e-ISSN 2444-8729

https://doi.org/10.14201/eks20171823358

Hacia la educación del futuro: El pensamiento computacional como mecanismo de aprendizaje generativo

Towards the Education of the Future: Computational Thinking as a Generative Learning Mechanism

Eduardo Segredo, Gara Miranda, Coromoto León

Dpto. de Ingeniería Informática y de Sistemas, Universidad de La Laguna, Spain. {esegredo, gmiranda, cleon}@ull.edu.es

Resumen

La transformación de la educación tradicional en una educación *"SMART"* (del inglés, *"Sensitive, Manageable, Adaptable, Responsive and Timely"*) implica la modernización integral de todos los procesos educativos. Para dicha transformación, la incorporación de nuevas pedagogías se vuelve imprescindible a nivel metodológico, mientras que el uso de entornos interactivos e inteligentes de aprendizaje supone un hito fundamental a nivel tecnológico. En cualquier caso, el objetivo último de esta transformación es formar y transformar a los estudiantes del futuro para que desarrollen habilidades del siglo XXI y puedan convertirse así en ciudadanos de nuestro mundo en continuo cambio. La tecnología y las computadoras son un aspecto esencial para esta modernización, no solo en términos de soporte tecnológico, sino también en términos de ofrecer nuevas metodologías para el desarrollo de nuevas pedagogías y habilidades. En este contexto, el pensamiento computacional aparece como un mecanismo prometedor para fomentar estas nuevas competencias básicas, ya que ofrece herramientas que se ajustan a los intereses del alumnado y les da la posibilidad de comprender mejor los fundamentos de nuestra sociedad y de los entornos basados en las Tecnologías de la Información y la Comunicación (TIC). En este trabajo, planteamos la necesidad de realizar un esfuerzo para fomentar el desarrollo del pensamiento computacional como una oportunidad para transformar las pedagogías tradicionales en metodologías adaptadas al futuro. Además, presentamos una visión general sobre el pensamiento computacional y analizamos el estado actual de la educación *"SMART"*, haciendo hincapié en la falta de metodologías que permitan apoyar esta transición. Por último, proporcionamos —a aquellos educadores interesados en conseguir un cambio real— información sobre iniciativas dedicadas a la difusión o promoción del pensamiento computacional; herramientas o materiales de apoyo para el desarrollo del pensamiento computacional entre los estudiantes; así como una síntesis de las experiencias y los resultados existentes en relación a la aplicación del pensamiento computacional en entornos educativos.

Palabras clave

Pensamiento computacional; Educación *"SMART"*; Resolución de problemas; Programación informática; Aprendizaje generativo

Abstract

The transformation of traditional education into a *Sensitive, Manageable, Adaptable, Responsive and Timely* (SMART) education involves the comprehensive modernisation of all educational processes. For such a transformation, smart pedagogies are needed as a methodological issue while smart learning environments represent the technological issue, both having as an ultimate goal to cultivate smart learners. Smart learners need to develop 21st century skills so that they can become into smart citizens of our changing world. Technology and computers are an essential aspect for this modernisation, not only in terms of technological support for smart environments but also in terms of offering new methodologies for smart pedagogy and the development of smart skills. In this context, computational thinking appears as a promising mechanism to encourage core skills since it offers tools that fit learners' interests and gives them the possibility to better understand the foundations of our ICT-based society and environments. In this work, we raise to make an effort to encourage the development of computational thinking as an opportunity to transform traditional pedagogies to smarter methodologies. We provide a general background about computational thinking and analyse the current state-of-the-art of smart education, emphasizing that there is a lack of smart methodologies which can support the training of 21st century smart skills. Finally, we provide —to those educators interested in pursuing the philosophy of smart education— information about initiatives devoted to the dissemination or promotion of computational thinking; existing tools or materials which support educators for the development of computational thinking among the students; and previous experiences and results about the application of computational thinking in educational environments.

Keywords

Computational thinking; SMART education; Problem solving; Computer programming; Generative learning

Recepción: 25-04-2017 Revisión: 10-05-2017 Aceptación: 25-05-2017 Publicación: 30-06-2017

1. Introduction

Moving towards the education of the future involves a comprehensive modernization of all educational processes. Such a modernization implies the introduction of smart technologies, systems and devices with the aim of creating new opportunities for academic and training organizations in terms of higher standards and innovative approaches. Most of this modernization is the result of the rapid development of Computer Science fields. However, Computer Science drives innovation throughout the world economy, but it remains marginalized throughout the current education systems. It is necessary to disseminate the real benefits of learning Computer Science in children and young students, focusing primarily on skills and competences developed since it will improve the future access to the labor market, regardless of the profession or sector involved. Nowadays, in our digital economy, it is not enough to be a technological consumer or user, so it is essential to train students —at pre-university and university levels— to be active citizens and creators in a technology-driven society. Citizens of the future must have full confidence in the tools and technologies involved in a smart environment.

In order to perform such a transformation, many initiatives have been launched to promote Computer Science and programming among the population, especially among children and young people. Learning how to program a computer has many benefits for those who practice it, but the highlight is that it helps people to think about solving problems. That is the reason why a new approach to education is being developed currently —at all education levels— for including "*computational thinking*" as an essential element of the curricula. In this paper, the foundations and basic concepts about computational thinking will be presented. Some of the most successful and global initiatives for the dissemination of Computer Science and computational thinking will be also introduced since they could serve as a starting point for those interested on the development of these skills among students. Finally, special attention will be paid to the existing tools which have been specifically designed for teaching students the basics about programming. A thorough study of existing tools and experiences focused on enabling the development of computational thinking will be held and made available to professionals in the educational environments. The achievement of an appropriate education for the present times, not only requires smart devices and smart systems but also students with an appropriate training and specific skills which make them possible to manage in a smart environment.

2. Smart education

Given that computational thinking helps to promote problem solving abilities, critical thinking, and creativity, both educators and business leaders, are increasingly recognizing that it is a new basic

skill necessary for economic opportunity and social mobility. In the coming years, we should build on that progress, by offering every student the opportunity to properly develop this skill. In this sense, educational environments play a key role. From educational institutions, we should make an effort to encourage the development of computational thinking as an opportunity to transform traditional pedagogies into smarter methodologies. This way, we will be able to transform traditional education into "*SMART education*". This term is a concept that has been gaining popularity and recognition in recent years, especially in higher education environments. In this sense, it seems natural that the related term "*smart University*" has also emerged as a key concept in the field. Since it is a relatively recent and novel research field, there are different views about it and its main concepts. Tikhomirov's vision (Tikhomirov & Dneprovskaya, 2015) is that "Smart University is a concept that involves a comprehensive modernization of all educational processes. … The smart education is able to provide a new university, where a set of ICT and faculty leads to an entirely new quality of the processes and outcomes of the educational, research, commercial and other university activities". According to (Coccoli, Guercio, Maresca, & Stanganelli, 2014), smart education can be considered as the education in a smart environment supported by smart technologies, making use of smart tools and smart devices.

In order to achieve these distinctive features, technology is a fundamental and necessary element, but it is not sufficient. Technology should be a fundamental tool, but not the ultimate goal when smart education is being pursued. So, at this moment, if we want to transform the traditional education into a smart education, the implementation and use of technology itself will not be enough. In this regard, a smart educational system should offer rich, interactive, and ever-changing learning environments by exploiting the suite of technologies and services available through the Internet, by empowering individuals' abilities and attitudes, and by encouraging them to interact and collaborate in a framework in which people are co-responsible for raising and appraising the inclination of everyone (Coccoli, et al., 2014). Such smart educational systems act in the context of smart cities, which offer smart services and applications to their citizens to enhance their quality of life. Therefore, smart education should be focused on the use of the available technologies to improve the performance of the educational institutions and to enhance the quality of their graduates.

2.1. Skills for smart citizens

When thinking about the quality and training of future graduates, it is essential to identify the set of skills to develop among learners, and try to detect suitable mechanisms to strengthen such skills. The 21st century demands skills and competence from people in order to function and live effectively at work and leisure time (Zhu, Yu, & Riezebos, 2016). As a key research in the education field, several studies (Greenstein, 2012; Trilling & Fadel, 2012) and initiatives have emerged in order to define,

classify, and promote 21st century skills. In (Trilling & Fadel, 2012), three different dimensions were identified in order to classify 21st century skills: learning and innovation skills (critical thinking and problem solving, communications and collaboration, creativity and innovation); digital literacy skills (information literacy, media literacy, *Information and Communication Technologies* (ICT) literacy); and career and life skills (flexibility and adaptability, initiative and self-direction, social and cross-cultural interaction, productivity and accountability, leadership, and responsibility). In (Zhu, et al., 2016), the authors proposed four levels of abilities in smart education that students should master to meet the needs of the modern society. These abilities are basic knowledge and core skills (reading, writing, arts, *Science, Technology, Engineering, and Mathematics* (STEM), etc.); comprehensive abilities (critical thinking and real-world problem solving); personalized expertise (master information and technology literacy, creativity, and innovation); and collective intelligence (communicate clearly and effectively, collaborate effectively and respectfully in diverse teams). The *Partnership for 21st Century Learning* (p21.org) has proposed a model based on four main components: core subjects (writing, reading, mathematics, art, etc.); learning and innovation skills (creativity, innovation, critical thinking and problem solving, and communication and collaboration); information media and technology skills (needed to manage the abundance of information and also contribute to the build of IT: information literacy, media literacy, and ICT literacy). The *North Central Regional Educational Laboratory* and the *Metiri Group* (North Central Regional Educational Laboratory and Metiri Group, 2003) suggest that 21st century skills are built on basic literacies of language and numeracy. These literacies are essential to later develop what are considered the four basic academic achievements: digital-age literacy, inventive thinking, effective communication, and high productivity. Regarding productivity skills, it involves prioritizing and planning, using real-world tools, and the ability to produce relevant high quality products.

Technology is so present in all areas of our lives, that most experts consider fundamental the inclusion of digital and ICT literacy as a basic ability for all learners and 21st century citizens. Given the importance of digital skills, *the Organization for Economic Co-operation and Development* has organized 21st century skills into different categories to potentially distinguish between those that are more strongly related to ICT from those that are not (Organisation for Economic Co-Operation and Development, 2009): ICT functional skills (that includes skills relevant to mastering the use of different ICT applications), ICT skills for learning (which include skills that combine both cognitive abilities or higher-order thinking skills with functional skills for the use and management of ICT applications), and 21st century skills which bring all those skills considered necessary in the knowledge society but where the use of ICT is not a necessary condition.

2.2. Smart pedagogies and generative learning

The newly required skills will force the educational institutions to transform and adapt in order to cope with learners' needs. It is mandatory to somehow reach integration between the education systems and the industries and organizations which are requesting multidisciplinary workers with complementary competencies and skills. As a result, in a smart environment, the curricula and the courses should also be transformed from traditional to smart, thus promoting a vision that is not limited to the simple acquisition of knowledge, but aims to create culturally qualified personnel by anticipating users' demands (Coccoli, et al., 2014). Moreover, in the context of smart education it makes no sense to train and deal with traditional learners. Smart education must be directed to *smart learners*: learners of the 21st century who are used to the new technologies and the changing world. So, if smart education involves the training of new abilities in a new type of learners by using new technologies and in the context of new curricula, it should be necessary to apply new teaching methodologies. If educators keep applying traditional training techniques, we hardly will get to different or smarter results. For this reason, in the context of smart education, it is completely necessary to implement smart pedagogies. The study of new and smart pedagogies however, is still an open research field which needs to be deeper analyzed.

Previous works have identified the importance of smart methodologies in the context of smart education. For example, in (Zhu & He, 2012) the authors stated that "the essence of smart education is to create intelligent environments by using smart technologies, so that smart pedagogies can be facilitated as to provide personalized learning services and empower learners, and thus talents of wisdom who have better value orientation, higher thinking quality, and stronger conduct ability could be fostered". In the basis of such a definition, in (Zhu, et al., 2016), three essential elements were identified in smart education: smart environments, smart pedagogy, and smart learners. This way, smart pedagogies are needed as a methodological issue, while smart learning environments represent the technological issue, both having as an ultimate goal to cultivate smart learners as results. In this sense, smart pedagogies and smart environments support the development of smart learners.

Smart pedagogies deal with learning processes that should be tailored according to the students' learning needs, including requirements, background, interests, and preferences, among others (Sampson & Karagiannidis, 2002). Interest-driven personalized learning emphasizes the interests of students and can foster intrinsic motivations, thus promoting the personalized expertise for students (Gradel, Edson, Gradel, & Edson, 2011). Smart pedagogies must also deal with new technologies and smart environments so many studies are devoted to online and cooperative learning (*Transforming American education: Learning powered by technology*, 2010). In (Zhu, et al., 2016), a set of instructional

strategies were proposed in order to accomplish new pedagogies of smart education:

- Class-based differentiated instruction: differentiated instruction is a process to approach teaching and learning for students with different abilities in the same class.
- Group-based collaborative learning: two or more people learn something together.
- Individual-based personalized learning: adjusting approach (differentiation) and connecting to the learners' interests and experiences to meet the students' needs and provide supporting to foster learning ability among individual students.
- Mass-based generative learning: generative learning involves the creation and refinement of personal mental constructions about the environments (Ritchie & Volkl, 2000).

The basic premise of *generative learning* theories is that learning occurs when learners apply appropriate cognitive processes to incoming information (Fiorella & Mayer, 2014): selecting (attending to relevant material), organizing (mentally organizing incoming material into a coherent cognitive structure) and integrating (connecting cognitive structures with each other and with relevant material activated from long-term memory). In (Fiorella & Mayer, 2014) the authors identify eight learning strategies that promote such understanding: learning by summarizing, learning by mapping, learning by drawing, learning by imagining, learning by self-testing, learning by self-explaining, learning by teaching, and learning by enacting. From our point of view, *learning by programming* should be also considered as a promising learning strategy since it is able to encompass several of the above features while representing a source of motivation and interest for learners.

Some of the aforementioned strategies can be supported by the usage of *mindtools*. Mindtools (Jonassen, 2014) are computer systems that engage students in meaningfully and constructively thinking and learning via stimulating or guiding them to interpret, analyze, synthesize, and organize knowledge during the learning process (Chu, Hwang, & Tsai, 2010). In (Jonassen, Carr, & Yueh, 1998), it is emphasized the importance of mindtools by addressing that "technologies should not support learning by attempting to instruct the learners, but rather should be used as knowledge construction tools that students learn with, not from". In this way, learners function as designers, and the computers function as mindtools for interpreting and organizing their personal knowledge (Jonassen, et al., 1998). Computer applications, such as database systems, spreadsheets, expert systems, semantic nets, video conferencing systems, multimedia and hypermedia authoring tools, programming tools, and simulation programs, among others, are potential mindtools if they are used properly (Jonassen, 2000). To help students to comprehend and organize knowledge, solve problems, and make inferences based on what they have learned, it is important to provide them the right mindtools to deal with different learning tasks or solve different types of problems at the right time and in the right context

(Chu, et al., 2010). Therefore, mindtools also play an important role in helping students to learn in smart ways (Hwang, 2014). Consequently, rather than using the power of computer technologies to disseminate information, they should be used in all subject domains as tools for engaging learners in reflective, critical thinking about the ideas they are studying (Kirschner & Wopereis, 2003).

We are interested in mindtools because they are related to helping users to think for themselves, make connections among concepts, and create new knowledge. With the usage of mindtools we can train a way of thinking about and using ICT, other technologies, learning environments, or intentional and incidental learning activities/opportunities (constructivist in nature), so that users of those tools can represent, manipulate, and reflect on what they know instead of reproducing what others tell them (Kirschner & Wopereis, 2003). Some authors however, have detected what it is called the "*technological paradox*" (Salomon, 2016): the consistent tendency of the educational system to preserve itself and its practices by the assimilation of new technologies into existing instructional practices. Technology becomes "domesticated", which really means, that it is allowed to do precisely that which fits into the prevailing educational philosophy of cultural transmission.

Considering the opportunities that technologies offer in the field of education, we are interested in applying them not only to "*modernize*" the old methodologies, but also to implement new pedagogical strategies that better suit within a smart education. We propose the introduction of computational thinking as a tool for generative learning and a strategy to develop some of the most demanded skills for nowadays students. Computational thinking can be developed without an explicit usage of computers. However, we are interested on the development of computational thinking through computer programming foundations, since it better matches with the students' interests and motivations.

3. Computational thinking

Computational thinking could be described as the thought processes involved in problem formulation and solutions representation, so that these solutions can be implemented by a processing information agent (either a human, a computer or combinations of both). This term became famous thanks to Wing (2006), who introduced computational thinking as a procedure that allows problem solving, designing systems, and understanding human behavior by the use of fundamental concepts of computing. The concept is relatively recent, so there is still no consensus on its definition, thus having multiple variants (Barr & Stephenson, 2011; K. Brennan & Resnick, 2012; Grover & Pea, 2013). For instance, the *International Society for Technology in Education (ISTE)*, as well as the *Computer Science Teachers Association (CSTA)*, defines computational thinking as a process for problem solving which includes at least the following dimensions:

- Formulate problems to allow the use of computers to solve them.
- Organize and analyze data logically.
- Represent data through abstractions, models and simulations.
- Automate solutions through algorithmic thinking, i.e. through a series of orderly steps that achieve those solutions.
- Identify, analyze and implement possible solutions in order to find the most efficient and effective combination of steps and resources.
- Generalize the process of problem solving to wide range of problems.

Since the first appearance of the term in 2006 (Wing, 2006), computational thinking has attracted attention in the context of primary and secondary education, and not only in English-speaking countries, but also in others, such as Spain (García-Peñalvo, 2016a; 2016b; Llorens-Largo, 2015). *The National Research Council (NRC)* of the United States recommends mathematics and computational thinking as one of the eight main practices in the STEM fields (*A Framework for K-12 Science Education,* 2012). In USA, *Computer Science for All* is the President's bold new initiative "to empower all American students, from kindergarten through high school, to learn computer science and be equipped with the computational thinking skills they need to be creators, and not just consumers, in the digital economy, and to be active citizens in our technology-driven world". Many other initiatives have emerged worldwide for the dissemination of computational thinking among young people and among the population in general. This promotion is usually done from the approach of computer programming. In words of Steve Jobs: "Everybody in this country should learn how to program a computer… because it teaches you how to think".

In this sense, some of the definitions of computational thinking believe that students make use of computational thinking even when they do not use any kind of software tool. Conversely, programming itself implies that students make use of computational thinking through the construction of artefacts (Kafai & Burke, 2013; Resnick, et al., 2009). Considering computer programming as a methodology for computational thinking, in (Brennan & Resnick, 2012), three dimensions were proposed: computational concepts, computational practices, and computational perspectives. Table 1 shows a description and some examples for each of those three dimensions. They allow us to understand how students address programming learning. The knowledge of the programming language involves the syntactic, semantic, and schematic knowledge (computational concepts), as well as the strategic knowledge (computing practices).

Dimension	Description	Examples
Concepts	Concepts used by programmers	Variables, statements, etc.
Practices	Problem solving practices that arise during programming tasks	Be incremental and iterative
		Testing and debugging Reusability
		Abstraction Modularity
Perspectives	Students' knowledge about themselves, their relationships with equals, and the technological world that surrounds them	Express and question ideas about technology

Table 1. Dimensions for computational thinking

3.1. Computer programming and problem solving

Computer programming, algorithmic programming, or simply *programming*, is the process of designing, coding, debugging, and maintaining the source code of computer programs. The source code is written in a programming language in order to create programs that exhibit a desired behavior. Programs are usually created to address the solution of a given problem. Programmers analyze problems and define the algorithms which facilitate their solution through the usage of computers. An algorithm is a method that consists of a sequence of precise instructions for solving a given problem (Futschek, 2006). *Algorithmic thinking* is a concept strictly related to computational thinking. It is considered one of the key concepts which allow people to be fluent in the use of information technology. The NRC describes algorithmic thinking as a set of concepts that includes functional decomposition, repetition (iteration and/or recursion), organization of basic data (structures, registers, matrix, list, etc.), generalization and parameterization, algorithms vs. programs, top-down design, and refinement, among others. According to (Futschek, 2006), algorithmic thinking includes the following capabilities or competencies: 1) analyze given problems, 2) specify or represent a problem accurately, 3) find the basic and appropriate operations (instructions) to solve a given problem, 4) build an algorithm to solve the problem following the given sequence of actions, 5) think about all possible cases (special or not) of a given problem, and 6) improve the efficiency of an algorithm.

Algorithmic thinking can be understood as the pre-programming step, i.e., the analysis phase prior to the implementation of the computer program. Globally thinking about the process: there is a problem to be solved, so the programmers deeply study and analyze the problem in order to design an algorithm for its resolution, and finally, they write the source code which implements the designed algorithm. As a result, a computer program —which is able to solve the given problem— is obtained. In the field

of computer programming or software development there are a number of tools which can assist during this process of analysis, design, and implementation: software for data modelling, planning, project management, debugging, testing, etc. The usage of such a technology is not only valuable for computer programmers but also for learners who are interested on training time management, project management, team management, and decision making, among other abilities.

"*Mindtools: Essential Skills for an Excellent Career*" ('Mind Tools', 2016) is a web platform for training the practical, straightforward skills necessary to excel in a professional career. These skills can help learners to become exceptionally effective, thus making possible to become a great manager or leader. These skills can be trained and, if done in a proper manner, can make the very most of the opportunities open to students. According to ('Mind Tools', 2016), the most essential skills for an excellent career are leadership skills, team management, strategy tools, problem solving, decision making, project management, time management, stress management, communication skills, creativity tools, learning skills, and career skills. Many of those skills are trained when developing computer programs. Problem solving can be seen as the main task or objective, while some other issues appear necessary during the problem-solving process. In fact, problems are at the center of what many people do at work every day. Whether you are solving a problem for a client (internal or external), supporting those who are solving problems, or discovering new problems to solve, the problems you face can be large or small, simple or complex, as well as easy or difficult.

A fundamental part of every manager's role is finding ways to solve them. Therefore, being a confident problem solver is really important for a person's success. Much of that confidence comes from having a good process to use when approaching a problem (Jonassen, 2010). There are four basic steps in solving a problem ('Mind Tools', 2016): 1) defining the problem, 2) generating alternatives, 3) evaluating and selecting alternatives, and 4) implementing solutions. For the first step, it is necessary to develop communication abilities and critical thinking. Creativity is essential for the second step. Decision making is required for the third step. Finally, some abilities for the management of time, projects or teams are involved in the fourth and last steps. These general steps for problem solving can be extended to software development environments. In fact, it can be seen as a particular case of problem solving, since in this case, the unique particularity is that the implementation of solution is made through the usage of computers. Therefore, those involved in computer programming inherently develop these skills for problem solving. As we previously mentioned, computational thinking could be described as the thought processes involved in formulating problems and representing their solutions, so that these solutions can be executed by an information processing agent. Bearing the above in mind, what has been called computational thinking is implicitly developed by those engaged in programming or the development of IT applications: the language of computers and the foundations of computers are used to talk about the universe and its processes.

What we propose in the current work is that computational thinking may be used as a more general learning methodology, not uniquely devoted to those interested in a professional career in the field of Computing, but also for every learner interested on training useful and promising skills. We propose a problem-based smart learning environment including information processing activities, scaffolding and reflection to develop both, computing practices and computational perspectives (see Table 1). For educators who are not experts on computer programming issues, the first approach to computational thinking is to find projects, initiatives, courses, materials, and tools that can help them during the process.

4. Initiatives and projects

Learning how to program a computer has many benefits for those who practice it, but the highlight is that it helps people to think about solving problems. That is the reason why a new approach to education is being developed currently —at all education levels— for including computational thinking as an essential element of the curricula. Moreover, many initiatives have been launched to promote programming among the population, especially among children and young people. For instance, we should note *TACCLE 3 – Coding* (García-Peñalvo, 2016a), a European Union Erasmus+ KA2 Programme project aimed to support primary school staff that teaches computing to 4-14 years old children. Another important initiative is *The Hour of Code* ('Code.org', 2016) is a global initiative consisting of one-hour introduction to computer science. It was designed to demystify code and show that everyone can learn the basics. The goal is not to teach everybody to become an expert computer scientist in one hour. Only one hour is enough to learn that computer science is fun and creative, that it is accessible at all ages, for all students, regardless of their background. Similar initiatives are: *Made With Code, Code Club, CoderDojo, Code Week, All you need is* {C<3DE}, and *Bebras Contest,* among others.

Computer Science for All ('Computer Science For All', 2016) is a project promoted by the White House which intends to empower a generation of American students with the computer science skills they need to thrive in a digital economy. *Google CS First* ('Google CS First', 2016) is a project which is intended to inspire kids to create with technology through free computer science clubs. Google is also promoting computational thinking by the creation and dissemination of materials and courses for educators ('Google for Education', 2016). In addition to these projects and dissemination initiatives, some tools have emerged —most of them based on visual programming languages— to allow teaching programming to non-experts users.

In computing, a *visual programming language* is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. They allow users to program through visual expressions, spatial arrangements of text and graphic symbols,

used either as elements of syntax (Ralston, Reilly, & Hemmendinger, 2000). Traditional programming languages such as Java or C++ have representations that closely resemble the computer's way of thinking (Smith, Cypher, & Tesler, 2000). On the other hand, visual programming languages use representations that are closer to human language. These visual programming languages are usually less powerful than traditional languages as they are domain-specific. It is better to use visual programming languages rather than traditional programming languages to facilitate the three dimensions of computational thinking because unnecessary syntax is reduced and the commands are closer to spoken languages. Users usually need only to drag and drop command blocks (Lye & Koh, 2014). With these features, those programming tools help students to reduce the cognitive load and "allow students to focus on the logic and structures involved in programming rather than worrying about the mechanics of writing programs" (Kelleher & Pausch, 2005).

5. Tools for computational thinking

5.1. Logo

Logo (Papert, 1980) is a dialect of Lisp with much of the punctuation removed to make the syntax accessible to newbies. It was intended to allow users to explore a wide variety of topics from mathematics and science to language and music. The most well-known part of Logo is the Logo *turtle.* It began as a robotic turtle that could draw on the ground and was later replaced by a simulated actor in a two-dimensional graphical world that can move, turn, and leave trails. The turtle's directions are object-centric; if a user tells the turtle to "*forward 10*" (FD 10), it will move in its own forward direction rather than a direction defined by the screen. Logo is an interpreted language with descriptive error messages. Since Logo was the first proposal in such a field, many studies have been conducted in order to somehow measure the effects that learning programming —and thus developing computational thinking— have on the development of other cognitive abilities (Clements, 1987; Clements & Gullo, 1984; Miller, Kelly, & Kelly, 1988; Nastasi, Clements, & Battista, 1990; Statz, 1973).

5.2. Scratch

Scratch (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010; Resnick, et al., 2009) —developed at the *Massachusetts Institute of Technology (MIT) Media Lab*— offers a visually appealing environment allowing students to learn programming without initially having to write syntactically correct code. Scratch is based on programming 2D graphical objects called sprites, set against a background called the stage. Users write scripts with graphical blocks that represent various programming constructs to

animate the sprites, make them interact amongst themselves, and change their appearances. Scratch allows students to import images and sounds, apart from creating their own media, to make media-rich projects, which can be shared by the community of users in order to create novel ones. Scratch has an easy-to-use application interface organized into panels, which are presented based on color-coded commands classified by their functionality. It uses blocks which fit into each other like toy building bricks, only when their combination is meaningful and right.

Scratch is one of the most extended tools for the introduction of programming to non-experts users (Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). It is also a consolidated tool for the development of computational thinking skills (Q. Brown, et al., 2008; Ferrer-Mico, Prats-Fernàndez, & Redo-Sanchez, 2012; Gülbahar & Kalelioğlu, 2014).

5.3. Snap!

Snap! (Harvey, et al., 2014; 'Snap! (Build Your Own Blocks) 4.0', 2016) is a free online block-based educational programming language that allows students to create interactive stories, animations, and games, among other creations, while they also learn about mathematical and computational ideas. Snap! was inspired by Scratch, but also targets both novice and more advanced students by including and expanding Scratch's features. Snap! 4.0 is entirely browser-based with no software that needs to be installed locally.

The most important features that differentiate Snap! from Scratch include: first class functions or procedures (their mathematical foundations are called also "*Lambda calculus*"), first class lists (including lists of lists), first class sprites (in other words, prototype-oriented instance-based classless programming), and mix sprites codification of Snap! programs to Python, JavaScript, and C, among other mainstream languages.

5.4. Alice

Alice ('Alice', 2016; Conway, Pausch, Gossweiler, & Burnette, 1994; Cooper, Dann, & Pausch, 2000; Kelleher & Pausch, 2007; UVa User Interface Group, 1995) is an innovative development environment that allows three-dimensional animations to be created. At the same time, Alice is an educational tool aimed to introduce object-oriented programming concepts. Thanks to its usage, students can learn programming basic notions through the creation of animated stories and simple videogames. For doing that, different three-dimensional objects (people, animals, and vehicles, among others) are located in a virtual world, and students design a program in order to animate all those objects. There exists a variant of Alice, referred to as *Looking Glass* ('Looking Glass', 2016), which was developed by the Washington University in St. Louis. It provides some novelties with respect to Alice, such as a

set of high-definition animations, a library of three-dimensional characters and landscapes, and the possibility of creating new complex projects by reusing previously published ones, among others.

Several are the works that can be found in the related literature regarding the usage of Alice for educational purposes, and more particularly, regarding computational thinking (Tabet, Gedawy, Alshikhabobakr, & Razak, 2016).

5.5. App Inventor

App Inventor (Abelson & Friedman, 2010; 'MIT App Inventor', 2016; Xie, Shabir, & Abelson, 2015) is a visual programming tool based on blocks which allows completely functional applications for Android devices to be built. Students can program their first application in only a few hours, and build much more complex applications in a shorter period of time in comparison to the usage of traditional text-based programming languages. As it was stated by its authors, App Inventor "seeks to democratize software development by empowering all people, especially young people, to transition from being consumers of technology to becoming creators of it" ('MIT App Inventor', 2016).

With respect to the usage of App Inventor as an educational tool for promoting computational thinking, there also exist a significant number of papers published in the related literature (Maiorana, Giordano, & Morelli, 2015; Roscoe, Fearn, & Posey, 2014).

5.6. Greenfoot

Greenfoot ('Greenfoot', 2016; Henriksen & Kölling, 2004; Kölling, 2008a, 2010) is aimed to teach object-oriented programming with Java. Students create worlds where they locate different actors in order to generate different graphic-based applications, such as games, simulations, and stories, among others. There exist communities for both learners and educators. The former is called The Gallery and provides a platform to publish and discuss different projects. With respect to the latter, it is referred to as the *Greenroom* (N. Brown, Stevens, & Kölling, 2010), and it allows discussing teaching strategies, exchanging experiences and sharing resources. In Greenfoot standard textual Java code is used for coding. Greenfoot enables an easy transition into other development environments, such as *BlueJ* ('BlueJ', 2016; Kölling, 2008b), as well as into more professional programming tools.

With respect to the related literature, it is worth mentioning that the number of papers published regarding Greenfoot as a tool for promoting computational thinking is, as far as we know, almost non-existent in comparison to other tools, like Alice or App Inventor (Rick, Ludwig, Meyer, Rehder, & Schirmer, 2010). However, several papers comparing Greenfoot, Alice, and Scratch, in terms of their features, goals, and audiences have been published (Fincher & Utting, 2010; Utting, Cooper, Kölling, Maloney, & Resnick, 2010).

5.7. Pencil Code

Pencil Code (Bau & Bau, 2014; Bau, Bau, Dawson, & Pickens, 2015; 'Pencil Code', 2016) allows drawing art, playing music, and creating games by means of a collaborative programming site. In addition it can also be used to experiment with mathematical functions, geometry, graphing, webpages, simulations, and algorithms. Although Pencil Code mainly focuses on the language *CoffeeScript* ('CoffeeScript', 2016), it can also be used for learning JavaScript, HTML, and CSS. It is worth mentioning the wide range of useful reference materials and examples that are provided at the Pencil Code website. Educators have a large number of printable classroom materials at their disposal, as well as the Pencil Code teacher's manual.

Taking into account that Pencil Code is one of the most recently proposed tools, literature regarding the usage of this tool for promoting computational thinking is almost non-existent (Weintrop, 2015).

5.8. AgentSheets and AgentCubes

AgentSheets ('AgentSheets', 2016; Alex Repenning, 1993) is a tool that allows students to create agent-based computational science applications, simulations, and games, and share them online. At the same time, it may be used to teach computer science concepts and logic, as well as to promote computational and algorithmic thinking. In a similar way, *AgentCubes* ('AgentCubes', 2016; Ioannidou, Repenning, & Webb, 2009; A. Repenning & Ioannidou, 2006) provides the mechanisms required for creating three-dimensional shapes. Those shapes can be then programmed, turned into games, and published online. We should note that, in opposition to the approaches introduced in previous sections, which are free, complete versions of AgentSheets and AgentCubes must be purchased, although there is available a trial version of AgentSheets, as well as a free lite version of AgentCubes. Finally, a completely online version of AgentCubes, termed as AgentCubes online ('AgentCubes online', 2016), can also be found.

In the cases of AgentSheets and AgentCubes, there exist a noticeable number of publications in regard to their usage to develop computational thinking, and more generally, for educational purposes. For a complete list of publications, the reader is referred to ('AgentSheets', 2016).

5.9. AgentSheets and AgentCubes

The aforementioned tools are ideal for introducing computational thinking —and programming main foundations— to young people and adults and, of course, at different education stages. However, when dealing with younger students (especially children under 10), it is necessary to have other tools that

are better suited to their needs. In such a case, there are some available apps, games, and educative tools as: *Kodable, Cargobot, ScratchJr, LightbotJr, Robot Turtles, Hopscotch, Lightbot, Kodu, Gamestar Mechanic, GameMaker, My Robot Friend, SpaceChem, CodeCombat, Minecraft.edu,* etc. At the same time, other tools involve the usage of hardware, thus becoming much more attractive for students: *Raspberry Pi, Hummingbird Robotics Kit, Lego® Mindstorms, Dash and Dot, and Sphero and Ollie*, among others. In most cases they are based on programming robots.

6. Discussion

A comprehensive research has been conducted in order to detect existing initiatives, projects and tools which can support the development of computational thinking. However, when first approach is done to a field, it is important to have a general and global view about alternatives and its features. Table 2 shows a comparison of some of the most important tools we have analyzed in the previous section. The following dimensions have been selected:

Free software: indicates whether the tool has been released under some free software license or, on the contrary, if a license has to be purchased.

- Online tool: shows if the tool can be accessed and used through a navigator or if it has to be installed on a computer.
- Online repository available: is there any online repository where users can upload their projects in order to share them with the community?
- Project reusability/remixing: can users download projects from an online repository and use them as the starting point for their new creations?
- Learning difficulty: this dimension is related to the learning difficulty of the tool. Three different levels have been established (low, medium, and high).
- Block-based/Text-based/Both: indicates if the tool allows users to program through blocks, text, or if both options are available.
- Target programming language: this dimension shows if the tool is aimed at teaching a specific programming language.

It can be observed that the number of free software tools is much higher than the number of tools which have to be purchased. The above shows the tendency to make tools that promote computational thinking abilities available to the largest possible amount of people. After all, computational thinking should be viewed as a general approach for problem solving, and it should not be only applied by

computer scientists or developers. Another advantage of free software tools is they may be altered by the community with the aim of improving them and making them more powerful. At the same time, Table 2 also shows how the tendency is to provide online tools with online repositories where users can share their creations, as well as download them to start new projects.

With respect to the learning difficulty, and generally speaking, block-based tools are easier to learn and use than text-based ones, with the exception of Logo. Although Logo is a text-based tool, it provides a set of very intuitive commands which makes its learning and usage very straightforward. It is worth mentioning the case of Blockly, which is a library to create visual programming languages. Therefore, its learning difficulty is much higher than the remaining tools, since it is aimed at developers rather than learners who want to develop their computational thinking abilities. Finally, we should note that Pencil Code is the only tool that provides both block-based and text-based programming modes. Moreover, only a few tools are aimed at teaching specific programming languages: Logo and Pencil Code.

	Free	Online tool	Online repository available	Project reusability / remixing	Learning difficulty	Block-based / Text-based / Both	Target programming language
Logo (Turtle Academy)	✓	✓	✓	×	Low	Text-based	Lisp (dialect)
Scratch	✓	✓	✓	✓	Low	Block-based	N/A
Snap!	✓	✓	×	×	Low	Block-based	N/A
Alice	✓	×	×	×	Medium	Block-based	N/A
Looking Glass	✓	×	✓	✓	Medium	Block-based	N/A
App Inventor	✓	✓	✓	✓	Low	Block-based	N/A
Greenfoot	✓	✓	✓	✓	High	Text-based	Java
Pencil Code	✓	✓	✓	✓	Low	Both	CoffeeScript, JavaScript, HTML, CSS
AgentSheets	×	✓	✓	×	Medium	Block-based	N/A
AgentCubes	×	✓	✓	×	Medium	Block-based	N/A
AgentCubes Online	✓	✓	✓	✓	Medium	Block-based	N/A

Table 2. Comparison of tools that promote the development of computational thinking abilities depending on different features

7. Conclusions and future steps

Many initiatives have arisen to encourage the presence of computational thinking in primary and secondary classes (Lye & Koh, 2014). However, not so many countries have made a clear position about introducing computational thinking in the curricula. From our point of view, computational thinking could also bring many benefits to pedagogical methodologies. The achievement of smart education not only requires smart devices and smart systems but also students with an appropriate training and specific skills which make them possible to manage in a smart environment. For this reason, we have performed a deep analysis about computational thinking and its possibilities for developing a "smart" and higher-quality education for the citizens of the future.

The research carried out, as well as the obtained findings and outcomes enable us to extract the following conclusions:

- For developing more powerful and helpful learning environments, it is not enough to incorporate new technologies, but it is also mandatory to introduce new learning criteria and methodologies.
- "*Being smart*" should not be confused with "*being digital*", i.e., the ICT infrastructures are the means, not the end, so it is not enough to train learners on the usage of isolate computer programs. In our digital economy, it is not enough to be a technological consumer or user, it is necessary to be active citizens and creators.
- Computational thinking provides a new opportunity for training 21st century skills and for developing new learning strategies.
- This work provides a thorough review of existing projects, initiatives, tools, and experiences whose objective is focused on the development of computational thinking abilities. The idea was to provide a comprehensive and detailed vision for those interested in introducing computational thinking into their education environments.

As shown in the current work, there are many resources and tools which can help us to promote computational thinking among learners. However, it would be also interesting to measure how the training on computational thinking impacts on the students' development. It is important to measure not only the development of computational thinking, but also the impact this can have on overall skill capacities for solving problems in any field. It is not trivial at all to get a measure of the development of computational thinking, but much less trivial is to establish a relationship among the effects that this development may have on other cognitive abilities of the individual. Consequently, it would be worth designing and carrying out qualitative and quantitative analyses about how the development

of computational thinking influences the improvement of general skills and the ability to understand, model, and solve problems.

8. References

A Framework for K-12 Science Education: Practices, Crosscutting Concepts, and Core Ideas. (2012). Washington, D.C.: National Academies Press. Retrieved from http://nap.edu/catalog/13165

Abelson, H. & Friedman, M. (2010). App Inventor--A view into learning about computers through building mobile applications. In *Proceedings of the 2010 SIGCSE Symposium*.

AgentCubes. (2016). Retrieved 25 September 2016, from http://www.agentcubes.com/

AgentSheets. (2016). Retrieved 25 September 2016, from http://www.agentsheets.com/index.html

Alice. (2016). Retrieved 23 September 2016, from http://www.alice.org/index.php

Barr, V., & Stephenson, C. (2011). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Education Community? *ACM Inroads, 2*(1), 48-54. doi: http://dx.doi.org/10.1145/1929887.1929905

Bau, D., & Bau, D. A. (2014). A Preview of Pencil Code: A Tool for Developing Mastery of Programming. In *Proceedings of the 2nd Workshop on Programming for Mobile & Touch* (pp. 21-24). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/2688471.2688481

Bau, D., Bau, D. A., Dawson, M., & Pickens, C. S. (2015). Pencil Code: Block Code for a Text World. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 445-448). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/2771839.2771875

BlueJ. (2016). Retrieved 25 September 2016, from http://bluej.org/

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. Presented at the Annual American Educational Research Association Meeting, Vancouver, Canada. Retrieved from http://web.media.mit.edu/~kbrennan/files/Brennan_Resnick_AERA2012_CT.pdf

Brown, N., Stevens, P., & Kölling, M. (2010). Greenroom: A Teacher Community for Collaborative Resource Development. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (pp. 305-305). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/1822090.1822181

Brown, Q., Mongan, W., Kusic, D., Garbarine, E., Fromm, E., & Fontecchio, A. (2008). *Computer Aided Instruction as a Vehicle for Problem Solving: Scratch Boards in the Middle Years Classroom*. Presented at the 2008 Annual Conference & Exposition. Retrieved from https://peer.asee.org/computer-aided-instruction-as-a-vehicle-for-problem-solving-scratch-boards-in-the-middle-years-classroom

Chu, H.-C., Hwang, G.-J., & Tsai, C.-C. (2010). A knowledge engineering approach to developing mindtools for context-aware ubiquitous learning. *Computers & Education, 54*(1), 289-297. doi: http://dx.doi.org/10.1016/j.compedu.2009.08.023

Clements, D. H. (1987). Longitudinal Study of the Effects of Logo Programming on Cognitive Abilities and Achievement. *Journal of Educational Computing Research, 3*(1), 73-94. doi: http://dx.doi.org/10.2190/RCNV-2HYF-60CM-K7K7

Clements, D. H., & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. *Journal of Educational Psychology, 76*(6), 1051-1058. doi: http://dx.doi.org/10.1037/0022-0663.76.6.1051

Coccoli, M., Guercio, A., Maresca, P., & Stanganelli, L. (2014). Smarter universities: A vision for the fast changing digital era. *Journal of Visual Languages & Computing, 25*(6), 1003-1011. doi: http://dx.doi.org/10.1016/j.jvlc.2014.09.007

Code.org. (2016). Retrieved 26 September 2016, from https://code.org/

CoffeeScript. (2016). Retrieved 25 September 2016, from http://coffeescript.org/

Computer Science For All. (2016, January 30). Retrieved 29 September 2016, from https://www.whitehouse.gov/blog/2016/01/30/computer-science-all

Conway, M., Pausch, R., Gossweiler, R., & Burnette, T. (1994). Alice: A Rapid Prototyping System for Building Virtual Environments. In *Proceedings of Conference Companion on Human Factors in Computing Systems* (pp. 295-296). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/259963.260503

Cooper, S., Dann, W., & Pausch, R. (2000). Alice: A 3-D Tool for Introductory Programming Concepts. *Journal of Computing in Small Colleges, 15*(5), 107-116.

Ferrer-Mico, T., Prats-Fernàndez, M. À., & Redo-Sanchez, A. (2012). Impact of Scratch Programming on Students' Understanding of Their Own Learning Process. *Procedia - Social and Behavioral Sciences, 46*, 1219-1223. doi: http://dx.doi.org/10.1016/j.sbspro.2012.05.278

Fincher, S., & Utting, I. (2010). Machines for Thinking. *Trans. Comput. Educ., 10*(4), 13:1-13:7. doi: http://dx.doi.org/10.1145/1868358.1868360

Fiorella, L., & Mayer, R. E. (2014). *Learning as a Generative Activity: Eight Learning Strategies that Promote Understanding. Cambridge University Press*. Retrieved from http://www.cambridge.org/es/academic/subjects/psychology/educational-psychology/learning-generative-activity-eight-learning-strategies-promote-understanding?format=AR&isbn=9781316258576#contentsTabAnchor

Futschek, G. (2006). Algorithmic Thinking: The Key for Understanding Computer Science. In R. T. Mittermeir (Ed.), *Informatics Education - The Bridge between Using and Understanding Computers* (pp. 159-168). Berlin Heidelberg: Springer. doi: http://dx.doi.org/10.1007/11915355_15

García-Peñalvo, F. J. (2016a). A brief introduction to TACCLE 3 - coding European project. In 2016 International Symposium on Computers in Education (SIIE) (pp. 1-4). doi: http://dx.doi.org/10.1109/SIIE.2016.7751876

García-Peñalvo, F. J. (2016b). What Computational Thinking Is. *Journal of Information Technology Research, 9*(3), v-viii.

Google CS First. (2016). Retrieved 29 September 2016, from https://www.cs-first.com/

Google for Education. (2016). Retrieved 29 September 2016, from www.google.com/edu/resources/programs/exploring-computational-thinking/

Gradel, K., Edson, A. J., Gradel, K., & Edson, A. J. (2011). Cooperative Learning: Smart Pedagogy and Tools for Online and Hybrid Courses. *Journal of Educational Technology Systems, 39*(2), 193-212. doi: http://dx.doi.org/10.2190/ET.39.2.i

Greenfoot. (2016). Retrieved 25 September 2016, from http://www.greenfoot.org/door

Greenstein, L. M. (2012). *Assessing 21st Century Skills: A Guide to Evaluating Mastery and Authentic Learning* (1st edition). Thousand Oaks: Corwin.

Grover, S., & Pea, R. (2013). Computational Thinking in K-12 A Review of the State of the Field. *Educational Researcher, 42*(1), 38-43. doi: http://dx.doi.org/10.3102/0013189X12463051

Gülbahar, Y., & Kalelioğlu, F. (2014). The Effects of Teaching Programming via Scratch on Problem Solving Skills: A Discussion from Learners' Perspective. *Informatics in Education, 13*(1), 33-50.

Harvey, B., Garcia, D. D., Barnes, T., Titterton, N., Miller, O., Armendariz, D., … Paley, J. (2014). Snap! (Build Your Own Blocks). In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 749-749). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/2538862.2539022

Henriksen, P., & Kölling, M. (2004). Greenfoot: Combining Object Visualisation with Interaction. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming*

Systems, Languages, and Applications (pp. 73-82). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/1028664.1028701

Hwang, G.-J. (2014). Definition, framework and research issues of smart learning environments - a context-aware ubiquitous learning perspective. *Smart Learning Environments, 1,* Article 4. doi: https://doi.org/10.1186/s40561-014-0004-5

Ioannidou, A., Repenning, A., & Webb, D. C. (2009). AgentCubes: Incremental 3D end-user development. *Journal of Visual Languages & Computing, 20*(4), 236-251. doi: https://doi.org/10.1016/j.jvlc.2009.04.001

Jonassen, D. H. (2000). *Computers as Mindtools for Schools, Engaging Critical Thinking*. Upper Saddle River, New Jersey: Prentice-Hall.

Jonassen, D. H. (2010). *Learning to Solve Problems: A Handbook for Designing Problem-Solving Learning Environments.* New York, USA: Taylor & Francis.

Jonassen, D. H. (2014). Mindtools (Productivity and Learning). In R. Gunstone (Ed.), *Encyclopedia of Science Education* (pp. 1-7). Netherlands: Springer. doi: https://doi.org/10.1007/978-94-007-6165-0_57-1

Jonassen, D. H., Carr, C., & Yueh, H.-P. (1998). Computers as Mindtools for Engaging Learners in Critical Thinking. *TechTrends, 43*(2), 24-32. doi: https://doi.org/10.1007/BF02818172

Kafai, Y. B., & Burke, Q. (2013). Computer Programming Goes Back to School. *Phi Delta Kappan, 95*(1), 61-65. doi: https://doi.org/10.1177/003172171309500111

Kelleher, C., & Pausch, R. (2005). Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv., 37(*2), 83-137. doi: https://doi.org/10.1145/1089733.1089734

Kelleher, C., & Pausch, R. (2007). Using Storytelling to Motivate Programming. *Commun. ACM, 50*(7), 58-64. doi: https://doi.org/10.1145/1272516.1272540

Kirschner, P., & Wopereis, I. G. J. H. (2003). Mindtools for teacher communities: A European perspective. *Technology, Pedagogy and Education, 12*(1), 105-124. doi: https://doi.org/10.1080/14759390300200148

Kölling, M. (2008a). Greenfoot: A Highly Graphical Ide for Learning Object-oriented Programming. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 327-327). New York, NY, USA: ACM. doi: https://doi.org/10.1145/1384271.1384370

Kölling, M. (2008b). Using BlueJ to Introduce Programming. In J. Bennedsen, M. E. Caspersen, & M. Kölling (Eds.), *Reflections on the Teaching of Programming* (pp. 98-115). Berlin Heidelberg: Springer. doi:

https://doi.org/10.1007/978-3-540-77934-6_9

Kölling, M. (2010). The Greenfoot Programming Environment. *Trans. Comput. Educ., 10*(4), 14:1-14:21. doi: https://doi.org/10.1145/1868358.1868361

Llorens-Largo, F. (2015). Dicen por ahí. . . que la nueva alfabetización pasa por la programación. *ReVisión, 8*(2), 11-14.

Looking Glass. (2016). Retrieved 23 September 2016, from https://lookingglass.wustl.edu/

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51-61. doi: https://doi.org/10.1016/j.chb.2014.09.012

Maiorana, F., Giordano, D., & Morelli, R. (2015). Quizly: A live coding assessment platform for App Inventor. In *2015 IEEE Blocks and Beyond Workshop* (pp. 25-30). doi: https://doi.org/10.1109/BLOCKS.2015.7368995

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by Choice: Urban Youth Learning Programming with Scratch. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 367-371). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/1352135.1352260

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *Trans. Comput. Educ., 10*(4), 16:1-16:15. doi: https://doi.org/10.1145/1868358.1868363

Miller, R. B., Kelly, G. N., & Kelly, J. T. (1988). Effects of Logo computer programming experience on problem solving and spatial relations ability. *Contemporary Educational Psychology, 13*(4), 348-357. doi: https://doi.org/10.1016/0361-476X(88)90034-3

Mind Tools: Essential Skills for an Excellent Career. (2016). Retrieved 29 September 2016, from http://www.mindtools.com/

MIT App Inventor. (2016). Retrieved 23 September 2016, from http://appinventor.mit.edu/explore/

Nastasi, B. K., Clements, D. H., & Battista, M. T. (1990). Social-cognitive interactions, motivation, and cognitive growth in Logo programming and CAI problem-solving environments. *Journal of Educational Psychology, 82*(1), 150-158. doi: https://doi.org/10.1037/0022-0663.82.1.150

North Central Regional Educational Laboratory and Metiri Group. (2003). 21st Century Skills: Literacy in the Digital Age. Retrieved from http://pict.sdsu.edu/engauge21st.pdf

Organisation for Economic Co-Operation and Development. (2009). *21st Century Skills and Competences for New Millennium Learners in OECD Countries* (EDU Working paper No. 41). Retrieved from http://www.oecd.org/officialdocuments/publicdisplaydocumentpdf/?cote=EDU/WKP(2009)20&doclanguage=en

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Books, Inc.

Pencil Code. (2016). Retrieved 25 September 2016, from https://pencilcode.net/

Ralston, A., Reilly, E. D., & Hemmendinger, D. (2000). *Encyclopedia of Computer Science* (4th ed.). Hoboken, NJ, USA: Wiley.

Repenning, A. (1993). Agentsheets: A Tool for Building Domain-oriented Visual Programming Environments. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (pp. 142-143). New York, NY, USA: ACM. doi: http://dx.doi.org/10.1145/169059.169119

Repenning, A., & Ioannidou, A. (2006). AgentCubes: Raising the Ceiling of End-User Development in Education through Incremental 3D. In *Visual Languages and Human-Centric Computing* (VL/HCC'06) (pp. 27-34). doi: https://doi.org/10.1109/VLHCC.2006.7

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for All. *Commun. ACM, 52*(11), 60-67. doi: https://doi.org/10.1145/1592761.1592779

Rick, D., Ludwig, J., Meyer, S., Rehder, C., & Schirmer, I. (2010). Introduction to Business Informatics with Greenfoot Using the Example of Airport Baggage Handling. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 68-69). New York, NY, USA: ACM. doi: https://doi.org/10.1145/1930464.1930474

Ritchie, D., & Volkl, C. (2000). Effectiveness of Two Generative Learning Strategies in the Science Classroom. *School Science and Mathematics, 100*(2), 83-89. doi: https://doi.org/10.1111/j.1949-8594.2000.tb17240.x

Roscoe, J. F., Fearn, S., & Posey, E. (2014). Teaching Computational Thinking by Playing Games and Building Robots. In *2014 International Conference on Interactive Technologies and Games* (iTAG) (pp. 9-12). doi: https://doi.org/10.1109/iTAG.2014.15

Salomon, G. (2016). It's Not Just the Tool but the Educational Rationale that Counts. In E. Elstad (Ed.), *Educational Technology and Polycontextual Bridging* (pp. 149-161). Rotterdam, The Netherlands: SensePublishers. doi: https://doi.org/10.1007/978-94-6300-645-3_8

Sampson, D., & Karagiannidis, C. (2002). Personalised Learning: Educational, Technological and Standardisation Perspective. *Interactive Educational Multimedia, 4*, 24-39.

Smith, D. C., Cypher, A., & Tesler, L. (2000). Programming by Example: Novice Programming Comes of Age. *Commun. ACM, 43*(3), 75-81. doi: https://doi.org/10.1145/330534.330544

Snap! (Build Your Own Blocks) 4.0. (2016). Retrieved 29 September 2016, from http://snap.berkeley.edu/

Statz, J. (1973). The Development Of Computer Programming Concepts And Problem-Solving Abilities Among Ten-Year-Olds Learning Logo. Electrical Engineering and Computer Science - Dissertations. Retrieved from http://surface.syr.edu/eecs_etd/256

Tabet, N., Gedawy, H., Alshikhabobakr, H., & Razak, S. (2016). From Alice to Python. Introducing Text-based Programming in Middle Schools. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 124-129). New York, NY, USA: ACM. doi: https://doi.org/10.1145/2899415.2899462

Tikhomirov, V., & Dneprovskaya, N. (2015). Development of strategy for smart University. In *Open Education Global International Conference*. Banff, Canada.

Transforming American education: Learning powered by technology. (2010). (National Educational Technology Plan). Retrieved from https://www.ed.gov/sites/default/files/NETP-2010-final-report.pdf

Trilling, B., & Fadel, C. (2012). *21st Century Skills: Learning for Life in Our Times* (1st Ed.). San Francisco: John Wiley & Sons Inc.

Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, Greenfoot, and Scratch - A Discussion. *Trans. Comput. Educ., 10*(4), 17:1-17:11. doi: https://doi.org/10.1145/1868358.1868364

UVa User Interface Group. (1995). Alice: Rapid Prototyping for Virtual Reality. *IEEE Comput. Graph. Appl., 15*(3), 8-11. doi: https://doi.org/10.1109/38.376600

Weintrop, D. (2015). Blocks, text, and the space between: The role of representations in novice programming environments. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 301-302). doi: https://doi.org/10.1109/VLHCC.2015.7357237

Wing, J. M. (2006). Computational Thinking. *Commun. ACM, 49*(3), 33-35. doi: https://doi.org/10.1145/1118178.1118215

Xie, B., Shabir, I., & Abelson, H. (2015). Measuring the Usability and Capability of App Inventor to Create

Mobile Applications. In *Proceedings of the 3rd International Workshop on Programming for Mobile and Touch* (pp. 1-8). New York, NY, USA: ACM. doi: https://doi.org/10.1145/2824823.2824824

Zhu, Z.-T., & He, B. (2012). Smart Education: new frontier of educational informatization. *E-Education Research, 12*, 1-13.

Zhu, Z.-T., Yu, M.-H., & Riezebos, P. (2016). A research framework of smart education. *Smart Learning Environments, 3*(1), Article 4. doi: http://dx.doi.org/10.1186/s40561-016-0026-2