Leveraging Robot Programming to Foster Computational Thinking

Ilenia Fronza[1], Nabil El Ioini[1] and Luis Corral[2]
[1]*Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy*
[2]*ITESM / UAQ, E. Gonzalez 500, 76130, Queretaro, Mexico*

Keywords: Educational Robotics, Computational Thinking, Maze.

Abstract: In 2013, ACM recognized Computational Thinking (CT) as "one of the fundamental skills desired of all graduates". This means that, especially in liberal education environments, one of the challenges of CT courses is to motivate students who are discouraged upfront as they perceive programming as a difficult task. Applications that have tangible results typically stimulate students' interests. For instance, Educational Robotics (ER) is recognized as a tool to enhance higher order thinking skills and to facilitate teamwork. In this paper, we describe a course that has been designed to use ER (i.e., programming a maze-solving robot) to foster CT. Each activity of the course has been designed to foster specific CT skills and to contribute to CT assessment, which remains a challenge in CT research. We report the results of an experiment, in a liberal education environment, with a total of 13 ninth graders (15.4% M, 84.6% F).

1 INTRODUCTION

Computational Thinking (CT) consists of "the thought processes involved in formulating a problem and expressing its solution(s) in such a way that a computer - human or machine - can effectively carry out" (Wing, 2014). In 2013, ACM recognized CT as "one of the fundamental skills desired of all graduates" (Joint Task Force on Computing Curricula, ACM and IEEE Computer Society, 2013). Therefore, motivational concerns need to be addressed to engage the broadest audience possible (Settle et al., 2012), by reaching also students who do not consider themselves candidates for STEM disciplines (Prey and Weaver, 2013). For example, in liberal education environments, students are usually not inclined to STEM disciplines (Fronza et al., 2015); moreover, the difficulties in learning computer science (CS) usually frustrate students and lead many of them to avoid activities that require programming skills (Stamouli et al., 2004). Since there is a relationship between motivation and success in learning CS (Carbone et al., 2009), motivating students becomes of paramount importance (Alhazbi, 2016). To this end, CT can be used to show that programmers, beside coding, need for example to interact with others and look for ideas in order to solve problems with a computational strategy (Hambrusch et al., 2009).

Activities that have tangible results typically stimulate students' interests. Considering this, in the last few years several teaching approaches have been proposed which use concrete artifacts including mobile devices (Fronza et al., 2015) or robots (Sarmento et al., 2015). In particular, the usage of educational robotics (ER) is not recent and it is now recognized as a tool to enhance higher order thinking skills and to facilitate teamwork (Atmatzidou and Demetriadis, 2016). Unfortunately, one of the limitations of ER is represented by the acquisition costs of robots (Sarmento et al., 2015). Recently, low cost do it yourself (DIY) robots have been proposed to address this issue; these robots are built on top of open source hardware specifications and programmable using free software tools such as Scratch and Coffee (Resnick et al., 2009; Sarmento et al., 2015).

In this paper, we describe a course at high school level that has been designed to foster CT skills using ER. Each activity of the course has been designed to foster specific CT skills and to allow CT assessment, which represents a challenge in CT research (Moreno-Leon et al., 2015). Indeed, identifying what students are expected to learn in each phase of the course facilitates the introduction of CT in schools curricula, as an assessment methodology enables the evaluation of the success or failure of the course. In this paper we report the results of the first edition of the course, with 13 ninth graders (15.4% M, 84.6% F) in a liberal education environment.

Fronza, I., Ioini, N. and Corral, L.
Leveraging Robot Programming to Foster Computational Thinking.
DOI: 10.5220/0006310101090116
In *Proceedings of the 9th International Conference on Computer Supported Education (CSEDU 2017) - Volume 2*, pages 109-116
ISBN: 978-989-758-240-0

The paper is organized as follows: Section 2 provides background information; Section 3 details the structure of the course; Section 4 reports the results of the first edition of the course; Section 5 draws conclusions and provides directions for further research.

2 BACKGROUND

Computational Thinking. Ten years have passed since the publication of Jeannette Wing's seminal article on CT (Wing, 2006). During these ten years, media have been heavily promoting *coding and programming* (Crow, 2014); while this has represented an important change respect to teaching only computer literacy (Fronza et al., 2014), focusing only on coding ignores the broader aims of CT. Indeed, coding is a tool for supporting CT (Bateman, 2014), but the operational definition of CT (Table 1) involves concepts, practices, and perspectives (Brennan and Resnick, 2012) that are not strictly related to coding and therefore can be fostered using other tools and activities (Catlin and Woollard, 2014). The same holds for the skills included in the definition of CT (Table 2) proposed in 2011 in (ISTE and CSTA, 2011).

Table 1: The three CT dimensions (Brennan and Resnick, 2012).

CT dimensions	Definition
Computational concepts	• Sequences • Loops • Events • Parallelism • Conditionals • Operators • Data
Computational practices	• Being incremental and iterative • Testing and debugging • Reusing and remixing • Abstracting and modularizing
Computational perspectives	• Expressing • Connecting • Questioning

Research has recently focused on defining curricula for teaching CT. However, gaps still exist that call out for empirical inquiries, in particular in the K-12 context (Grover and Pea, 2013) to understand what CT concepts should be learned in the different stages of schooling (Voogt et al., 2015). Moreover, there is a need to teach learners how to use CT within the disciplines (Hemmendinger, 2010). Another part of current research deals with the need of evaluating the effectiveness of CT courses (Brennan and Resnick, 2012; Moreno-Leon et al., 2015; Werner et al., 2012).

Table 2: Skills improved by CT (ISTE and CSTA, 2011).

	Skill	Definition
1	Data collection	Gather appropriate information
2	Data analysis	Make sense of data, find patterns, and draw conclusions
3	Data representation	Depict and organise data in appropriate graphs, charts, words, or images
4	Problem decomposition	Break down a problem into smaller, manageable parts
5	Abstraction	Reduce complexity to define main idea
6	Algorithms and procedures	Find a series of ordered steps to solve a problem
7	Automation	Have computers or machines do repetitive or tedious tasks
8	Simulation	Run experiments using models
9	Parallelization	Organise resources to simultaneously carry out tasks to reach a common goal

Educational Robotics (ER). ER allows to reach a wide audience, as robots typically stimulate students' interests (Catlin and Woollard, 2014). Besides gaining students' attention, ER provides opportunities to teach CT; indeed, ER integrates several areas, such as programming and engineering design (Shoop et al., 2016). Moreover, working with robots is normally done in groups; this creates a powerful environment to foster CT practices and perspectives, such as connecting with others and expressing ideas (Table 1).

Acquisition costs of robots have been recently reduced by the availability of relatively cheap hardware, which even allows to build robots independently. For this reason, research has been recently exploring the possibility to program robots using free and open source programming tools languages. Among them, visual programming languages (VPL) are becoming one of the most common choice for ER, given their popularity in schools (Resnick et al., 2009; Sarmento et al., 2015) and their ability to foster CT (Brennan and Resnick, 2012). For instance, Sarmento et al. (2015) propose a solution (called Coffee) in which a VPL is used to program robots. The same paper provides an overview of the existing robots and programming environments. Nevertheless, the paper does not clarify how exactly the proposed activities contribute to the development of CT skills.

3 COURSE DESCRIPTION

CT courses need to be able to engage the broadest audience possible (Settle et al., 2012; Wing, 2006), by reaching also the students who do not consider themselves candidates for STEM disciplines (Prey and Weaver, 2013). This means that courses need to be able to motivate possible uninterested learners in the classroom, for example in a liberal education context. Moreover, it is clearly important to avoid losing the gained attention; this might happen, for example, if the students perceive the programming part as not accessible to them (Alhazbi, 2016), which is not an off chance in a liberal education context where the main disciplines can be, for example, humanities and social sciences.

The goal of the course proposed in this paper is to foster CT skills through ER. The course leverages students' general curiosity for robots to gain their interest (Catlin and Woollard, 2014). Moreover, CT is used to show that, with a computational strategy, programming a robot can be an accessible task; CT is also used as a means to show that programmers, beside coding, need for example to interact with others to solve problems with a computational strategy (Hambrusch et al., 2009). The remaining part of this section describes tools and structure of the course, which have been chosen to foster CT skills.

3.1 Tools

As suggested in (Sung and Samuel, 2014), our course pursues technology-independent learning outcomes. Indeed, in our course programming a robot is not the main focus, but it is used as a tool to support the cognitive tasks (i.e., CT) involved in ER (Fronza et al., 2015). In this view, preference was given to tools that allow fostering CT skills.

As a robot, we chose *mBot*[1], which is proposed in the market as an all-in-one solution to enjoy the hands-on experience of programming, electronics, and robotics. The main reasons of our choice are:

- It is relatively cheap (i.e., around 100 USD). This was one of our most important requirements, because it increases the possibility to activate this course in schools. Moreover, affordable robots can be bought by the families of those students keen on programming at home.
- It can be programmed with *mBlock*[2], a VPL inspired by Scratch 2.0, which is considered a good tool to foster CT (Brennan and Resnick, 2012). Moreover, Scratch is very popular in schools, and this reduces the time dedicated to learning the tool set.
- mBot can be programmed also using the Arduino IDE. Therefore, learners can progress on to Arduino programming, and more expert students can program directly using the Arduino IDE.

The goal was to build a maze and instruct the robot to find its way out of it. To build the maze, we printed on two A0 posts a grid of black lines (3 cm width) crossing every 22.5 cm. We used home made 3D-printed pegs[3] to hold up cardboard walls. The resulting maze is shown in Figure 1. This design was chosen to foster creativity and experimentation, since it allows the students to design many mazes just moving walls. Moreover, it is light and portable, thus it can be used in different rooms easily. Finally, it is relatively cheap, which increases the possibility to activate this course.

For an initial training on using mBot and its sensors, we used a simple follow line example provided as part of the mBot kit.

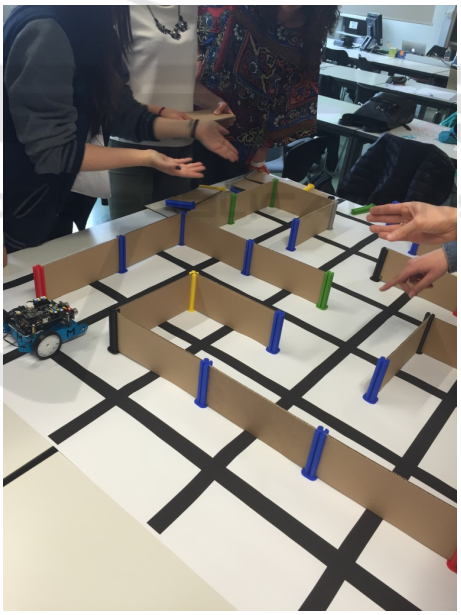

Figure 1: The tool set used in the course.

3.2 Structure of the Course

The course consists of four phases (10 hours in total, split in 3 days). In each phase, particular importance is given to organized work and documentation. By the end of the course, the participants are expected to

[1]http://learn.makeblock.com/en/mbot/
[2]http://www.mblock.cc
[3]http://www.thingiverse.com/thing:1169585

present their final solution in front of the class. This section explains all the phases of the course, and details how each phase has been designed to foster specific CT skills.

1. Introduction to mazes (1 hour). In this phase, we provide a brief historical outline about mazes. According to J. Wing's definition, applying CT does not only mean solving a problem, but also formulating it (Wing, 2014). The goal of this phase is to let the participants brainstorm to formulate the problem themselves, step by step, instead of having a problem provided by the teacher. Moreover, in this phase the students start collecting, analysing and representing relevant data (Table 3). For instance, the students should ask: who is walking in the maze? A robot, or a human? Which types of mazes exist? Is it always possible to exit a maze?

Table 3: CT skills (Brennan and Resnick, 2012) fostered during each phase of the course.

	Phase			
	1	2	3	4
Data collection	✓			✓
Data analysis	✓			✓
Data representation	✓		✓	✓
Problem decomposition		✓	✓	✓
Abstraction		✓	✓	✓
Algorithms and procedures		✓	✓	✓
Automation			✓	✓
Simulation			✓	✓
Parallelization				✓

2. Solving mazes without robots (1 hour). In this phase, each student solves a random generated maze (printed on paper) and annotates the algorithm (Figure 2). Afterwards, we distribute a second maze so that students can understand that their algorithms are customized to one single maze. This is used to open a discussion on how robots can be instructed to solve mazes, independently of their structure.

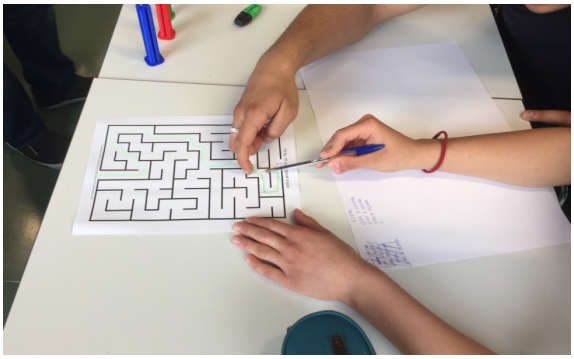

Figure 2: Finding an algorithm to solve mazes.

As shown in Table 3, this phase exercises multiple CT skills. First, the students need to decompose the problem to understand how to exit the maze; second, they need to grasp the major concepts that define the problem (i.e., abstraction) and that can be applied to all mazes. Third, they start working on an algorithm.

3. Learning the tool set (2 hours). In this phase, the participants familiarize with the tool set that they will use during the fourth phase. In particular, the goal is to make students aware of the technical possibilities (for example, to follow lines using sensors, or to "see" walls in the maze) that they can use to design their solution of the problem.

First, the robot is introduced by describing its structure, sensors, and functionality. Afterwards, as an introductory exercise, a follow line problem is proposed: the students have to identify the problem, explain a high-level solution, and then structure a detailed algorithm. The designed solution is implemented by the teacher (and projected to the entire class) following the students' ideas, and new computational concepts are introduced meanwhile. Therefore, in this phase, the students start working on algorithms, automation and simulation (Table 3). Moreover, data representation skills are exercised. For instance, they sketch a visualization of a possible algorithm to follow a line: 1) the robot goes straight when both the sensors "see" the black line, 2) it turns right when the left sensor does not see the black line, and 3) it turns left when the right sensor does not see the black line.

4. Programming a maze-solving robot (6 hours). In this phase, the students find the solution for the problem, which consists of coding the robot to exit a maze. They are free to choose their own approach, such as following lines or checking the presence of obstacles (i.e., maze walls); the instructors act as moderators when needed, and help in designing the solution or in solving technical problems.

Since ER facilitates teamwork (Atmatzidou and Demetriadis, 2016), in this phase the students are divided in teams. The idea is to maximize team size to minimize the number of robots, in order to make the course financially more accessible to schools. According to the available literature, the optimal group size is five (Hackman and Vidmar, 1970); in software development, though, a common rule suggests "seven plus or minus three" members (Holzinger et al., 2005). In bigger teams, the management of the communication process becomes more difficult; indeed, after seven members in a team, each additional member reduces decision effectiveness by 10% (Blenko et al., 2013). For this reason, we create teams of four to seven students, and the teachers act

as moderators in the teams when needed. Working in teams also exercises the parallelization skill (ISTE and CSTA, 2011), as the students need to divide the tasks and coordinate their activities.

Each team works autonomously to select a strategy just using pen and paper, without computer and robot. During this activity, we highlight the importance of the resulting design document: it will guide the coding activities and will be used as a reference for testing. Then, the teams start coding in short iterations; this means that each team can use the maze to test the solution, whenever its members think to have an improved solution, or want to check if an issue has been solved. This has the advantage of encouraging simulations to discover and fix errors sooner; indeed, robots ease this activity, as most of the times they provide a direct evidence of the correctness of the solution. For example, if the robots hits a wall, then its sensors might not be used correctly.

Team competitions are used to improve social engagement (Dicheva et al., 2015): during this phase, the teams compete to find a solution and can find different solutions to the same problem. The only rule is that, when one team is testing its solution (there is just one maze in the room for all the teams), all the students should stop and participate in the test (Figure 3); this way, they can learn from others' mistakes, or find new ideas and change their own solutions.

Figure 3: All the students check the solutions together, and try to get new ideas from others' success or mistakes.

This phase has been designed to stimulate CT dimensions (Table 1) and CT skills (Table 3):

- Computational perspectives. Working in teams fosters the ability of expressing ideas. Moreover, observing the other teams' tests improves the ability of *questioning*, i.e, criticising a solution or asking for explanations.
- Computational practices. Students work incrementally and iteratively; moreover, they can reuse previous code, such as the code written during the third phase of the course.
- CT skills. This course covers all the CT skills (ISTE and CSTA, 2011). Indeed, students need to collect, analyse and represent the information needed to solve the problem. Moreover, they learn how to abstract and decompose the problem to find an algorithm. Finally, they use the robot to automate the solution, simulate frequently and work in parallel in the team.

In the next section we detail the results of the first edition of the course.

4 RESULTS

We experimented the course with 13 ninth graders (15.4% M, 84.6% F), in a liberal education environment. In this section, we describe our results in terms of the students' performance and the tool set adequacy.

Students' Performance. ER succeeded in gaining the participants' interest: all of them were curious about the topic, but at the same time they were not sure to be able to code the robot. Given this curiosity, all the students participated actively in the brainstorming session of the first phase of the course. As expected, the main questions that emerged in this phase were: Which types of mazes exist? Is it always possible to exit a maze? Can we find existing algorithms to solve a maze? What is the difference between a person and a robot in solving a maze?.

During the second phase, the discussion focused on possible algorithms. The brainstorming session led to the idea of the so called left-hand (or right-hand) rule: by keeping the left-hand (or the right-hand) in contact with one wall of the maze the solver is guaranteed to reach the exit. This idea made students brainstorm on the meaning of "keeping a hand in contact with a wall" for a robot. Since at this point the students did not know the robot's characteristics, after some speculations on the topic they decided to solve the issue later.

During the third phase the participants learned all the tools they could use to solve the problem. Also, they learned how to design a solution and represent information. For instance, Figure 4 shows how they sketched the solution for the follow line task: the robot goes straight when both the sensors are on the black line, and it adjusts its direction (to the left or to the right) otherwise.

In the fourth phase, two teams of 6 (5 F and 1 M) and 7 (6 F and 1 M) students were created, and

one instructor was assigned to each team as a moderator. Both the teams managed team dynamics constructively; it should be noted here that these students were trained to teamwork, as they usually work in teams during project activities at school. The environment was relaxed and playful, and all the team members were able to discuss and collaborate. During the tests of the other team, the participants were curious and asked questions about the solution; the testing team, instead, was usually proud of showing the achieved results. Mistakes (e.g., the robot crashing against a wall) were taken as cheerful moments. Both the teams did not manage to implement an algorithm similar to the left or right-hand rule. Instead, they designed the two following solutions.

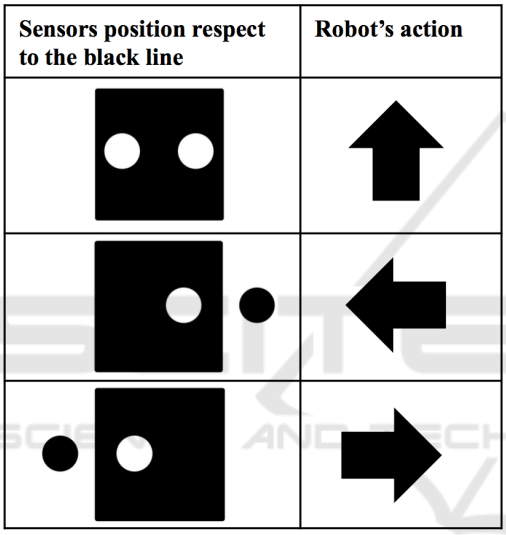

Figure 4: Participants' sketch of the main idea of the algorithm to follow a line.

Team A (7 students) tried to implement the left-hand rule algorithm, using the sensor in front of the robot as a "hand", which means that the robot should turn left "sometimes" to check for the presence of a wall. The team tried to let the robot turn left at each cross of two black lines: the robot was starting with one sensor on the black line and one outside it, and was turning left when both the sensors were on the black line (i.e., when two black lines were crossing). However, the team could not handle the complexity of the algorithm and decided to adopt another solution in which the part of the code that recognised crosses was reused. The final solution included a hard coded logic, as a specific algorithm was implemented for a specific maze: go straight until three crosses are counted, turn left, go right until tow crosses are counted, and so on. The solution, however, did not work properly, as the same cross was counted multiple times. Indeed, the sensors were detecting the same line many times. Team A did not have enough time to fix the issue, but the logic behind the solution was correct, even if it was customised to a single maze.

Team B (6 students) decided, first, to implement the following algorithm: go straight until an obstacle is detected, go left if there is no obstacle on the left, otherwise go right, or go back if there are obstacles on left and right. Being not able to manage proximity when looking for walls, and because of lack of time, also this team decided to implement a hard coded solution. The algorithm is similar to team A's one, but the robot turns left or right after a certain distance (which is specific of the considered maze). This solutions is logically correct but, respect to team A, it does not use sensors at all.

Even if both the teams have produced an hard coded solution, it should be considered that having more time, which was not possibile due to school constraints, would have probably allowed the participants to solve technical issues and implement the general solutions initially designed. Moreover, programming the robot was not the main focus of the course, but it was used as a tool to support the cognitive tasks involved (i.e., CT). Also, it is important to emphasise that all the participants were happy of their initial solution; some of them, indeed, even asked to extend the course to do that.

Despite the recent research in this field, assessing the learning of CT remains a challenge (Meerbaum-Salant et al., 2010; Moreno-Leon et al., 2015). Currently, programming environments are most often used as the basis for assessment (Werner et al., 2012). For instance, when adopting Brennan and Resnick's definition (Table 1) of CT, code inspections can reveal if students have been using computational concepts timely (Brennan and Resnick, 2012). Other forms of assessments, such as interviews or tasks, are used when adopting other CT definitions (Table 2) or to capture the other dimensions of CT (Table 1). At the end of the course, we analysed the final projects' blocks to provide an evidence of the computational concepts that have been used by students (Brennan and Resnick, 2012). Code inspections revealed that both the teams used the following concepts: sequences, loops, events, conditionals, operators, and data. Only parallelism was not used, as expected. To take advantage of all the available time, which was limited to ten hours, we chose not to insert additional tasks or interviews to complement our assessment.

Tool Set Adequacy. In general, the tool set was adequate to the course needs. The home-made maze succeeded in being light and portable. Moreover,

the removable walls encouraged experimentation: the students could create smaller and simpler mazes, or ad-hoc structures to test their code (e.g., dead-end streets). One defect of our cardboard walls is that they can be easily damaged by pegs, if they are inserted frequently. For this reason, plastic walls would be preferable. As for the robot, during the course we did not encounter relevant problems or damages. The main problem was represented by the programming language, mBlock, which does not include some original commands of Scratch (e.g., "when Flag clicked"). For this reason, even expert Scratch users need some time to learn some concepts. Moreover, it is almost impossible for beginners to figure out how to use sensors, button, or IR input; indeed, the available documentation of mBot explains only basic concepts, and needs some technical knowledge. For this reason, teachers' support is fundamental for the success of the course.

5 CONCLUSION AND FUTURE WORK

In this paper, we described a course at high school level that has been designed to foster Computational Thinking (CT) skills using Educational Robotics (ER), with the main goal of reaching also the students who do not consider themselves candidates for STEM disciplines (Prey and Weaver, 2013), for example in liberal education environments. We leveraged students' general curiosity for robots (Catlin and Woollard, 2014) and we chose tools and strategies (e.g., visual programming languages and teamwork) to ease the learning process. Moreover, CT is used to show that a good computational strategy can facilitate the programming part (Hambrusch et al., 2009). We designed each phase of the course to foster specific CT skills and to contribute to CT assessment (Meerbaum-Salant et al., 2010; Moreno-Leon et al., 2015).
We experimented the course in a liberal education environment, with a total of 13 ninth graders (15.4% M, 84.6% F). The accomplishments of the teaching program are reflected by the solutions that the students developed, under the supervision of the teaching staff. Both teams have produced hard coded solutions; nevertheless, having more time available would have probably allowed the participants to solve technical issues (mostly caused by the scarce documentation of the robot) and to implement the general solutions initially designed. All the participants, however, were proud of their solutions; some of them, indeed, asked to extend the course to continue working on the problem and implement the initially designed solutions. In a future edition of this course, having more time available would also allow to insert additional tasks or interviews to check that all the phases are fostering the specific CT skills they were designed for.
In general, results of case studies are difficult to generalize (Wohlin et al., 2000). In our case, more analysis is required to generalize to other situations; therefore, further case studies should be conducted involving students of different (liberal) curricula. Moreover, other case studies would help in extending and validating the assessment strategy.

REFERENCES

Alhazbi, S. (2016). *Active Blended Learning to Improve Students' Motivation in Computer Programming Courses: A Case Study*, pages 187–204. Springer International Publishing, Cham.

Atmatzidou, S. and Demetriadis, S. (2016). Advancing students computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, 75, Part B:661 – 670.

Bateman, K. (2014). Let's not forget the computing curriculum's bigger picture: Computational thinking. http://www.computerweekly.com/feature/Lets-not-forget-the-computing-curriculums-bigger-picture-Computational-thinking. Accessed: 21/02/2017.

Blenko, M., Mankins, M., and Rogers, P. (2013). *Decide and Deliver: Five Steps to Breakthrough Performance in Your Organization*. Harvard Business Press.

Brennan, K. and Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *2012 Annual Meeting of the American Educational Research Association, Vancouver, Canada*, pages 1–25, Vancouver, Canada. AERA.

Carbone, A., Hurst, J., Mitchell, I., and Gunstone, D. (2009). An exploration of internal factors influencing student learning of programming. In *Proceedings of the Eleventh Australasian Conference on Computing Education-Volume 95*, pages 25–34. Australian Computer Society, Inc.

Catlin, D. and Woollard, J. (2014). Educational robots and computational thinking. In *Proceedings of 4th International Workshop Teaching Robotics, Teaching with Robotics and 5th International Conference Robotics in Education*, pages 144–151, Padova, Italy.

Crow, D. (2014). Why every child should learn to code. http://www.theguardian.com/technology/2014/feb/07/year-of-code-dan-crow-songkick. Accessed: 21/02/2017.

Dicheva, D., Dichev, C., Agre, G., and Angelova, G. (2015). Gamification in education: a systematic mapping study. *Educational Technology & Society*, 18(3):1–14.

Fronza, I., El Ioini, N., and Corral, L. (2015). Students want to create apps: Leveraging computational thinking to teach mobile software development. In *Proceedings of*

the 16th Annual Conference on Information Technology Education, SIGITE '15, pages 21–26, New York, NY, USA. ACM.

Fronza, I., El Ioini, N., Janes, A., Sillitti, A., Succi, G., and Corral, L. (2014). If I had to vote on this laboratory, I would give nine: Introduction on Computational Thinking in the lower secondary school: Results of the experience. *Mondo Digitale*, 13(51):757–765.

Grover, S. and Pea, R. (2013). Computational thinking in k–12. a review of the state of the field. *Educational researcher*, 42(1):38–43.

Hackman, J. R. and Vidmar, N. J. (1970). Effects of size and task type on group performance and member reactions. *Sociometry*, 33:37–54.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., and Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *SIGCSE Bull.*, 41(1):183–187.

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads*, 1(2):4–7.

Holzinger, A., Errath, M., Searle, G., Thurnher, B., and Slany, W. (2005). From extreme programming and usability engineering to extreme usability in software engineering education. In *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, volume 1, pages 169–172.

ISTE and CSTA (2011). Operational definition of computational thinking for k-12 education. http://csta.acm.org/Curriculum/sub/CurrFiles/ CompThinkingFlyer.pdf. Accessed: 21/02/2017.

Joint Task Force on Computing Curricula, ACM and IEEE Computer Society (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, New York, NY, USA. 999133.

Meerbaum-Salant, O., Armoni, M., and Ben-Ari, M. M. (2010). Learning computer science concepts with scratch. In *Proceedings of the Sixth International Workshop on Computing Education Research*, ICER '10, pages 69–76, New York, NY, USA. ACM.

Moreno-Leon, J., Robles, G., and Roman-Gonzalez, M. (2015). Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED-Revista de Educacin a Distancia*, 46:1–23.

Prey, J. C. and Weaver, A. C. A. (2013). Fostering gender diversity in computing. *Computer*, 46(3):22–23.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11):60–67.

Sarmento, H. R., Reis, C. A. S., Zaramella, V., Almeida, L. D. A., and Tacla, C. A. (2015). Supporting the Development of Computational Thinking: A Robotic Platform Controlled by Smartphone. In *Proceedings of the Second International Conference on Learning and Collaboration Technologies*, volume 9192 of *Lecture Notes in Computer Science*, pages 124–135. Springer International Publishing.

Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., and Wildeman, B. (2012). Infusing computational thinking into the middle- and high-school curriculum. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 22–27, New York, USA. ACM.

Shoop, R., Flot, J., Friez, T., Schunn, C., and Witherspoon, E. (2016). Can computational thinking practices be taught in robotics classrooms? In *International Technology and Engineering Education Conference*, pages 1–15. Carnegie Mellon Robotics Academy/University of Pittsburgh.

Stamouli, I., Doyle, E., and Huggard, M. (2004). Establishing structured support for programming students. In *Frontiers in Education, 2004. FIE 2004. 34th Annual*, pages F2G–5. IEEE.

Sung, K. and Samuel, A. (2014). Mobile application development classes for the mobile era. In *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*, ITiCSE'14, pages 141–146, New York, NY, USA. ACM.

Voogt, J., Fisser, P., Good, J., Mishra, P., and Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4):715–728.

Werner, L., Denner, J., Campe, S., and Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 215–220, New York, NY, USA. ACM.

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

Wing, J. M. (2014). Computational thinking benefits society. http://socialissues.cs.toronto.edu. Accessed: 21/02/2017.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA.