COMPUTATIONAL THINKING TEST: DESIGN GUIDELINES AND CONTENT VALIDATION

Marcos Román González

Universidad Nacional de Educación a Distancia - UNED (SPAIN)

Abstract

We live in a society full of digital and programmable objects. In this context, being code-literate involves an inescapable requirement for any citizen of the XXI century. We believe that a person is code-literate when the ability to read and write in the language of computers and to think computationally has been developed. So it is not surprising that computational thinking (CT) is being located at the focus of educational innovation as a set of problem solving skills to be acquired by new generations of students. However, we still lack international consensus on a definition of CT, nor a clear idea of how to incorporate CT to our education systems at various levels. Similarly, there is a striking gap about how to measure and assess CT. In reply, this paper presents the design of a Computational Thinking Test aimed at Spanish students between 12 and 13 years old (grades K-7 & K-8): we describe the guidelines on which whole test and each of its items have been designed, as well as the content validation process through expert's judgment procedure. Through this process, the initial version of the test (40 items length) was depurated to a 28 items version, which is currently being applied to the target population. Finally, possible limitations of the test and possible concurrency of the same with other international evidence on computational thinking assessment are discussed.

Keywords: Computational thinking, computational thinking test, code literacy, computer science education.

1 INTRODUCTION

We live in a digital ecosystem full of programmable objects driven by software [1]. In this context, being able to handle the language of computers emerges as an inescapable skill, a new literacy, which allows us to participate fully and effectively in the digital reality that surrounds us: it is about to *'program or be programmed'* [2]. The term 'code-literacy' has recently been coined to refer to the process of teaching and learning of read-write with computer programming languages. Thus, it is considered that a person is code-literate when is able to read and write in the language of computers and other machines, and to think computationally [3]. If code-literacy refers ultimately to a new read-write practice, the concept of computational thinking (CT) refers to the underlying problem-solving cognitive process that allows it.

In this framework, it is not surprising that there is renewed interest in many countries to introduce CT as a set of problem-solving skills to be acquired by new generations of students; even more, CT is becoming viewed at the core of all STEM disciplines [4]. Although learn to think computationally has long been recognized as important [5], as computation has become pervasive, underpinning communication, science, culture and business in our society [6], CT is increasingly seen as an essential skill to create rather than just consume technology [7]. The recent decision to introduce computer science teaching from primary school onwards in the UK reflects the growing recognition of CT importance [8].

However, there is still little consensus on a formal definition for CT, and discrepancy in beliefs of how it should be integrated into educational programs [9]. Almost ten years ago, in 2006, Jeanette Wing's foundational paper defined that CT "involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" [10]. CT's essence is thinking like a computer scientist when confronted with a problem. But this first generic definition is being revisited and specified in successive attempts over the last few years, still not reaching an agreement. So, in 2008 Wing clarified, "computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent" [11]. Four years later, this definition is simplified by Aho, who declares CT as the thought processes involved in formulating problems so "their solutions can be represented as computational steps and algorithms" [12]. Moreover, in 2011 the Computer Science Teachers Association (CSTA) and the International Society for Technology in

Education (ISTE) developed an operational definition of computational thinking that provides a framework and common vocabulary for Computer Science K-12 educators: computational thinking is a "problem-solving process that includes (but is not limited to) the following characteristics: formulating problems in a way that enables us to use a computer and other tools to help solve them; logically organizing and analyzing data; representing data through abstractions such as models and simulations; automating solutions through algorithmic thinking; identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources; generalizing and transferring this problem solving process to a wide variety of problems" [13].

Even Google has provided its own definition of CT as "a set of problem-solving skills and techniques that software engineers use to write programs that underlie the computer applications you use such as search, email, and maps. However, CT is applicable to any subject (…) Specific computational thinking techniques include: problem decomposition, pattern recognition, pattern generalization to define abstractions or models, algorithm design, and data analysis and visualization" [14].

As a synthesis, Grover & Pea have recently contributed with a complete review of the state of the field [15], highlighting the lack of international consensus on a definition of CT. They conclude that the following elements are now widely accepted as comprising CT and form the basis of curricula that aim to support its learning as well as assess its development:

- Abstractions and pattern generalizations (including models and simulations);
- Systematic processing of information; Symbol systems and representations;
- Algorithmic notions of flow of control;
- Structured problem decomposition (modularizing);
- Iterative, recursive, and parallel thinking;
- Conditional logic;
- Efficiency and performance constraints;
- Debugging and systematic error detection

Similarly, there is a striking gap about how to measure and assess CT, fact that must be addressed. Without attention to assessment, CT can have little hope of making its way successfully into any curriculum. Furthermore, to judge the effectiveness of any curriculum incorporating CT, measures that would enable educators to assess what the child has learned need to be validated [15].

Most recent research addressing questions of CT assessment, such as *'Fairy Assessment in Alice'*, has used either student-created, or predesigned programming artifacts to evaluate students' understanding and use of abstraction, conditional logic, algorithmic thinking, and other CT concepts to solve problems [16]. In South Africa, a *Computational Thinking Framework (CTF)* has been developed to serve as a foundation for designing and evaluating CT materials (e.g. Light-Bot), and student assessments [9]. This research group has also studied first year college student performance in a *Test for Computational Thinking* [17]; the questions used in this assessment were sourced from the Computer Olympiad 'Talent Search' papers[1]. In Kentucky University, Walden et al. examine connections between CT and critical thinking in college students, developing their own CT instrument: a combination of multiple choice and short answer questions that tests evaluation of simple algorithms, efficient sorting, quality of digital information storage, and file structures; this instrument is not verified yet valid or reliable [18]. In Taiwan, Lee et al. assess CT skill among high school and vocational school students [19], using fifteen tasks from the International Bebras Contest[2].

But there is still a lack of tests designed for middle-school students that has been subjected to a complete validation process. To begin to reply this vacuum, this paper presents the design of a *Computational Thinking Test (CT-test)* aimed at Spanish students between 12 and 13 years (grades K-7 & K-8): we describe the guidelines on which whole test and each of its items have been designed, as well as the content validation process through expert's judgment procedure.

[1] http://www.olympiad.org.za/talent-search/

[2] http://www.bebras.org/

2 METHODOLOGY

2.1 Design guidelines and principles

Our *Computational Thinking Test (CT-test)* was initially designed (version 1.0 – October 2014)[3] under the following principles:

- **Aim:** *CT-test* aims to measure the development level of computational thinking in the subject.
- **Operational definition of measured construct:** CT involves the ability to formulate and solve problems by relying on the fundamental concepts of computing, and using logic-syntax of programming languages: basic sequences, loops, iteration, conditionals, functions & variables.
- **Target population:** *CT-test* is designed and intended for Spanish students between 12 and 13 years old (grades K-7 & K-8).
- **Instrument Type:** multiple choice test with 4 answer options (only one correct).
- **Length:** 40 items.
- **Estimated completion time:** 45 minutes.

Each item of the *CT-test* is designed and characterized in the following five dimensions:

- **Computational concept addressed:** each item addresses one or more of the following 8 computational concepts, ordered in increasing difficulty: Basic directions (5 items); Loops – repeat times (5 items); Loops – repeat until (5 items); If – simple conditional (5 items); If/else – complex conditional (5 items); While conditional (5 items); Simple functions (5 items); Functions with parameters (5 items). These computational concepts are aligned with *'CSTA K12 Computer Science Standards'* for grades K-7 & K-8 [20].
- **Environment-Interface of the item:** *CT-test* items are presented in any of the following two environments-interfaces: *'The Maze'* (31 items); *'The Canvas'* (9 items).
- **Answer alternatives style:** in each item, the response alternatives may be presented in any of these three styles: Visual arrows (8 items); Visual blocks (24 items); Textual (8 items).
- **Existence or non-existence of nesting:** depending on whether the item solution involves a script with (29 items) or without (11 items) nesting computational concepts (a concept embedded in another concept to a higher hierarchy level).
- **Required task:** depending on which of the following cognitive tasks is required for resolution of the item: Sequencing: the student must sequence, stating in an orderly manner, a set of commands (18 items); Completion: the student must complete an incomplete given set of commands (14 items); Debugging: the student must debug an incorrect given set of commands (8 items).

See item examples in **Fig. 1**, **Fig.2**, **Fig. 3** and **Fig. 4**, detailing their specifications in the five dimensions.

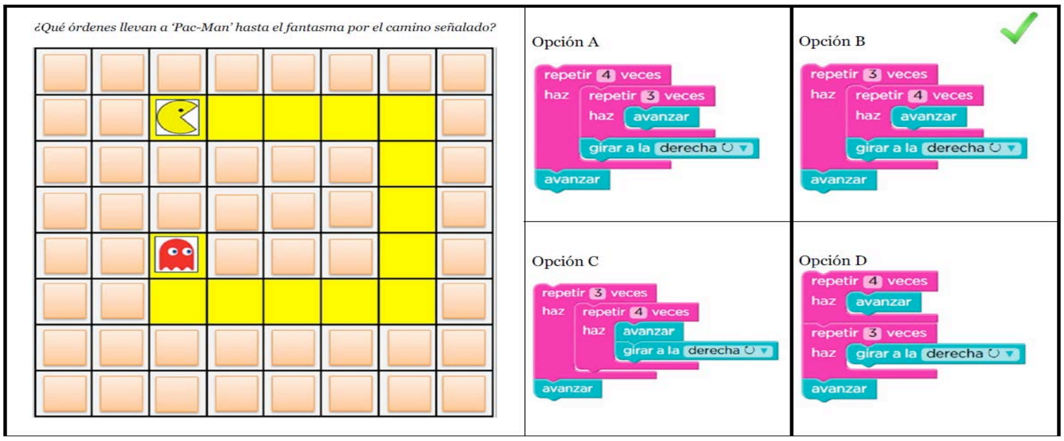
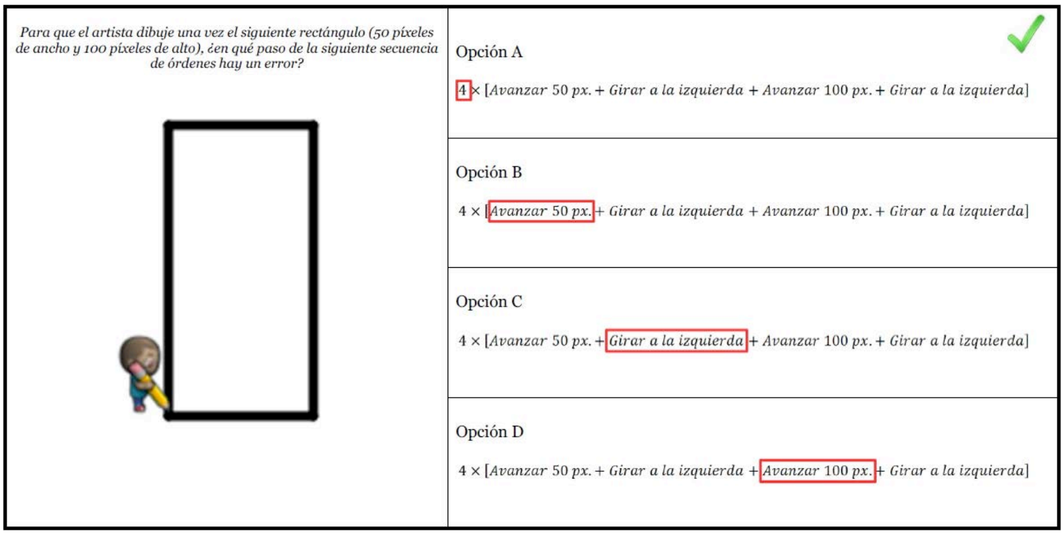


Fig. 1. Item details: Loop – repeat times; *'The Maze'*; Visual blocks; Yes-nesting; Sequencing.

[3] Available in high definition at: https://db.tt/66FJPURK



Para que el artista dibuje una vez el siguiente rectángulo (50 píxeles de ancho y 100 píxeles de alto), ¿en qué paso de la siguiente secuencia de órdenes hay un error?

Opción A ✓

$\boxed{4} \times [Avanzar\ 50\ px. + Girar\ a\ la\ izquierda + Avanzar\ 100\ px. + Girar\ a\ la\ izquierda]$

Opción B

$4 \times [\boxed{Avanzar\ 50\ px.} + Girar\ a\ la\ izquierda + Avanzar\ 100\ px. + Girar\ a\ la\ izquierda]$

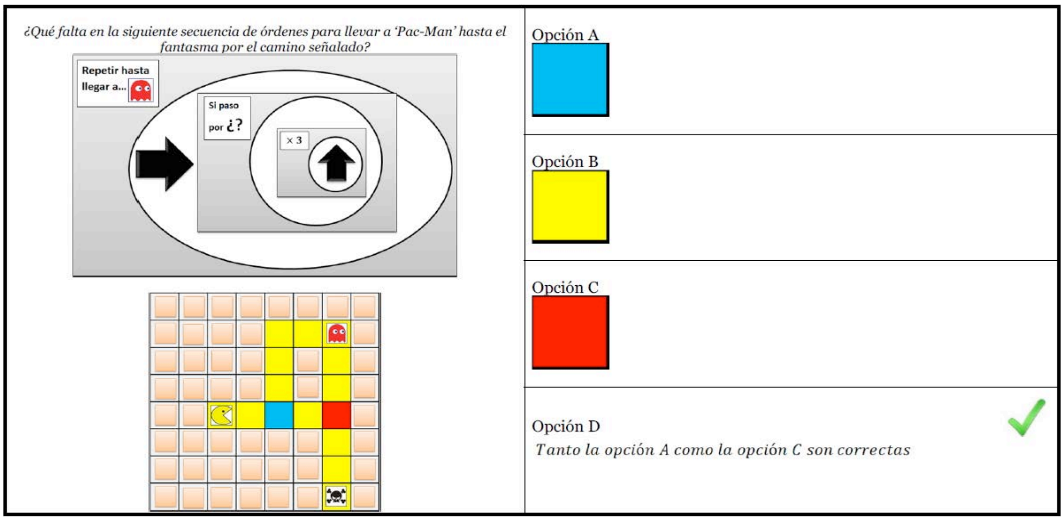
Opción C

$4 \times [Avanzar\ 50\ px. + \boxed{Girar\ a\ la\ izquierda} + Avanzar\ 100\ px. + Girar\ a\ la\ izquierda]$

Opción D

$4 \times [Avanzar\ 50\ px. + Girar\ a\ la\ izquierda + \boxed{Avanzar\ 100\ px.} + Girar\ a\ la\ izquierda]$

Fig. 2. Item details: Loop – repeat times; *'The Canvas'*; Textual; No-nesting; Debugging.



¿Qué falta en la siguiente secuencia de órdenes para llevar a 'Pac-Man' hasta el fantasma por el camino señalado?

Opción A 🟦

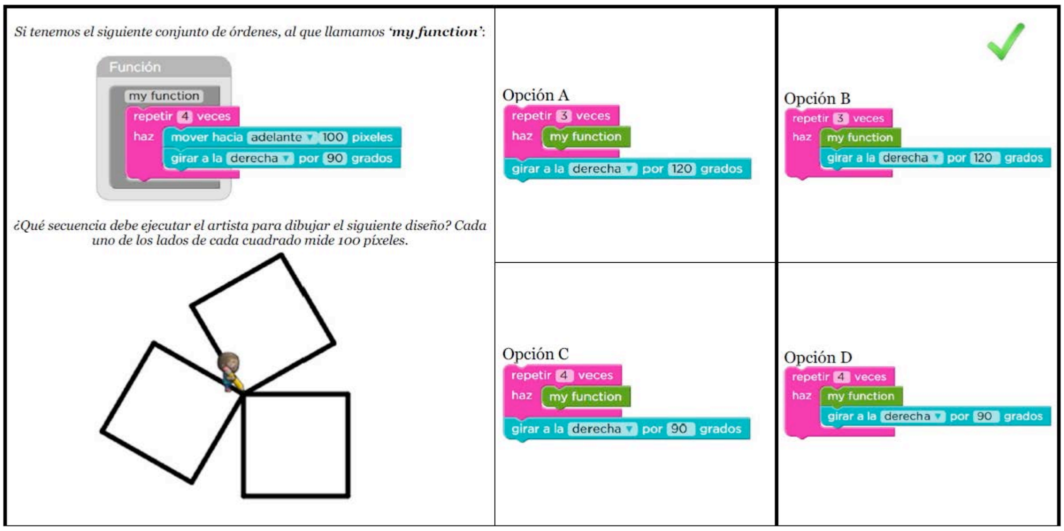
Opción B 🟨

Opción C 🟥

Opción D ✓

Tanto la opción A como la opción C son correctas

Fig. 3. Item details: Loop – repeat until + If – simple conditional; *'The Maze'*; Visual arrows; Yes-nesting; Completion.



*Si tenemos el siguiente conjunto de órdenes, al que llamamos **'my function'**:*

```
Función
  my function
    repetir 4 veces
    haz  mover hacia adelante 100 pixeles
         girar a la derecha por 90 grados
```

¿Qué secuencia debe ejecutar el artista para dibujar el siguiente diseño? Cada uno de los lados de cada cuadrado mide 100 píxeles.

Opción A

```
repetir 3 veces
haz  my function
girar a la derecha por 120 grados
```

Opción B ✓

```
repetir 3 veces
haz  my function
     girar a la derecha por 120 grados
```

Opción C

```
repetir 4 veces
haz  my function
girar a la derecha por 90 grados
```

Opción D

```
repetir 4 veces
haz  my function
     girar a la derecha por 90 grados
```

Fig. 4. Item details: Loop – repeat times + Simple functions; *'The Canvas'*; Visual blocks; Yes-nesting; Sequencing.

2.2 Expert's judgment procedure

For the content validation procedure of our *CT-test*, 39 experts were invited to collaborate in order to provide their valuation of the instrument. Finally, 20 experts accepted; this panel of experts has the following features: 14 men and 6 women; average age 36,9 years; belonging to the following professional groups (you can belong to more than one) as seen in **Table 1**.

Table 1. Professional profile of our panel of experts

Professional group	***Nº of experts***
K-7 to K-10 Computer Science Teachers (Middle School)	11
K-11 to K-12 Computer Science Teachers (High School)	8
Job Training Computer Science Teachers	5
University Computer Science Teachers	5
Winners *Premios Apps Fundación Telefónica*[4] (App Development Contest)	4
Members of *Programamos.es*[5]	3
Members of *Computer Science Teachers Association*[6] *(Valencia, Spain)*	3

Our *CT-test* (version 1.0 – October 2014) was sent to the 20 accepting experts, and following valuations were required:

- **For each of the 40-items:** difficulty level (Likert scale of ten points); relevance to measure CT (Likert scale of ten points); whether to include or not the item in the final version of the *CT-test*; open suggestions for item's improvement.
- **For each of the 5-dimensions:** adequacy to have considered the dimension for the *CT-test* design (Likert scale of ten points); whether to consider or not the dimension in the final *CT-test* design; open observations about the dimension.
- **For the whole instrument:** valuation on the length of the test (Likert scale of five points) & ideal length for final version; valuation on the estimated completion time of the test (Likert scale of five points) & ideal completion time relative to ideal length; overall valuation (Likert scale of ten points); and open final comments.

Valuations were collected through three online forms in Google Drive[7]

3 RESULTS

About the length of *CT-test* (v. 1.0), 40 items: 11,8% of the experts said is 'very long'; 58,8% said is 'quite long'; and 29,4% said is 'appropriate length'. About the estimated time of completion (45 minutes) in relation to the originally scheduled length scale (40 items): 70,6% of the experts indicated is 'very short' or 'quite short' time; 17,6% indicated 'appropriate time'; and 11,8% said 'quite long' time. Ideal length given by the experts for final version of *CT-test* was 28,3 items (Mean) or 30 items (Median); and ideal completion time relative to ideal length was 52,5 minutes (Mean) or 45 minutes (Median).

Difficulty level, in relation to the target population (12 & 13 yr.), perceived by experts to each item is shown in **Fig. 5**. Relevance to measure CT perceived by experts to each item is shown in **Fig. 6**. And whether to include or not each item in the final version of the *CT-test*, expressed as an acceptance rate, is shown in **Fig. 7**.

[4] http://www.fundaciontelefonica.com/

[5] http://programamos.es/

[6] http://www.apicv.es/

[7] http://goo.gl/6p1gcR http://goo.gl/JSrHSD http://goo.gl/xCZlU8

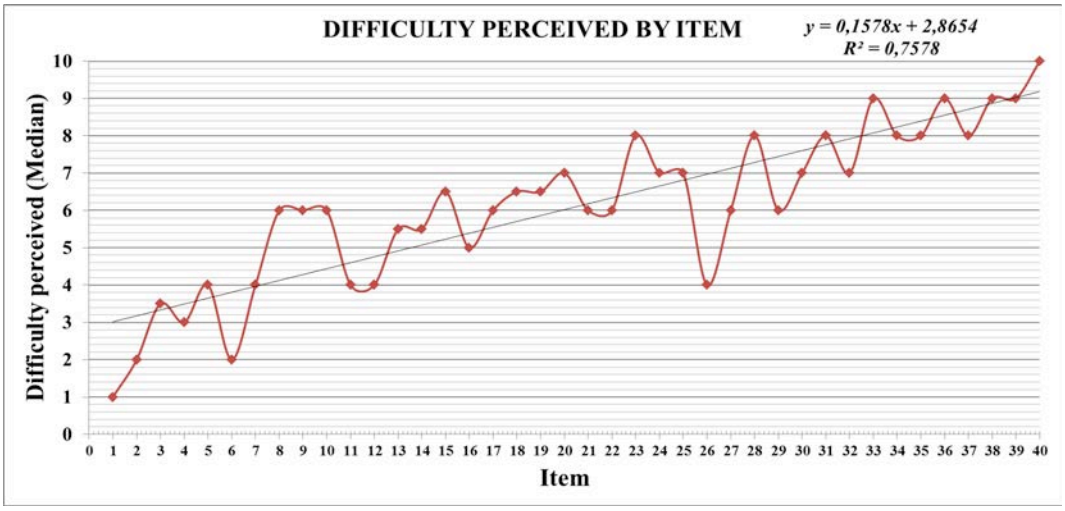


Fig. 5. Difficulty level perceived by experts to each item.

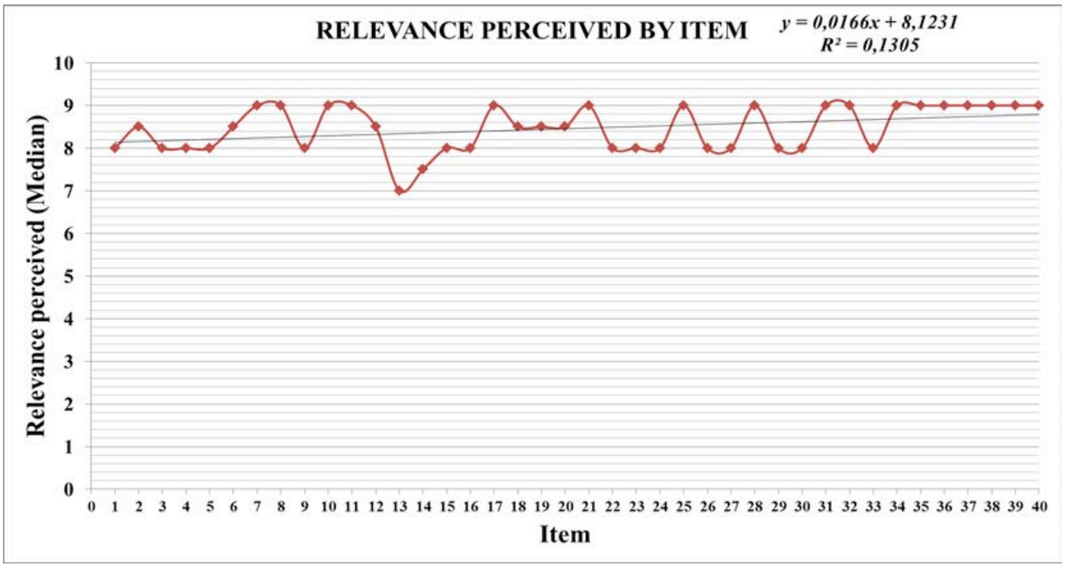


Fig. 6. Relevance to measure CT perceived by experts to each item.

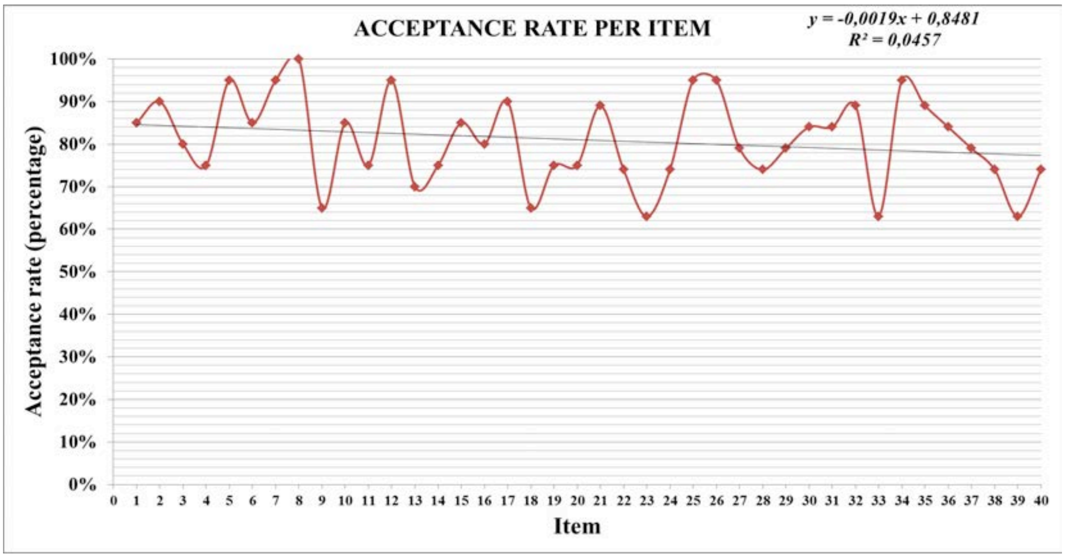


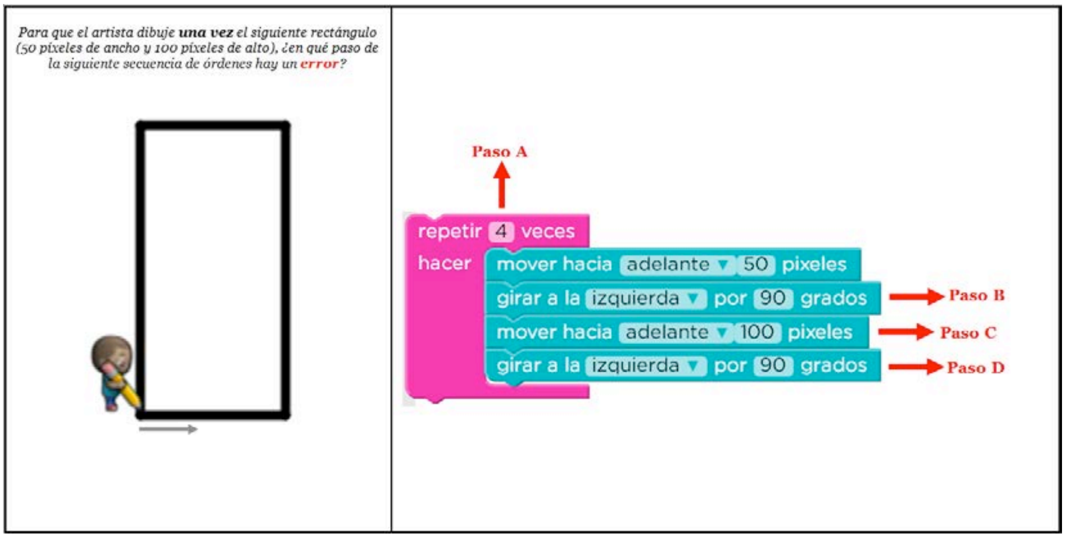
Fig. 7. Acceptance rate by experts to each item.

From the analysis of the above results, we can state: *CT-test* (v. 1.0) presents a growing difficulty along its items, something recommended for any instrument that aims to measure skills, covering the entire range of perceived difficulty (from 1 to 10); items have a high rate of acceptance by experts, at around 80-90%, although with a slight slope to the last group of items, perhaps because of its undue hardship for the target population; items have a high relevance for measuring CT, albeit with a slight increase towards the last group of items (where the most complex computational concept measured: 'Functions with parameters'), what advises to keep these items for a future version of the instrument aimed to an older population.

Moreover, considering the suggestions and comments of the experts to different items and construction dimensions of the instrument, the following decisions to refine the initial version of the *CT-test* are taken:

- Remove the latest group of 5 items corresponding to computational concept 'Functions with parameters' by its excessive complexity for the target population.
- Dispense from 'Textual' answer alternatives style; whereas visual styles (arrows or blocks) are most appropriate for the target population.
- Reformulate the items that require 'debugging' of an erroneous sequence of commands: show the code only once, and on the same note 4 options that the wrong step can be found.
- Include a hint about the initial direction of the stroke in *'The Artist'* interface.
- Introduce nesting only after entering corresponding computational concept without nesting; eliminate excessive nesting (double and triple nesting).
- As is suggested to reduce the length of the instrument and each computational concept is approached by a group of five items, we delete or revise the item with the lowest acceptance rate of each group.
- Include at the beginning of test, brief instructions and three examples that serve to familiarize students with the working environment.

Applying the decisions above, **Fig. 8** shows how an item from *CT-test* version 1.0. (See **Fig. 2**) was revised for *CT-test* version 2.0.


Para que el artista dibuje ***una vez*** *el siguiente rectángulo (50 píxeles de ancho y 100 píxeles de alto), ¿en qué paso de la siguiente secuencia de órdenes hay un* ***error****?*

```
repetir 4 veces
hacer   mover hacia adelante ▾ 50 pixeles
        girar a la izquierda ▾ por 90 grados
        mover hacia adelante ▾ 100 pixeles
        girar a la izquierda ▾ por 90 grados
```

Fig. 8. Revision of an item (see **Fig. 2.**) for *CT-test* version 2.0.

As a result of this depuration process was developed *CT-test* (version 2.0 – November 2014)[8] of 28 items in length, which is currently being implemented in large samples of the target population through an online form[9].

[8] Available in high definition at: https://db.tt/6hg2seLu

[9] http://goo.gl/IYEKMB

4 CONCLUSIONS

So in this paper we have submitted the content validation process through expert's judgment procedure of our *CT-test*. As a result, the initial version (*CT-test 1.0*) of 40 items in length has been refined into a final version (*CT-test 2.0*) of 28 items. Currently, the *CT-test* is being applied to large samples of the target population: students between 12 and 13 years old (grades K-7 & K-8) with the aim of building rules, scales and percentiles corresponding to that population. Additionally, we are applying the *CT-test* on smaller samples of the immediately lower grades (K-5 & K-6) and above (K-9 & K-10) with a view to properly position the floor and ceiling of the instrument; and to validate the hypothesis of CT's evolutionary development.

The *CT-test* is undergoing therefore a complete process to demonstrate its reliability and validity. As sources for concurrent validity of *CT-test*, we are using two types of measures: a) measures already validated from variables close to CT, such as logical reasoning, problem solving, and perceptual and attentional skills; b) CT alternative measures such as a selection of items from the Computer Olympiad 'Talent Search' or the International Bebras Contest, already mentioned in the introduction. As sources of predictive validity, we intend to correlate scores on the *CT-test* with the future performance of students in *'K-8 Intro to Computer Science'*[10] course from Code.org, and with the quality of their future products made with Scratch (objectively measured by *'Dr. Scratch'*[11] tool).

Some of the possible purposes to which *CT-test* can be assigned after completing its validation process, could be: pre-test measure of initial CT level in students; early detection of students with special skills for programming tasks, or with special needs in this regard; evaluation of curricula that aim teaching and learning computer science; academic and professional orientation of students toward STEM disciplines.

Finally, by applying the *Computational Thinking Framework (CTF)* [9] to our instrument, we note as obvious limitation thereof, as *CT-test* is composed entirely of closed multiple choice items, it only measures CT at their lower levels of complexity ('Recognize' and 'Understand'). An instrument intended to measure CT also at higher levels of complexity ('Apply' and 'Assimilate') should include items which require not only recognize but also evoke the correct algorithm (as, indeed, suggested one of our experts), and open complex problems whose resolution require students transfer creatively computational concepts. In the same vein, we wonder whether our test is really a *CT-test* or 'just' an *AT-test (Algorithm Thinking Test)*.

REFERENCES

[1] Manovich, L. (2013). *Software Takes Command*. New York: Bloomsbury.

[2] Rushkoff, D. (2010). *Program or be programmed*. New York: OR Books.

[3] Román, M. (2014). Aprender a programar 'apps' como enriquecimiento curricular en alumnado de alta capacidad. *Bordón. Revista de Pedagogía, 66*(4), 135-155.

[4] Henderson, P. B., Cortina, T. J., & Wing, J. M. (2007). Computational thinking. *ACM SIGCSE Bulletin, 39*(1), 195-196.

[5] Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York; Basic Books.

[6] Howland, K., & Good, J. (2015). Learning to communicate computationally with flip: A bi-modal programming language for game creation. *Computers & Education, 80*, 224-240. doi: http://dx.doi.org/10.1016/j.compedu.2014.08.014

[7] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., et al. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60-67.

[8] Brown, N. C. C., Kölling, M., Crick, T., Peyton Jones, S., Humphreys, S., & Sentance, S. (2013). Bringing computer science back into schools: Lessons from the UK. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education,* 269-274.

[10] http://studio.code.org/s/20-hour

[11] Beta version: http://drscratch.programamos.es/

[9] Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013). Computational thinking in educational activities: An evaluation of the educational game light-bot. *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education,* 10-15.

[10] Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.

[11] Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions. Series A, Mathematical, Physical, and Engineering Sciences, 366*(1881), 3717-3725. doi: http://dx.doi.org/10.1098/rsta.2008.0118

[12] Aho, A. V. (2012). Computation and computational thinking. *The Computer Journal, 55*(7), 832-835.

[13] CSTA & ISTE (2011). *Operational Definition of Computational Thinking for K–12 Education*. Retrieved from http://csta.acm.org/Curriculum/sub/CurrFiles/CompThinkingFlyer.pdf

[14] Google for Education (2015). *Exploring Computational Thinking*. Retrieved from https://www.google.com/edu/resources/programs/exploring-computational-thinking/

[15] Grover, S., & Pea, R. (2013). Computational thinking in K–12. A review of the state of the field. *Educational Researcher, 42*(1), 38-43.

[16] Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The fairy performance assessment: Measuring computational thinking in middle school. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education,* 215-220.

[17] Gouws, L., Bradshaw, K., & Wentworth, P. (2013). First year student performance in a test for computational thinking. *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference,* 271-277.

[18] Walden, J., Doyle, M., Garns, R., & Hart, Z. (2013). An informatics perspective on computational thinking. *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education,* 4-9.

[19] Lee, G., Lin, Y., & Lin, J. (2014). Assessment of computational thinking skill among high school and vocational school students in Taiwan. *World Conference on Educational Multimedia, Hypermedia and Telecommunications, 2014*(1), 173-180.

[20] CSTA (2011). *K–12 Computer Science Standards (Level 2)*. Retrieved from http://csta.acm.org/Curriculum/sub/CurrFiles/CSTA_K-12_CSS.pdf