive survey reviewing multiple applications and robotic toolkits (García-Peñalvo et al., 2016).

Computational thinking has also been criticised because of its vagueness and the way it sees the world as a series of problems that have computational solutions (Easterbrook, 2014). The author argues that this view is too narrow and that we need a wider system thinking approach able to deal with a broader context also including the human behaviour and environmental impact, e.g. for sustainability challenges. We are convinced that a problem based approach can actually provide such a perspective and that a computer can be used to understand the dynamics of a system. However, such activities are already quite advanced and reserved for college level, and it is less an issue for younger kids who will enjoy discovering a range of thinking styles and problem solving approaches.

The benefits of multi-disciplinary challenges like the FLL have also been reported. Students involved in such challenges usually improved their learning about real-world applications, problem solving, engagement, communication, and the technology/engineering cycle (Chris, 2013). A more general empirical study on the effect of educational robotics on pupil's capabilities indicates significant impact on mathematics/scientific investigation, teamwork and social skills as well as for technical and soft skills (Kandlhofer and Steinbauer, 2015).

Finally, using gamification for teaching STEM has been reported to encourage a reciprocated engagement between instructors and students during the entire length of the course (Machajewski, 2017). A specific study reviewed and compared how seven countries in Europe and the US are gamifying programming education (Lindberg et al., 2018). It points out that programming games are not a silver bullet capable of replacing Scratch-like tools. It also stresses some learning strategies such as problem-based and the current lack of social learning among players.

5 CONCLUSIONS AND PERSPECTIVES

This short paper has surveyed different educational goals with a focus on learning to program. We have looked into the evolution of teaching paradigms. Based on this, we have sketched a roadmap of activities from demystification to acquiring various coding skills together with more general STEAM skills. We have also illustrated different types of activities that can help growing post-millennial kids to learn coding but also a larger set of useful innovation, communication and collaboration skills. This can be organised at school but requires coordination and involvement of teachers across different disciplines. For now, in Belgium, the approach is only being adapted by a limited number of schools based on motivated teachers but it is certainly a direction to promote for the future evolution of teaching programs.

At this point, our experience is still limited and the feedback is mostly qualitative regarding the effectiveness of the proposed activities. However, we believe this input is worth being shared and will encourage others to publish their own experience. This kind of material could eventually be more systematically gathered, consolidated and published in order to provide better guidelines, especially when it comes to the training of teachers at school or volunteers.

ACKNOWLEDGMENT

Thanks to all the volunteers organising the reported activities: Devoxx4kids, Confluent des Savoir (UNamur), Technobel LEGO Studio, Coder Dojo Wallonia and CS department of UCLouvain.

REFERENCES

Amador, J., Miles, L., and Peters, C. (2006). *The Practice of Problem-Based Learning: A Guide to Implementing PBL in the College Classroom*. JB - Anker. Wiley.

Baker, J. (2000). The 'classroom flip': Using web course management tools to become the guide by the side. In *In selected papers from the 11th Int. Conf. on College Teaching and Learning, Jacksonville, US*.

Baker, M. C. (2012). The little book of computational thinking. http://www.educationvision.co.uk/respaper.html.

Balanskat, A. and Engelhardt, K. (2015). Computer programming and coding Priorities, school curricula and initiatives across Europe. http://www.eun.org/resources/detail?publicationID=661.

Bradford, L. (2016). Why every millennial should learn some code. Forbes http://bit.do/learning-kids-to-code.

Chris, C. (2013). Learning with FIRST LEGO League . In *Proc. of Society for Information Technology & Teacher Education Int. Conf.*

D-Soul (2013). Tech Giants Promote Video with a Simple Message: Kids Need to Learn Programmming. http://bit.do/kids-coding.

Devoxx4Kids (2010). Inspire children to programming, robotics and engineering. http://www.devoxx4kids.org.

Easterbrook, S. (2014). From computational thinking to systems thinking: A conceptual toolkit for sustainability computing. *ICT for Sustainability 2014, ICT4S*.

Fishing Cactus (2018). Algobot. https://www.algobot.be.

Gallup (2016). Research Study (commissioned by Google) Trends in the State of Computer Science in U.S. K-12 Schools. http://csedu.gallup.com/home.aspx.

García-Peñalvo, F. J. et al. (2016). A survey of resources for introducing coding into schools. In *Proc. of the 4th Int. Conf. on Technological Ecosystems for Enhancing Multiculturality*, TEEM '16. ACM.

Ghassaei, A. (2017). Origami Simulator. http://apps.amandaghassaei.com/OrigamiSimulator.

Google (2012). Blockly, a javascript library for building visual programming editors. https://developers.google.com/blockly.

IOI (1989). International olympiad in informatics. http://www.ioinformatics.org/index.shtml.

Kamen, D. (1999). First lego league. http://www.firstlegoleague.org.

Kandlhofer, M. and Steinbauer, G. (2015). Evaluating the impact of educational robotics on pupils' technical- and social-skills and science related attitudes. *Robotics and Autonomous Systems*, 75.

Kaplan, A. M. and Haenlein, M. (2016). Higher education and the digital revolution: About MOOCs, SPOCs, social media, and the Cookie Monster. *Business Horizons*, 59(4):441 – 450.

Liao, B. et al. (2011). The global network of free computer programming clubs for young people. https://coderdojo.com/.

Lindberg, R. S., Laine, T. H., and Haaranen, L. (2018). Gamifying programming education in K-12: A review of programming curricula in seven countries and programming games. *British Journal of Educational Technology*.

Machajewski, S. T. (2017). Application of Gamification in a College STEM Introductory Course: A Case Study. ERIC, Ph.D. Dissertation, Northcentral University School of Business.

Microsoft (2018). Make code. https://www.microsoft.com/makecode.

MIT (2002). Scratch. https://scratch.mit.edu.

MIT (2012). App inventor - learn to build android apps in hours. http://www.appinventor.org.

Partovi, H. and Partovi, A. (2013). Bring computer science to your school or district. https://code.org.

Promethean (2017). Teaching Generation Z: do kids work better with pen and paper than any other medium? https://resourced.prometheanworld.com/teaching-generation-z.

Saines, G., Erickson, S., and Winter, N. (2013). The most engaging game for learning programming. https://codecombat.com.

Schor, J. et al. (2016). Code monkey. https://www.playcodemonkey.com.

Taccle3 (2015). Supporting primary teachers to teach coding. www.taccle3.eu.

Tarnoff, B. (2017). Tech's push to teach coding isn't about kids' success – it's about cutting wages. The Guardian http://bit.do/coding-education-silicon-valley.

Trilling, B. and Fadel, C. (2009). *21st Century Skills: Learning for Life in Our Times*. Wiley.

U.Canterbury (2018). Computer Science without a Computer V4. https://csunplugged.org.

Wing, J. M. (2006). Computational thinking. *Commun. ACM*, 49(3).