Are Educators Ready for Coding?
From Students Back to Teacher: Introducing the Class to Coding the Other Way Round

Stefano Federici[1], Elisabetta Gola[1], Denise Brau[2] and Andrea Zuncheddu[2]
[1]*Department of Education, Psycology and Philosophy, University of Cagliari, via Is Mirrionis 1, Cagliari, Italy*
[2]*Faculty of Human Studies, University of Cagliari, via Is Mirrionis 1, Cagliari, Italy*

Keywords: Computing, Coding, Education, Scratch, BloP.

Abstract: During the last year, several countries, such as England, Finland and Italy, have decided to focus their new school curricula on computing, coding and IT. However, present teachers do not feel confident about moving to this new paradigm. Furthermore, coding would be relegated to be taught for just a few hours. Luckily, recent new tools have been designed to introduce young students to coding that can be also easily used by teachers to create engaging multimedia supports for their everyday lessons. In this paper, we describe several experiments that show how a new path from teachers to students and then back to teachers can be followed in order to build a new model of digital teaching. The proposed model does not require present teachers to become proficient IT experts.

1 INTRODUCTION

Coding and IT as fundamental skills for today students has had a great surge of attention during last year. In the United Kingdom, in September 2014, a new curriculum (UK Dept. of Education, 2013) has started in all maintained UK schools that puts, for the very first time, a strong emphasis on *computing*. In Italy, in September 2014, the Italian Government launched the “La Buona Scuola” (The Good School) collaborative initiative. The government asked Italian and foreign people to comment and suggest on the draft proposal for enhancing the Italian school system (Italian Dept. of Education, University and Research, 2014). In the proposal, they introduced more Arts, Economics, Foreign Languages and, at least, one hour of *coding* per week for all schools of all grades. In Finland, in November 2014, the National Board of Education has decided to reduce the time dedicated to handwriting in order to increase the time dedicated to *typing* (Finnish National Board of Education, 2014). In the United States, in December 2014, government acknowledged Code.org’s efforts to reach students with the Hour of Code and President Obama became one of the almost 100 million people to try *coding* this past year.

If teaching how to use type on a keyboard is something that can be possibly taught by almost all teachers in the first grades, teaching how to code is something different, involving knowledge that only a few of the UK and Italian teachers have in their portfolio.

Coding is the ability to analyse specific problems and to write computer programs to solve them by means of abstraction, logic, algorithms and data representation (Italian Dept. of Education, University and Research, 2014). Computing is the ability to evaluate and apply information technology, including new or unfamiliar technologies, to be a “competent, confident and creative users of information and communication technology” (UK Dept. of Education, 2013).

Not surprisingly, *more than half* of UK teachers feels not confident in their ability to teach computing skills, and only 7% feels very confident (TES and Nesta, 2014). The purpose of the Italian Dept. of Education is similarly ambitious, aiming to introduce internet connection and interactive whiteboards in all Italian classrooms and coding as a subject starting at the primary school.

But how can only a few hours of coding during the year prepare students so that they can benefit from the rigorous methods of computer science?

In our view, coding should *not be a separate subject* by itself but it should become an interdisciplinary instrument used by teachers of all

494 Federici S., Gola E., Brau D. and Zuncheddu A..
Are Educators Ready for Coding? - From Students Back to Teacher: Introducing the Class to Coding the Other Way Round.
DOI: 10.5220/0005491604940500
In *Proceedings of the 7th International Conference on Computer Supported Education* (CSEDU-2015), pages 494-500
ISBN: 978-989-758-107-6
Copyright © 2015 SCITEPRESS (Science and Technology Publications, Lda.)

subjects. Teachers from all subjects should be trained (or they should self-train) to teaching coding.

2 WHY IS CODING DIFFICULT TO TEACHERS

Preparing teachers for coding is not intrinsically difficult because of specific problems linked to coding. Coding is logical thinking, and logical thinking is at the base of all school subjects: languages and literature have structure, coordination, consequences, etc.; history has causes and consequences; geography, physics, science and technical subjects in general have changes in the status of something due to specific causes; mathematics has hypotheses and theses.

Nevertheless, as we already saw, the most part of teachers think that they are not be able to use a tool (whatever tool) to teach students how to code.

3 THE OTHER WAY ROUND

In recent years, something has changed and excellent and free tools have been designed. These tools are not only ideal to introduce students of all grades to coding but, more importantly, teachers can use them to build their interactive lessons for all school subjects. Therefore, learning how to use these tools can be of help to teachers, as they can add interactivity and multimedia to their lessons to engage their students and help them to grasp the subject. Learning how to use these tools, as we will show in a moment, is really easy, even for teachers of non-technical subjects and these tools are also easy and fun to be used by students. Teachers can then introduce students to the basic usage of these tool (basic lessons of coding) and students will help teachers in learning the most advanced features of these tools and the best strategies to build their interactive and multimedia lessons.

Instead of a simple, one-way teaching from teachers to students, we propose a new path from teachers to students and then back to teachers. This new path will be easier to follow by teachers and, more importantly, more proficuous, as the coding activity will not be relegated to a subject in its own for just a few hours a week. All subjects will teach lessons in which the necessary coding instructions will be explained so to better understand the main subject of the lesson. More importantly, by explaining a subject in a coding-driven way, the internal mechanisms of the subject will be more clear and under the eyes of the students.

If teachers are not ready for coding, today students certainly are (Tapscott, 1998). They were born with digital devices in their hands and they love using them. In the following, we will be showing that a new path from teacher to students to teacher is now possible.

3.1 What Do We Need?

In order to have a single tool that can be fruitfully used by both students and teachers we need a tool that:

1. allow students of all ages to learn how to code
2. allow students of all ages to learn all the important mechanisms of coding
3. allow students of all ages to develop interesting digital products that they can share online with their friends
4. is lightweight and multiplatform, so that it will easily runs on all OSs and all kind of devices (PC, tablets, smartphone) even not very recent
5. allow teachers to create interactive and multimedia supports to their lessons
6. is easy use so that even non-technical teachers can quickly learn how to use it

But which is the most important feature among the ones listed above? Even if the goal of current national projects is to allow students to learn how to code (point 1), the most important feature, from what we have said, is clearly number 6, closely followed by number 5. Indeed, if teachers will not be able to quickly grasp how to use this tool, and will not be deeply involved in its usage, the whole project will quickly fade away.

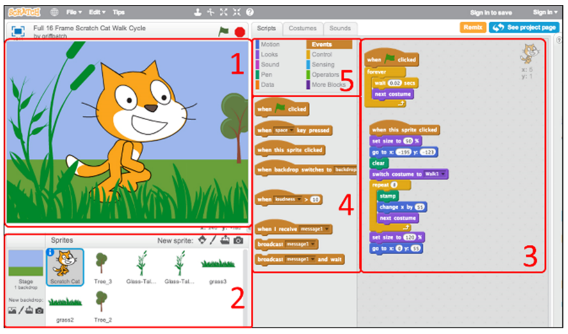


Figure 1: Scratch 2.0.

The basic components of such a tool are available today thanks to new programming tools developed during the last decade based on the metaphor of *construction blocks* (*block languages*, Federici and

Gola, 2014) in order to allow even small children, starting at age of 8, to learn how to code. One of the most famous and successful of those tools is Scratch (figure 1), developed by the Lifelong Kindergarten Group of the Multimedia Labs of MIT (Resnick et al, 2009).

Scratch's interface has a stage (area 1, in figure 1) on which several different sprites (area 2) perform their scripts (area 3) described by a predefined set of possible actions represented by coloured blocks in the palette (area 4). Blocks are grouped in categories (area 5). Blocks of a given category have the same colour and allow the sprite to perform similar actions.

Scratch, and its sibling tools, such as Snap (http://snap.berkeley.edu), Code.org Studio, Star Logo (http://education.mit.edu/projects/starlogo-tng), all have the basic features that make then ideal tools to introduce students to coding. Indeed, they:

- are available in several different languages, so to make it less intimidating to non-English speaking countries
- have a predefined and always visible set of instructions (blocks), so that users do not have to remember what the building blocks of the language are
- have a simple mechanism to put blocks together, so that users can easily grasp how to build up complex instructions starting from simpler ones
- make available lots of easy-to-use interactive and multimedia mechanisms so that students can be easily engaged
- most of them run in a simple web browser, so that nothing must be installed in order to use them

Not all tools are ideal instrument for teachers to build their interactive lessons. To give a few examples, Code.Org Studio mechanisms are too simple, and StarLogo TNG is not available as an online tool. However, Scratch and Snap are very good candidates.

By using these tools, teachers can make their lessons clearer and more engaging, can create and maintain them very easily. On the other side, students can learn how to code in a goal-oriented fashion, by understanding how the digital lessons prepared by their teachers have been built starting from the basic building blocks of the language. Therefore, they understand, at the same time, how the subject explained is built right from the inside.

These goals can be reached if these tools are really easy to learn and, more importantly, if teachers can learn back from their students what their students have learned by themselves.

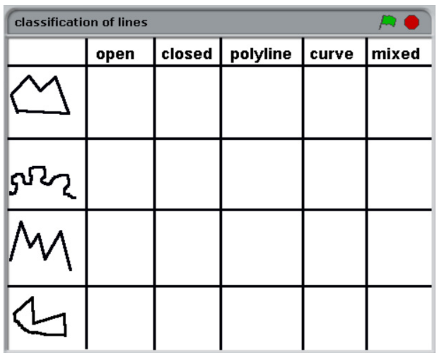
3.2 Introducing Teachers of Non-Technical Subjects to Coding. An Experiment

In order to see if Scratch could be easily learned - even by teachers of non-technical subjects-, we run a small experiment (Brau, 2011). The base hypothesis of such an experiment was that Scratch was intuitive enough so that, after an extremely short introduction, even teachers not very confident with technology could nevertheless grasp the basic mechanisms of the tool by being able i) to realize very simple behaviours very soon by themselves and, being guided, ii) to quickly build their first educational project. The secondary goal of the experiment was also to gather evidence about what should have been improved in Scratch in order to make it really intuitive and easy to use.

The teachers' actions were recorded during a 2-hours session, in order to review their behaviours. Two of the teachers taught only language, arts, history and geography, so they did not teach technical subjects, whereas the other two taught also mathematics, logics and science. All of them reached the same results, even if the two "non-technical" teachers had said before starting that they did not feel confident at all using computers and that they didn't use indeed them in their everyday life. None of them had ever used a programming language nor had previously known Scratch.

After having illustrated a few projects that could be developed with Scratch to support teaching of several school topics (we only showed the behaviour of the projects, not the actual code), we told them something about Scratch: that Scratch allowed to show images, to move them, to make them react to the mouse, to show speech bubbles, to produce sounds, to make images disappear and show up again. The only elements of the Scratch interface that were illustrated were the Green Flag button (to start projects) and the Red Stop button (to stop projects). Then we asked them to make something specific happen, and left them alone, each seated at their own PC, without the possibility of asking for suggestions to us or to their colleagues, until they were actually able to make that thing happen.

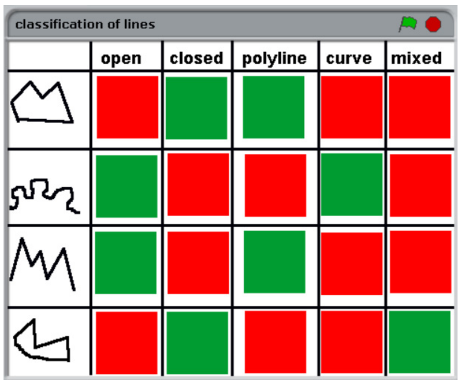
We asked them to perform 10 different tasks by making, in order, the Scratch cat: say "Hello!" in a speech bubble; say "Hello!" in a speech bubble for only 2 seconds; say "Hello!" in a speech bubble when the cat is clicked; say instead "I have been clicked" when the it is clicked; change look every time it is clicked; change look every time the space key is pressed; play the sound "meow" every time it is clicked; play the sound "Bravo!" every time it is clicked; say "Hello!" in a speech bubble for 2 seconds and then play the sound "Bravo!" every time it is clicked; have another cat on the Stage that's behaves exactly as the first one. All these tasks where performed by all four teachers in less than an hour and a half. Only the very first task took a little longer than the next ones (from a minimum of 15 minutes, for one of the technical teachers, to a maximum of 25 minutes, for the slowest of the non-technical teachers). To perform the first task they had to discover that there was more than one category of blocks and that to make the cat do something it was necessary to click a block in the palette or to drag it to the script area and then to click it. After each task was solved by all four teachers, we discussed the different solutions with the teachers. At the end of the whole session we illustrated to them what they had discovered: that in Scratch there are several "sprites" that can do different things on a "Stage"; that the blocks that allow the sprites to do something are organized in "categories" depending on their behaviour; that the blocks can be snapped together so that the sprites can have a whole series of behaviours with just one click; that the sprites can change their look; that the sprites can play different sounds; that we can record the sounds ourselves that will be then played by the sprites. Moreover some of the teachers had also discovered that the images of the sprites can be created by using an image editor internal to Scratch or that they can be created by shooting a picture with the internal webcam.


classification of lines

	open	closed	polyline	curve	mixed

Figure 2: classification of lines, interactive exercise for 2nd grade.

Then we guided them in building a very simple interactive project derived from an exercise in a science student book for the 2nd grade, namely the *classification of lines* (open vs. closed; polyline vs. curve vs. mixed) in which students must select the correct features for a few samples of lines by clicking the correct cells in a table (see figure 2) and getting visual feedback (figure 3).


classification of lines

	open	closed	polyline	curve	mixed
	🟥	🟩	🟩	🟥	🟥
	🟩	🟥	🟥	🟩	🟥
	🟩	🟥	🟩	🟥	🟥
	🟥	🟩	🟥	🟥	🟩

Figure 3: classification of lines, final result. Green is correct, red is wrong.

The result of the experiment was, in our view, a complete success. Not only the four teachers were able to follow us step by step, after having experienced Scratch for less than 90 minutes, completing a working project in 20 minutes. All of them said that they were also very interested in using Scratch to create interactive supports for their lessons (language, grammar, history, geography, mathematics, logics) as they had had no problems at understanding how to use it. Finally, and most important to us, they taught that Scratch could be used by teachers with no prior knowledge in coding to create educational projects to support their lessons.

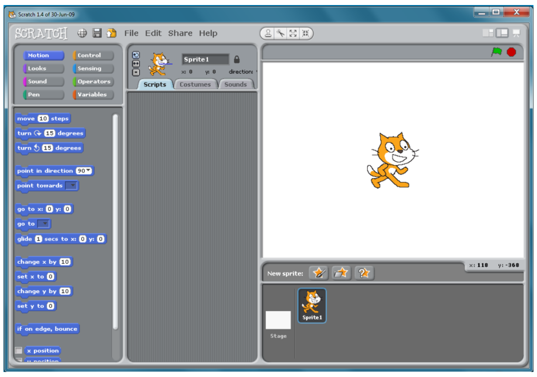


Figure 4: old Scratch 1.4 interface (stage to the right).

As an added bonus, we had some feedback on possible improvements to the Scratch interface, in order to make it more intuitive.

Interestingly enough, one of their suggestion, namely moving the Stage of Scratch 1.4 (figure 4) to the left and side of the interface, so to make the "natural inclusion" built-in in Scratch more clear (scripts are *made by* blocks that *belongs to* a sprite), has been included by the Scratch Team in the new version of Scratch (figure 5).

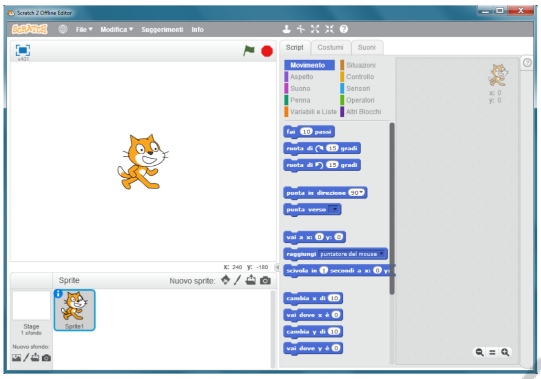


Figure 5: new Scratch 2.0 interface (stage to the left).

Finally, by analysing the teacher's behaviour when they were trying to perform the very first task, we discovered that, for a first-time user, it is very natural to try to make a sprite (e.g. the cat) doing something by dragging a block onto the sprite on the Stage. This is not allowed by Scratch: blocks dragged to the Stage are just sent back to the palette.

3.3 From Teacher to Students to Teacher. A Coding Experiment

The next point, that is that at least young students are interested in an instrument like Scratch and that they are able to improve their knowledge by themselves and then transfer this knowledge back to their teachers, was verified in a further experiment (Zuncheddu, 2015). The base hypothesis of this experiment was that the rich multimedia core of Scratch could ignite the interest of young students (two classes of the 5th grade) so that they would have started exploring the instrument by themselves and then would have reported their personal experiences back to the classroom. The experiment lasted for 8 weeks, a 2-hours session of Scratch 1.4 coding per week (we did not use the new Scratch 2.0 as it still has some bugs that could frustrate young children).

The first interesting outcome of the experiment was that, right after the very first lessons, several students used Scratch at home and then, at the next lesson, *reported to the classroom* what they discovered, that is how to draw images by using the internal image editor, how to record their voices by using the internal recorder, but also how to play sounds, how to make slideshow of images (shot with the internal webcam) with a background music.

The second outcome was instead that one of teachers of the two classes informally reported to us that she had had *detailed explanations from one of her students* about coding strategies that she had not immediately understood during the explanation.

As for the opinion of the students at the end of the experiment, the final outcome was that almost all of them (except three students out of 33) said that they would have liked to use Scratch *for all subjects*.

4 A BETTER TOOL

Even if Scratch, as we have already seen, is incredibly easy to use, sometimes, in order not to make it too complex to newbies, Scratch designers decided not to add several features, even if those features could make the creation of digital supports for teachers' lessons a lot easier. These features are not relevant to users using the tool just to learn how to code, but are very important to users that must use the tool to quickly develop a working project. Therefore, this make the present version of Scratch a tool that is not completely ideal for teachers.

Some of the important features that are missing in Scratch are:

- scenes: currently, if we want to create a project showing different situations (for example a project showing how different phrases can be split in meaningful parts) we have *to hide and show a lot of sprites* and tell to those sprites how they have to react to the user interaction for each different situation (in our example for each different phrase). By adding the automatic management of different situations, or "scenes", the number of scripts that is necessary for each sprite can be strongly reduced for the average project. This is also true for multiscene games, a kind of project that is very interesting to young students (Zuncheddu, 2015).
- more powerful undo: currently *only the last action* can be undone. This can get quickly very frustrating
- search for named elements (variables, messages, etc.) and strings: currently it is *not possible to find* the script where a given variable, message or string is used
- easier positioning: currently we are required to place sprites on the stage by specifying their *coordinates* (a mathematical notion that is not very handy for non-technical teachers). It would be better (Zuncheddu, 2015) if one could specify the position of a sprite also by putting marks on the stage and referring to them (like the X signs used on real stages)
- random behaviour: currently a Scratch project can behave differently each time it is run by

using *random numbers* (another notion that is not very handy for non technical teachers). It would be better if one could also choose from a small set of predefined random behaviours, for example one behaviour to arrange elements on the Stage in random order, another to choose and delete an item selected at random from a list, etc.

- dynamic interface: currently the interface has mostly fixed proportions (only the stage can be resized to at most two possible alternative sizes). It would be of help if the different areas (the script area, the palette, etc.) could occupy a larger area of the Scratch interface when needed.

5 BEYOND SCRATCH: SCRATCH MODS AND BLOP

Luckily enough, Scratch is an open source tool, so it can be modified in order to add useful features. Indeed, many *mods* (Scratch *mod*ification*s*) have been proposed by Scratch users starting with version 1.4 of Scratch. Indeed, Scratch users have proposed lots of *mods* (Scratch *mod*ification*s*) by starting with version 1.4 of Scratch. The first, and most famous among all mods, is BYOB, an excellent instrument that aims at making available a powerful programming language that, even in the form of a block language, have all the most advanced programming constructs so to be used also in computer science courses at the university level (Harvey and Mönig, 2010). BYOB has been recently re-implemented into Snap, a JavaScript version of BYOB running in all major browsers (Harvey, 2012).

The advanced programming constructs that are available in BYOB and Snap have made possible the development of the BloP platform (Federici and Gola, 2014). BloP (http://blocklanguages.org) is a tool that allows very easily to create new block languages, either the re-implementation of simplified versions of standard languages -such as C/C++, Logo, PHP, MySQL (Federici, 2011)- or completely new special purpose languages such as the Animated Sort Language (Federici and Stern, 2011). All this just using the simple blocks available in Scratch/Snap and embedding them in a *safe environment*, that is an environment that cannot be modified by students. This could allow the most motivated teachers to develop special-purpose programming languages for their specific subjects, such as, for example, a "Linguistic" programming language that could allow students to understand how correct sentences are built by putting together linguistic constructs (e.g. articles, names, verbs, etc.), or an "Artistic" programming language allowing students to learn how to mix colours together. The only limit, as always, is our fantasy.

6 SUPPORTING THE DIGITAL TEACHER: SCRATCHED, REMIXES, NON-CS STUDIOS

A new tool and a completely new way of teaching can be scaring to teachers not familiar with digital tools. Even if the primary school teachers involved in the Brau's experiment where all positive about the possibility of using Scratch as an educational instruments in their lessons, radically changing the way teachers teach is something that can get frustrating and teachers could soon abandon the tool if they do not feel supported in this process. Involving the students can be a great form of support for what concerns how to use the tool, but teachers should have a substantial basis of sample projects and examples for all possible subjects and levels from which to start to develop their own projects. *ScratchEd* (http://scratched.gse.harvard.edu), the website dedicated to Scratch educators that let teachers be in touch with other teachers and share resources and stories about using Scratch for computer science lessons, can be a great resource for useful ideas about *teaching Scratch*. However, it is not the ideal source for teachers of other subjects looking for inspiration and strategies about *teaching with Scratch*.

Luckily, many non-computer science teachers already use Scratch and share their projects on the Scratch website (http://scratch.mit.edu). Projects shared on the website can be freely downloaded and modified (*remixed*, in Scratch terminology) to create our own versions. During the last 2 years, we daily browsed the Scratch projects uploaded to the website in order to collect educational projects designed by teachers or by students to teach specific arguments for school subjects for all levels. We then arranged the collected projects in 10 "studios" ("studio" is the scratch name for a collection of projects) about "Language" (about 250 projects), "Foreign Languages" (150), "History" (100), "Geography" (150), "Music" (200), "Mathematics" (700), "Sciences" (1300), "Logics" (400), "Physics" (500), "Arts" (1000). Even if not all the about 4000 projects we collected are perfect and well designed, there are very good projects among them and, moreover, they are a very small part of the more than 8 million projects available on the Scratch website.

How are teachers supposed to be able to browse 8 million projects? When becoming a member of the Scratch website, teachers will soon be able to spot those members that share projects interesting to them. By becoming a *follower* of those members they will be able to know when they share further -hopefully interesting- projects but also when they appreciate projects by other members that, likely, will be of interest also to them. That is how we built our 4000 educational projects collection.

7 CONCLUSIONS

Coding is an essential skill for young generations. Block languages such as Scratch, Snap and BloP can become the pen of the 21st century. We have shown that very likely not only they could be easily learned by teachers but also that teachers could use them to create their digital lessons and, at the same time, train their students and be trained by them on coding.

The incredible amount of available resources and the positive feedback that teachers can get from their students is the key that make us confident that this new way of teaching could be a success.

REFERENCES

Brau, D., 2011. *The Digital Teacher: tools for easy creation of multimedia educational supports*. Faculty of Education, University of Cagliari. Degree Thesis. (in Italian).

Federici, S., 2011. User-Centered Computer Science: High-ceiling and Low-floor Approaches to Programming Languages and Algorithms. In *Proceedings of CSEDU 2011 - 3rd International Conference on Computer Supported Education*, SciTePress.

Federici, S., Gola, E., 2014. BloP: easy creation of Online Integrated Environments to learn custom and standard Programming Languages. In *SIREM-SIEL 2014, 1st Joint SIREM-SIel conference*. The Innovative LEDIpublishing Company.

Federici, S., Stern, L. 2011. A Constructionist Approach to Computer Science. In S. Barton et al. (Eds.), *Proceedings of Global Learn 2011*, AACE.

Finnish National Board of Education, 2014. *National Core Curriculum for Basic Education*. Finnish National Board of Education.

Harvey, B., 2012.The Beauty and Joy of Computing: Computer Science for Everyone. *Constructionism 2012*.

Harvey, B., Mönig, J., 2010. Bringing 'No Ceiling' to Scratch: Can One Language Serve Kids and Computer Scientists?. *Constructionism 2010*.

Italian Dept. of Education, University and Research, 2014. *The Good School*. Italian Dept. of Education, University and Research. (in Italian).

Nesta, TES, 2014. *Teachers feel unprepared for September's new computing curriculum*. http://www.nesta.org.uk/news/teachers-feel-unprepared-september's-new-computing-curriculum.

Resnick, M., Maloney, J., Monroy-Hernandez, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., Kafai, Y., 2009. Scratch: Programming for All. *Communications of the ACM, vol. 52, no. 11*.

Tapscott, D., 1998. *Growing up digital: the rise of the Net generation*. McGraw-Hill, New York.

UK Dept. of Education, 2013. *National curriculum in England. Computing programmes of study: key stages 1 and 2*. UK Dept. of Education.

Zuncheddu, A., 2015. *Scratch Me Five: an experiment of introduction of creative computing in Italian primary schools*. Faculty of Human Studies, University of Cagliari. Degree Thesis. (in Italian).