Revista Latinoamericana de Tecnología Educativa, 17(2)
http://dx.medra.org/10.17398/1695-288X.17.2.129

RELATEC
Revista Latinoamericana de Tecnología Educativa

ARTÍCULO / ARTICLE

Robótica DIY: pensamiento computacional para mejorar la resolución de problemas

DIY robotics: computational thinking based patterns to improve problem solving

Beatriz Ortega-Ruipérez[1] y Mikel Mirena Asensio Brouard[2]

Recibido: 27 Septiembre 2018
Revisado: 3 Diciembre 2018
Aceptado: 10 Diciembre 2018

Dirección autores:

[1] Universidad Internacional de La Rioja (UNIR). Calle de Almansa, 101, 28040 - Madrid (España)

[2] Departamento de Psicología Básica. Facultad de Psicología. Universidad Autónoma de Madrid. Campus de Cantoblanco. C/ Iv¡án P. Pavlov, 6. 28049 – Madrid (España)

E-mail / ORCID

beatriz.ortega.ruiperez@unir.net

http://orcid.org/0000-0002-3822-5745

mikel.asensio@uam.es

http://orcid.org/0000-0002-1020-5486

Resumen: La programación está incluyéndose en los currículos educativos de todo el mundo para desarrollar el pensamiento computacional. Sin embargo, no hay un consenso sobre qué procesos implica este pensamiento, ni sobre cómo intervenir y evaluar su desarrollo. Por tanto, el objetivo es proponer una estrategia de enseñanza para la programación y robótica, que realmente desarrolle este pensamiento y se pueda aplicar para resolver problemas, desde una perspectiva maker que facilite la transferencia de conocimientos a contextos reales. Para ello, se ha impartido un curso de robótica insistiendo en los procesos cognitivos de este pensamiento que se emplean habitualmente en la resolución de problemas (abstracción, tratamiento de datos, creación de un algoritmo), y animando a utilizar una estrategia computacional, utilizando los procesos de este pensamiento no empleados en la resolución de problemas (descomposición del problema, automatización, paralelismo, simulación). Para medirlo se han creado unas pruebas digitales basadas en el enfoque de sistemas-complejos-múltiples, utilizado en PISA 2012. Los resultados indican que el pensamiento computacional se aplica más fácilmente a la ejecución del algoritmo que a la representación del problema. Este hallazgo nos permite establecer un proceso de aprendizaje de la programación que facilite el desarrollo del pensamiento computacional, para resolver cualquier problema aplicando una estrategia computacional: centrándose primero en aplicar dicha estrategia a la creación del algoritmo y después a la representación del problema.

Palabras clave: Pensamiento computacional, Robótica, Resolución de problemas, Tecnología educativa, Lenguajes de programación.

Abstract: Programming is being included in educational curricula around the world to develop computational thinking. However, there is no consensus on what processes this thought implies, nor on how to intervene and evaluate its development. Therefore, the objective is to propose a teaching strategy for programming and robotics, which really develops this thinking and can be applied to solve problems, from a maker perspective that facilitates the transfer of knowledge to real contexts. To this end, a robotics course has been taught, insisting on the cognitive processes of this thinking that are commonly used in problem solving (abstraction, data processing, creation of an algorithm), and encouraging the use of a computational strategy, using the processes of this thought not employed in problem solving (decomposition of the problem, automation, parallelism, simulation). To measure it, digital tests have been created based on the multiple complex-systems approach, used in PISA 2012. The results indicate that computational thinking is applied more easily to the execution of the algorithm than to the representation of the problem. This finding allows us to establish a programming learning process that facilitates the development of computational thinking, to solve any problem by applying a computational strategy: focusing first on applying this strategy to the creation of the algorithm and then to the representation of the problem.

Keywords: Computational Thinking, Problem solving, Robotic, Educational technology, Programming Languages.

1. Introducción

El pensamiento computacional surge como un concepto transversal que engloba un conjunto de habilidades y procesos cognitivos. Estos procesos y habilidades son indispensables para los expertos en Ciencias de la Computación, y para otras disciplinas científicas (Wing, 2006). Desde la propuesta inicial de Wing, los procesos incluidos han aumentado (Wing, 2008, 2011, 2014, 2017). La persona que emplea un pensamiento computacional puede descomponer el problema en pequeños problemas que sean más fáciles de resolver, y reformular cada uno de estos problemas para facilitar su solución por medio de estrategias de resolución de problemas familiares. Por tanto, según Wing y otros autores, usando conceptos computacionales, es decir, haciendo cómputos, podemos mejorar la forma en la que nos aproximamos a los problemas, gestionamos nuestra vida diaria y nos comunicamos e interactuamos con otras personas (Stahl, Koschmann y Suthers, 2014; Wing, 2014). Este pensamiento podría ser un factor fundamental para resolver un problema utilizando un método riguroso y científico, por lo que esforzarnos en buscar formas de desarrollarlo en todas las personas podría llevarnos a conseguir una sociedad tecnológica exitosa y a facilitar el empoderamiento de las personas (National Research Council, 2010).

Herde, Wüstenberg y Greiff (2016) señalan que todavía no se conocen las estrategias que mejor pueden abordar la resolución de problemas complejos y afirman que, si estas estrategias se conociesen, deberíamos destinar nuestros esfuerzos a enseñar dichas estrategias. Si el pensamiento computacional se puede emplear como una estrategia útil para resolver problemas complejos, deberíamos investigarlo y proponer una forma de desarrollarlo mediante la programación. Además, para facilitar la resolución de problemas complejos en situaciones ajenas al contexto de programación empleando el pensamiento computacional, necesitamos alcanzar un dominio suficiente de los procesos cognitivos que utiliza este pensamiento, o de lo contrario este pensamiento se va a limitar a facilitar la resolución de problemas relacionados con la computación.

En los últimos años se están viendo diversos intentos de incluir el pensamiento computacional en los currículos educativos oficiales a través de la programación (Valverde-Berrocoso, Fernández-Sánchez y Garrido-Arroyo, 2015). Entre los intentos más estudiados encontramos por ejemplo el caso del Reino Unido y, en España, la asignatura Tecnología, Programación y Robótica, de la Comunidad de Madrid en el 2015/16. Sin embargo, casi todos estos intentos dan por supuesto que, enseñando a programar, se desarrollará este pensamiento, y eso es peligrosamente incierto. Para que la programación sea un buen recurso para desarrollar el pensamiento computacional, necesitamos marcar ciertas directrices, ya que, como afirma Zapata-Ros (2015), en la actualidad la tendencia más frecuente es enseñar a programar de una forma poco reflexiva, enseñando a crear código desde las tareas más sencillas y lúdicas a las más complejas y aburridas.

Esta forma de enseñar programación tiene como objetivo formar a personas capaces de utilizar la programación en su futuro profesional, en lugar de tener como objetivo el desarrollar el pensamiento computacional. Esto conlleva a que, en muchas propuestas educativas, se aprenda a programar sin pensar sobre el problema a resolver, sin hacer un diseño previo a la creación del programa que permita reflexionar sobre cómo resolverlo; por ejemplo, en la línea del modelo optimal de Chiu y Churchill (2015).

Como afirma Zapata-Ros (2015), fomentando actividades de cortar y pegar fragmentos de código para realizar operaciones concretas, sin entender la lógica que conlleva crear ese código.

Por estos motivos, se hace necesario investigar estrategias adecuadas para desarrollar este pensamiento aprendiendo programación, de forma que podamos aplicarlo a resolver problemas en contextos diferentes. Como señalan Ritchhart y Perkins (2005), la enseñanza del pensamiento, y no sólo la del pensamiento computacional, necesita todavía abordar la gran cuestión sobre cómo integrar esta enseñanza en otras prácticas, en el colegio y fuera del colegio, en un camino efectivo.

1.1. Estructura del pensamiento computacional y su relación con la resolución de problemas

El pensamiento computacional se compone de una serie de procesos cognitivos que todavía no tienen un claro y consistente apoyo empírico. Selby y Woollard (2013) establecen una definición de este concepto a partir de la revisión crítica de todas las publicaciones relevantes existentes hasta la fecha de su publicación, incluyendo y excluyendo las características y/o procesos cognitivos, con mayor y menor aparición en la literatura, respectivamente. Esta definición comparte prácticamente los mismos procesos cognitivos que proponen el Computer Science Teacher Association y el International Society Technology Education (CSTA y ISTE, 2011), quienes recogen de forma operativa las características que según ellos componen el pensamiento computacional. Se definen de una forma más operativa de cara a facilitar su estudio, por lo que han sido la base para elaborar la propuesta didáctica. Estos procesos son abstracción, tratamiento de datos (recopilación de datos, análisis de datos y representación de datos), creación de un algoritmo, descomposición de un problema, automatización, simulación y paralelismo.

En paralelo se han publicado otras propuestas sobre los procesos del pensamiento computacional, de cara a conseguir una definición operativa que permita medir el desarrollo de este pensamiento. Sin embargo, estas propuestas están demasiado ligadas a la programación, y al aprendizaje de la misma, por lo que algunos de los procesos que se proponen no son en sí procesos cognitivos relativos a un pensamiento, sino que son prácticas que se utilizan habitualmente en la programación, como por ejemplo la depuración de código (Brennan y Resnick, 2012), la absorción o colisión para medir el significado semántico de los juegos programados por los estudiantes (Koh, Nickerson, Basawapatna y Repenning, 2014) o la sincronización (Moreno-León, Robles y Román-González, 2015). Como vemos, estos aspectos se incluyen en algunas propuestas como componentes del pensamiento computacional, pero se pueden clasificar mejor como prácticas de programación que como procesos cognitivos que se emplean en contextos diferentes a la programación.

El marco propuesto por Brennan y Resnick (2012), quienes diferencian tres niveles de desarrollo del pensamiento computacional: conceptos, prácticas y perspectivas, es uno de los que más se están utilizando para valorar el desarrollo de este pensamiento. Sin embargo, Lye y Koh (2014) analizan muchos de los instrumentos de evaluación creados a partir de este marco y descubren que sólo 6 de las 27 propuestas revisadas evalúan las prácticas del pensamiento computacional, y únicamente 2 de las 27 incluyen la evaluación de las perspectivas del pensamiento computacional, lo que indica que realmente están evaluando el nivel más básico de desarrollo de este pensamiento, que correspondería a conocer los conceptos, pero no

alcanza el nivel de ponerlos en práctica. Una crítica hacia estos modelos es que evalúan el desarrollo del pensamiento computacional a través de prácticas de programación muy sencillas, especialmente en los mismos lenguajes de programación en los que han aprendido a programar, como ocurre con el software Scratch (Muñoz, Barcelos, Villarroel y Silveira, 2017; Robles, Moreno-León, Aivaloglou y Hermans, 2017; Sáez-López y Cózar-Gutiérrez, 2017).

Esta característica impide evaluar la transferencia del conocimiento, la aplicación de este pensamiento a contextos diferentes de la programación que han aprendido, ya que se evalúa en el mismo contexto de aprendizaje. Además, se emplean pequeñas tareas o problemas en las que se conocen todos los datos desde el inicio del problema, lo que no supone un problema complejo para el alumnado y, por tanto, descartando evaluar cómo el pensamiento computacional se aplica a la resolución de problemas complejos. Más de 10 años después del surgimiento del concepto, la definición del pensamiento computacional y, por tanto, la forma de intervenir para desarrollarlo y evaluar dicho pensamiento requiere una mayor investigación, tal y como afirman Shute, Sun y Asbell-Clarke (2017), antes de dar por supuesto que cualquier forma de hacerlo es válida por basarse en un marco teórico.

Esta investigación debe ser interdisciplinar, no sólo debe abordarse desde las ciencias de la computación y las ciencias de la educación, sino que debe incluir a la psicología, disciplina encargada de estudiar el pensamiento, y los procesos cognitivos que subyacen a cada tipo de pensamiento, de forma que logremos una buena definición. Si el pensamiento computacional se emplea en la resolución de problemas, debemos basar la intervención y evaluación en comprobar cómo se resuelven problemas en situaciones diferentes a la programación, y no en cómo se crean programaciones.

Por su parte, la resolución de problemas ha sido ampliamente estudiada, y para conocerla mejor se descompone tradicionalmente en dos fases: representación y proceso de ejecución (Holyoak y Morrison, 2012). Si nos centramos en la resolución de problemas complejos, cuando supuestamente es especialmente útil el pensamiento computacional, se mantienen estas dos fases. Greiff, Wüstenberg y Funke (2012) enriquecen este marco para estudiar la resolución de problemas complejos en las pruebas internacionales PISA 2012, en las que estudiantes de 15 años de todo el mundo miden su conocimiento en diferentes áreas para poder establecer comparaciones globales y actuar más rápido para mejorar los sistemas educativos (OECD, 2010).

En el enfoque propuesto por estos autores, un enfoque de sistemas-complejos-múltiples, se proponen varias pruebas digitales de corta duración de temáticas diversas, de forma que la fiabilidad de la evaluación aumenta al no interferir el conocimiento de un tema concreto con el resultado. Además, la presentación digital de las pruebas permite la interacción del sujeto con la información del problema, información que en los problemas complejos se encuentra de forma dinámica, de ahí que requiera interacción del sujeto para acceder a toda la información (para una revisión del concepto de interactividad en tecnología ver Asenjo y Asensio, en prensa)

El enfoque de sistemas múltiples complejos evalúa cuatro fases de la resolución de problemas complejos, que a su vez se enmarcan en las dos fases estudiadas tradicionalmente: representación y ejecución. Dentro de la fase de representación se encuentra la fase de exploración y entendimiento y la fase de representación y

formulación. Por su parte, la fase de ejecución incluye una fase de planificación y ejecución, y una segunda fase de monitorización y reflexión (Greiff, Wüstenberg y Funke, 2012).

En cada una de estas fases se emplean ciertos procesos cognitivos que se han definido como parte del pensamiento computacional y que son fácilmente identificables en cada una de estas fases. Estos procesos son: abstracción, tratamiento de datos (recopilación de datos, análisis de datos y representación de datos) y creación de un algoritmo. En la fase de exploración y entendimiento empleamos los procesos relacionados con el tratamiento de datos: recopilación, análisis y representación de datos. En la fase de representación y formulación necesitamos emplear la abstracción junto al tratamiento de datos. En la fase de planificación y ejecución utilizamos los procesos relacionados con la creación de algoritmos, con un pensamiento secuencial o algorítmico. En la fase de monitorización y reflexión, volvemos a utilizar los procesos relacionados con el tratamiento de los datos para analizar si la ejecución del plan está siguiendo el camino correcto.

Sin embargo, otros de los procesos definidos operativamente en el pensamiento computacional no se emplean habitualmente en la resolución de problemas. Estos procesos son: descomposición de un problema, automatización, simulación y paralelismo. El que estos procesos no se empleen habitualmente en la resolución de problemas, nos indica que son propios del pensamiento computacional, son estrategias propias que utilizan los científicos de la computación. Entre estos procesos destaca la descomposición del problema, que sería la clave de la estrategia computacional, necesaria para poder afrontar el problema con éxito. Esta descomposición del gran problema en pequeños problemas o submetas, permite que algunos de estos pequeños problemas se puedan abordar de forma independiente: algunos se podrían automatizar, otros se podrían resolver o avanzar en ellos de forma paralela y simultánea (Ortega-Ruipérez, 2018).

El estudio realizado por Park, Song y Kim (2015) con la técnica de electroencefalograma muestra cómo la carga cognitiva de los estudiantes que emplean un pensamiento computacional es menor que la carga cognitiva de los estudiantes que no emplean este pensamiento, lo que demuestra cómo el pensamiento computacional puede ayudarnos a abordar grandes problemas complejos, al hacer un mejor manejo de nuestra carga cognitiva. La teoría de carga cognitiva (Sweller, 1994) está actualmente en revisión integrando cada vez factores más amplios del funcionamiento mental y afectivo (Mayer y Estrella, 2014).

Partiendo de este razonamiento, el pensamiento computacional facilita la resolución de problemas gracias al aprendizaje de técnicas para la automatización, el paralelismo y la simulación, que son posibles gracias a la descomposición del problema, al abordarlo como pequeños cómputos (de ahí, el calificativo de computacional). Con este supuesto, se ha creado un modelo de análisis que nos va a servir para el estudio del pensamiento computacional y se puede ver en la figura 1.

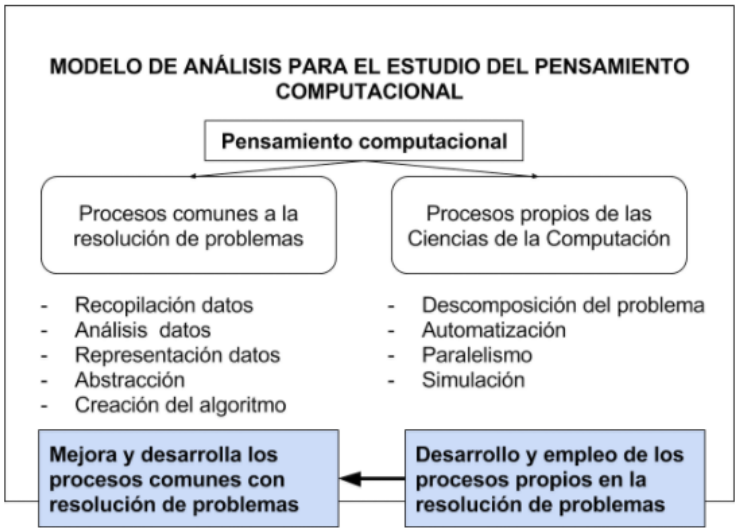

Figura 1. Modelo de análisis para el estudio del pensamiento computacional. Fuente: Elaboración propia.

Por tanto, estos procesos son propios del pensamiento computacional, mientras que los procesos de abstracción, tratamiento de datos y creación de un algoritmo son procesos habituales en la resolución de problemas, que pueden ayudarse de una estrategia computacional para abarcar problemas más complejos, al ser capaz de solucionarlos a través de la descomposición del problema, y al utilizar técnicas como la automatización, el paralelismo y la simulación (Ortega-Ruipérez, 2018).

1.2. Transferencia del pensamiento computacional a contextos reales: cultura maker y filosofía DIY

Las prácticas de aprendizaje basadas en problemas tienen una larga tradición de aplicación en educación y han sido objeto de una considerable cantidad de investigaciones (Hmelo-Silver, 2012). Sin embargo, sobre la base de una afirmación genérica sobre los resultados positivos, los procedimientos de ejecución y los modelos de explicación, están aún lejos de materializarse en unas propuestas generalizadas y ampliamente compartidas (Lu, Bridges y Hmelo-Silver, 2014). No obstante, para facilitar que el pensamiento computacional se aplique a contextos no educativos, y a cualquier problema de la vida diaria que no implique programación, se requiere un nuevo paradigma educativo en el que el alumnado sea capaz de aplicar los conocimientos aprendidos en estos contextos diferentes al contexto de aprendizaje. En este sentido, la cultura maker es un concepto que surge como respuesta a la ética DIY (Do It Yourself o hazlo tú mismo). Esta cultura promueve un aprendizaje activo, a través de hacer y construir cosas, y se basa en un aprendizaje informal, motivado por la auto-realización y fomentando aplicaciones nuevas de la tecnología. La cultura maker puede facilitar la aplicación del pensamiento computacional en estos contextos, ya que su forma de aprendizaje proporciona un ambiente idóneo para transferir el conocimiento.

La tecnología DIY se sustenta en la idea de que cualquier persona puede desarrollar tareas que antes sólo podían desempeñar especialistas. El empoderamiento que da esta tecnología a las personas, al sentirse capaces de crear herramientas

tecnológicas que les ayuden en sus tareas y conocer sus posibilidades para mejorar nuestras vidas, puede aumentar el interés de todo el alumnado hacia la tecnología, acercando por tanto la tecnología a toda la sociedad y no sólo a unos pocos interesados en este campo de estudio. Por ello, se propone una formación con un enfoque maker, que aumente el interés de por la tecnología desde una perspectiva de ser capaz de crear, y que ayude a desarrollar el pensamiento computacional hasta un dominio que permita utilizar este pensamiento como una estrategia ventajosa para la resolución de problemas complejos en cualquier ámbito. Este enfoque de formación con el objetivo de desarrollar el pensamiento computacional ya se está viendo en muchos lugares, como México (Jiménez-Rasgado, 2018) y Estados Unidos (Kafai y Burke, 2017).

El objetivo de esta investigación es proponer una estrategia de enseñanza para la programación y robótica, que realmente desarrolle este pensamiento y se pueda aplicar para resolver problemas, desde una perspectiva maker que facilite la transferencia de conocimientos a contextos reales. Para ello, se plantean como objetivos específicos, por una parte, crear una intervención a partir de la comparativa del pensamiento computacional y la resolución de problemas, que nos sirve para estructurarla; y por otra parte evaluar la aplicabilidad del pensamiento desarrollado a problemas de cualquier ámbito, como los recogidos en PISA 2012 (OECD, 2010).

2. Método

Esta investigación adopta una metodología cuantitativa, empleando un estudio pre-post que compara los resultados entre un grupo experimental, que participa en una formación de 30 horas desarrollada en un contexto de extraescolar bajo una filosofía DIY, y un grupo control que no participa en ninguna formación alternativa. De esta forma, se podrá conocer mejor la relación entre el uso de las estrategias utilizadas por los científicos de la computación y la resolución de problemas complejos.

La actividad extraescolar comenzó en el segundo trimestre, y el curso se ha desarrollado entre el 20 de febrero y el 24 de abril, lo que ha supuesto 2 meses de intervención. Las sesiones se han desarrollado una vez a la semana y han tenido una duración de dos horas. Antes del comienzo del curso, los estudiantes llevaban 10 horas de extraescolar, pero en esas sesiones habían estado aprendido a crear algoritmos sencillos, sin conocer las estrategias que utilizan los científicos de la computación para organizar el código y afrontar problemas.

Los participantes son 38 alumnos y alumnas de 4º de Educación Secundaria de la Comunidad Autónoma de Madrid, España, obteniendo el correspondiente consentimiento informado. Del total de los participantes, 21 de ellos han asistido a la extraescolar de programación, donde se incluyó el curso de pensamiento computacional. Por tanto, 21 de los 38 participantes pertenecen al grupo experimental, el 55,3% de la muestra total. El otro 44,7% de la muestra, lo que equivale a 17 alumnos y alumnas, no han participado en dicha extraescolar, por lo que estos alumnos pertenecerán a nuestro grupo de control.

Para esta investigación se ha desarrollado un curso para desarrollar el pensamiento computacional como material para aplicar en la extraescolar de robótica. Este curso permite diferenciar los grupos de la variable independiente. Para el diseño del curso, se han tenido en cuenta dos aspectos. Por una parte, se deben enseñar las

estrategias que llevan a cabo los científicos de la computación cuando se encuentran ante un problema de su campo. Estas estrategias, aplicadas al aprendizaje de la programación, son:

1) Automatización. Para ello, tendrán que incluir instrucciones de control en su código, como condicionales y bucles, en lugar de algoritmos simples.
2) Descomposición de los problemas. En este punto se insiste en que no deben abordar el problema como un todo, si no que deben intentar identificar las diferentes partes del problema para abordarlas de forma independiente.
3) Simulación. Para desarrollar habilidades en simulación, tienen que ser capaces de proponer una solución y simular los pasos que harían hasta alcanzar la solución, de forma que puedan anticipar cuándo una solución no va a ser válida.
4) Paralelismo. Debido a las características de los problemas complejos se hace necesario trabajar en varias capas de información. A partir de este supuesto, el paralelismo se debe enfocar hacia la simulación y la descomposición de los problemas. Esto quiere decir que el alumnado deberá ser capaz de simular las soluciones a diferentes partes del problema a la vez y, en especial, deberá ser capaz de engranar todos esos procesos en su simulación para acercarse lo más posible a la solución del problema.

Por otra parte, se deben tener en cuenta dónde se aplican estas habilidades: en la resolución de problemas. Para trabajar este pensamiento, se insiste en sus procesos a través de unos pasos clave, que se harán explícitos cada vez que se les presente un reto o piensen en un proyecto. Los pasos clave que se han utilizado para el curso se han cogido de la propuesta que hacen Marais y Bradshaw (2015):

1) Lograr un claro estado de ánimo. Este paso se refiere a que no entren en pánico cuando vean un problema. Que piensen que van a poder resolverlo, que toda la información que necesitan va a estar disponible o accesible fácilmente, y que tienen que creer en ellos mismos, en que conocen la estrategia para abordarlo.
2) Tratar de entender el problema a fondo. En este paso se les hace ver la importancia de leer y releer el problema tantas veces como haga falta, hasta que estén seguros de que saben cuál es el objetivo del problema y qué tienen que hacer. Se debe preguntarles qué han leído, qué se les está pidiendo, para comprobar que su representación del problema es correcta.
3) Examinar críticamente el problema. Cuando tienen claro lo que les pide el problema, tienen que explorar el entorno (el enunciado del problema, los recursos físicos disponibles, en el caso de pruebas digitales todo lo que era interactivo, etc.) para comprobar si tienen todos los datos. Con estos datos, tienen que idear una solución, construir un roadmap (mapa del problema) para intentar solucionar el problema. Este paso termina con un modelo mental que les permite saber lo que tienen que hacer.
4) Simplificar el problema en base a una solución prevista. En este paso se insiste en dos de las estrategias utilizadas por los científicos de la computación: descomposición del problema y automatización.

5) Identificar una posible ruta hacia la solución. Este paso de la resolución del problema depende en gran medida de la creatividad de cada persona, que se puede trabajar con técnicas específicas de creatividad.

6) Asegurarse de que el problema puede resolverse. Por último, se insiste en las otras dos estrategias utilizadas por científicos de la computación: simulación y paralelismo.

Además, se han desarrollado unas pruebas de resolución de problemas como medida de la variable dependiente, basadas en el enfoque de sistemas-complejos-múltiples propuesto por Greiff, Wüstenberg y Funke (2012). Este enfoque se utilizó en el programa internacional de evaluación a estudiantes PISA 2012 (OECD, 2010). Las pruebas de resolución de problemas nos permitirán concluir si existen diferencias en la variable dependiente, entre los grupos creados en función de la variable independiente. El pretest se aplicó en una hora el mismo día que empezaba el curso, el 20 de febrero, en horario lectivo, a ambos grupos. El postest se realizó a ambos grupos un día después de la última sesión, el 25 de abril, e igualmente se aplicó en una hora.

Se han utilizado un total de seis pruebas basadas de dicho enfoque, que aúna los marcos MicroDYN y MicroFIN. El primer marco es relativo a un enfoque cuantitativo de relación con el problema, es decir, las variables son cuantitativas; el segundo marco tiene un enfoque cualitativo, por lo que la relación con sus variables se basa en estados concretos. Dos de las pruebas incluidas, se han extraído del programa de evaluación PISA para conseguir mayor fiabilidad. Estas pruebas son Climatizador y Billetes, la primera relativa al marco cuantitativo, al que pertenece MicroDYN; y la segunda relativa al marco cualitativo, al que pertenece MicroFIN. Las cuatro pruebas restantes se llaman Vídeo, Tren, Agenda y Coche; las tres primeras con enfoque cualitativo y la cuarta con enfoque cuantitativo.

Estas pruebas, con varias preguntas cada una de diferentes fases, se han elaborado para la creación de una batería de pruebas que abarquen todas las fases de resolución de problema planteadas en este marco, en los diferentes niveles de dificultad. La batería de pruebas resultante, de 15 preguntas, se han sometido a pruebas psicométricas en un estudio piloto (Ortega-Ruipérez, 2018). Tras los análisis de fiabilidad pertinentes, de cara a aumentar la consistencia interna de las pruebas, gracias al estadístico alfa de Cronbach, se han eliminado 3 de las preguntas. De esta forma, se ha obtenido un alfa de Cronbach de 0,7, lo que indica una fiabilidad suficiente. Así, finalmente, la prueba consta de 12 preguntas que se distribuyen en las diferentes fases que se miden de resolución de problemas (ver tabla 1).

Para analizar los datos de estas pruebas, se ha partido de diferentes hipótesis que pretenden conocer el proceso de desarrollo del pensamiento computacional, es decir, se ha querido profundizar en el conocimiento sobre qué fases del proceso de resolución de problemas son más fáciles de afrontar con el empleo de una estrategia computacional. Las hipótesis de partida son: (1) el pensamiento computacional mejora la resolución de problemas complejos, (2) el pensamiento computacional mejora la representación del problema, (3) el pensamiento computacional mejora la ejecución del algoritmo.

Tabla 1. Batería de pruebas para evaluar la resolución de problemas complejos según el marco MCS (Greiff, Wüstenberg y Funke, 2012). Fuente: Elaboración propia.

Nivel de complejidad	Representar el problema		Ejecutar el algoritmo del problema	
	Fase: Explorar y comprender	**Fase: Representar y formular**	**Fase: Planear y ejecutar**	**Fase: Observar y reflexionar**
Nivel 2	Billetes. Pregunta 2 puntuación parcial	PREGUNTA ELIMINADA	Agenda. Pregunta 2	Vídeo. Pregunta 2 puntuación parcial
Nivel 3	Vídeo. Pregunta 1	Climatizador. P1	Billetes. Pregunta 1	Vídeo. Pregunta 2 puntuación total
Nivel 4	Coche. Pregunta 1	Tren. Pregunta 1	Coche. Pregunta 3 puntuación parcial	Billetes. Pregunta 3
Nivel 5	Billetes. Pregunta 2 puntuación total	Coche. Pregunta 2 puntuación parcial	PREGUNTA ELIMINADA	Tren. Pregunta 2
Nivel 6	Coche. Pregunta	Coche. Pregunta 2 puntuación total	Coche. Pregunta 3 puntuación total	PREGUNTA ELIMINADA

En la primera hipótesis se recoge el resultado de las 12 preguntas que contiene finalmente la prueba, siendo la máxima puntuación posible un 18. En la segunda hipótesis se recoge el resultado de las pruebas de exploración y entendimiento, es decir, las pruebas de las dos primeras columnas de la tabla 1. El resultado máximo posible es de 9 puntos. En la tercera hipótesis se analizan las preguntas de planificación y de monitorización y reflexión, que se encuentran en las dos columnas de la derecha en la tabla 1. El resultado máximo posible es igualmente de 9 puntos.

3. Resultados

Para hallar los resultados se han hecho pruebas descriptivas, como son media y desviación típica, y pruebas inferenciales, utilizando la prueba T de Student para el contraste significativo de las hipótesis comparando las medias de ambos grupos. Previamente se ha confirmado la distribución normal de la muestra realizando una prueba sobre la bondad de ajuste del modelo (prueba de Shapiro-Wilk), se ha obtenido un histograma como comprobar los resultados, y se comprobado la homocedasticidad de la muestra en cada prueba T de Student atendiendo al estadístico de Levene (Ortega-Ruipérez, 2018).

En la tabla 2 se puede ver un resumen de las medias de las diferentes hipótesis en ambos grupos, antes y después del curso maker de robótica, mientras que en la tabla 3 se puede ver los estadísticos P-Valor obtenidos en las diferentes hipótesis, de cara a su interpretación. Al observar la diferencia de medias en cada grupo entre la puntuación obtenida en el pretest y la puntuación obtenida en el postest, podemos apreciar la mejora significativa del grupo experimental respecto al grupo control en todas las fases de la resolución de problemas.

Tabla 2. Resumen de las puntuaciones medias obtenidas en las diferentes fases de la resolución de problemas complejos, tanto en el pretest como en el postest, en ambos grupos de investigación. Fuente: Elaboración propia.

Fase de resolución de problemas	Grupo	Puntuación media pretest	Puntuación media postest
Resolución de problemas complejos (Proceso completo)	Experimental	6.9524	9.9048
	Control	7.0000	7.2353
Adquisición del conocimiento (Representación del problema)	Experimental	3.2381	4.9048
	Control	3.2353	3.5294
Aplicación del conocimiento (Ejecución del algoritmo)	Experimental	3.7143	5.0000
	Control	3.7647	3.7059

Si observamos las medias relativas a la primera hipótesis, si el pensamiento computacional mejora la resolución de problemas complejos, vemos que en el grupo experimental la mejora ha sido de casi 3 puntos, mientras en el grupo control sólo se ha mejora 0.23 puntos. Recordemos que la máxima puntuación de la prueba eran 18 puntos (tabla 2). Si nos centramos en cada una de las fases principales: representación del problema y ejecución del algoritmo, vemos cómo en la primera hipótesis la mejora del grupo experimental es de 1.66 puntos sobre 9 posibles, mientras que en el grupo control la mejora es de 0.29 puntos. En la segunda hipótesis obtenemos un resultado en la misma línea, mientras que el grupo experimental mejora en 1.28 puntos sobre 9 posibles, el grupo control se mantiene igual (tiene un descenso de 0.06, lo que no resulta significativo para concluir un empeoramiento).

En cuanto a las pruebas estadísticas inferenciales, en las que se ha utilizado la prueba T de Student para muestras independientes, los resultados muestran que en el postest sigue sin haber diferencias significativas entre los grupos experimental y control (tabla 3). Sin embargo, como se puede apreciar si comparamos estos resultados respecto al pretest, el estadístico ha descendido bruscamente hasta situarse en valores mucho más próximos a 0.05, que ha sido el estadístico de referencia tomado al asumir un 95% de intervalo de confianza. En este caso vemos en la tabla 3 cómo la resolución de problemas en general y la ejecución del algoritmo han obtenido valores de 0.07, con lo que prácticamente se confirman estas dos hipótesis.

Tabla 3. Resumen de los P-Valor obtenidos para las diferentes hipótesis en relación con las fases de la resolución de problemas complejos, tanto en el pretest como en el postest. Fuente: Elaboración propia.

Fase de resolución de problemas	P-valor pretest	P-valor postest
Resolución de problemas complejos (Proceso completo)	0.967	0.072
Adquisición del conocimiento (Representación del problema)	0.997	0.123
Aplicación del conocimiento (Ejecución del algoritmo)	0.930	0.071

4. Conclusión

Después del análisis de los resultados, podemos concluir que se ha cumplido el objetivo de proporcionar una intervención adecuada para desarrollar el pensamiento computacional. Los resultados obtenidos muestran que la propuesta desarrollada es adecuada para trabajar este pensamiento. Además, los resultados obtenidos en el análisis inferencial indican que el desarrollo del pensamiento computacional es más complejo y requiere mayor tiempo de dedicación para el desarrollo que una formación concreta, como la intervención de 30 horas que se ha llevado a cabo en esta investigación. Estos resultados encajan con el hecho de que este pensamiento es propio de científicos de la programación que ya son expertos en la materia, la habilidad en estos procesos se va adquiriendo poco a poco con la práctica.

Con esta investigación hemos logrado conocer mejor cómo se desarrolla el pensamiento computacional, ya que estos resultados nos han permitido confirmar que es más sencillo aplicar en un primer momento el pensamiento computacional a la ejecución del algoritmo, y por lo tanto debemos priorizar esta fase para crear actividades iniciales en la enseñanza de la programación y de la robótica, que permitan desarrollar el pensamiento computacional con un proceso de aprendizaje más sencillo y asequible cognitivamente. Por tanto, podemos confirmar que la estrategia didáctica que se ha planteado favorece el desarrollo del pensamiento computacional, y debemos seguirla si queremos desarrollar este pensamiento a través de actividades de programación. Los resultados obtenidos nos han permitido recoger las siguientes consideraciones, que se deberán tener en cuenta para crear una propuesta didáctica de programación o de robótica con el objetivo de desarrollar el pensamiento computacional.

Como paso previo antes de poder afrontar problemas, se deben conocer las bases y las estrategias de programación. Cuando los aprendices tengan un suficiente dominio del campo, el conocimiento de estas estrategias les facilitará la ejecución de los pasos para resolver problemas. La forma para que los aprendices desarrollen el pensamiento computacional a través de la programación es plantear retos que supongan un problema a resolver por parte de los estudiantes, que supongan motivadores. En este caso se pueden utilizar enfoques contextualizados que despierten su interés, como puede ser utilizar una filosofía DIY en la que comprendan que ellos mismos pueden crear la tecnología que habitualmente compran. Para ello, será fundamental explicitar los pasos clave para trabajar la resolución de un problema (Marais y Bradshaw, 2015), utilizando estrategias propias del pensamiento computacional. Es decir, debemos proponer el uso de la descomposición del problema, y el uso de automatización, paralelismo y simulación, enfocadas para mejorar el tratamiento de datos, la abstracción del problema y la creación de los algoritmos que permitirán resolver el problema. Debemos insistir en estos pasos desde el primer reto que se les plantee hasta que ellos mismos ejecuten esos pasos de forma automática.

Durante los primeros retos que se propongan a los aprendices, el profesorado debe enfocarse a que el alumnado aprenda a utilizar una estrategia computacional para afrontar el proceso de resolución de problemas. Para ello, debemos proponer retos en los que se muestre claramente la representación del problema y asegurarnos de que los aprendices entienden el problema y todas sus variables. De esta forma, el profesor o profesora muestra cómo se llega a la representación del problema utilizando una estrategia computacional, cómo se tienen en cuenta las variables y los

razonamientos necesarios para llegar a esa adquisición del conocimiento sobre el problema. Por su parte, los aprendices se centrarán únicamente en resolver el problema, poniendo en marcha una estrategia computacional.

Posteriormente, cuando el desarrollo de este pensamiento sea superior, los aprendices deberán abordar la representación del problema con una estrategia computacional. Para ello, los aprendices se pueden apoyar en el uso de mapas mentales, diagramas de flujo y cualquier otra ayuda que les sirva para organizar la información y facilitar su entendimiento de cara a la resolución del problema. El alumnado, al haber sido instruido por el profesor o profesora durante los primeros retos sobre cómo se construye la representación, deberá aplicar los mismos razonamientos para construir la representación del problema utilizando una estrategia computacional.

Todo este conocimiento es fundamental para enseñar programación y robótica de forma que aseguremos el desarrollo del pensamiento computacional, de cara a la inclusión de asignaturas de programación en la educación formal de una forma no reproductiva, como afirmaba Zapata-Ros (2015), aunque sería también aplicable a contextos informales (Asensio y Asenjo, 2011; Santacana, Asensio y Llonch, 2018). Nuestra misión ha sido buscar la forma en la que la enseñanza de la programación resultase beneficiosa para todos, para avanzar de forma efectiva en la resolución de problemas, como competencia básica deseable en cualquier currículo educativo.

Como limitación de la investigación, que ha podido influir en los resultados y se deberá tener en cuenta para futuras investigaciones, se ha dado el inconveniente de haber desechado tres preguntas de las pruebas, lo que nos ha hecho perder una mayor variedad en los resultados que permita discriminar más entre los grupos. Se deberían crear más preguntas que completen los niveles de las diferentes fases de resolución de problemas. Es preciso mencionar que se realizaron análisis inferenciales antes de desechar las tres preguntas y en ese caso sí que se obtuvieron diferencias significativas entre los grupos, en la resolución de problemas en general y en la ejecución del algoritmo.

5. Referencias

Asenjo, E. y Asensio, M. (en prensa). The force under suspicion: building a perceived interactivity scale in museums. *Journal on Computing and Cultural Heritage_JOCCH*

Asensio, M. y Asenjo, E. (Eds.) (2011). *Lazos de Luz Azul. Museos y Tecnologías 1, 2 y 3.0*. Barcelona: Editorial Universitat Oberta de Catalunya.

Brennan, K. y Resnick, M. (2012) New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 annual meeting of the American Educational Research Association*. Vancouver, Canada

Chiu, T. K. F., y Churchill, D. (2015). Exploring the characteristics of an optimal design of digital materials for concept learning in mathematics: Multimedia learning and variation theory. *Computers y Education*, 82, 280–291. http://dx.doi.org/10.1016/j.compedu.2014.12.001

CSTA y ISTE (2011) *Computational Thinking Leadership Toolkit, First edition*. Computer Science Teachers Association (CSTA) y International Society for Technology in Education (ISTE). https://goo.gl/syFwSF

Greiff, S., Wüstenberg, S., y Funke, J. (2012). Dynamic problem solving: A new assessment perspective. *Applied Psychological Measurement*, *36*(3), 189-213. http://dx.doi.org/10.1177/0146621612439620

Herde, C. N., Wüstenberg, S., y Greiff, S. (2016). Assessment of complex problem solving: What we know and what we don't know. *Applied Measurement in Education*, 29(4), 265-277. http://dx.doi.org/10.1080/08957347.2016.1209208

Hmelo-Silver, C.E. (2012) International perspectives on problem-based learning: context, cultures, challenges and adaptations. *Interdisciplinary Journal of Problem-Based Learning*, *6,* 10-15. http://dx.doi.org/10.7771/1541-5015.1310

Holyoak, K. J. y Morrison, R. G. (Eds.)(2012) *The oxford handbook of thinking and reasoning*. Oxford University Press. http://dx.doi.org/10.1093/oxfordhb/9780199734689.001.0001

Jiménez-Rasgado, G. (2018). La programación como fuente motivadora para la construcción del conocimiento y el desarrollo de habilidades de pensamiento. *International Journal of Studies in Educational Systems, 2(8),* 269-278.

Kafai, Y. B. y Burke, Q. (2017) Computational Participation: Teaching Kids to Create and Connect Through Code. In P.J. Rich, C.B. Hodges (eds.), *Emerging Research, Practice, and Policy on Computational Thinking, Educational Communications and Technology: Issues and Innovations*. Springer.

Koh, K. H., Nickerson, H., Basawapatna, A., y Repenning, A. (2014, June). Early validation of computational thinking pattern analysis. In *Proceedings of the 2014 conference on Innovation y technology in computer science education,* 213-218. ACM. http://dx.doi.org/10.1145/2591708.2591724

Lu, Y., Bridges, E.M. y Hmelo-Silver, C.E. (2014) Problem-Based Learning. In: Sawyer, R.K. (Ed.) *The Learning Sciences*, pp. 298-318. N.Y.: Cambridge University Press.

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

Marais, C., y Bradshaw, K. (2015). Problem-solving ability of first year CS students: A case study and intervention. In *Proceedings of the 44th Conference of the Southern African Computers Lecturers' Association*.

Mayer, RE. y Estrella, G. (2014). Benefits of emotional design in multimedia instruction. *Learning and Instruction*, 33, 12-18. http://dx.doi.org/10.1016/j.learninstruc.2014.02.004

Moreno-Leon, J., Robles, G., y Roman-Gonzalez, M. (2015). Dr. Scratch: Análisis Automático de Proyectos Scratch para Evaluar y Fomentar el Pensamiento Computacional. *Revista de Educación a Distancia*, (46).

Muñoz, R., Barcelos, T. S., Villarroel, R., y Silveira, I. F. (2017). Using Scratch to Support Programming Fundamentals. *International Journal on Computational Thinking, 1(1),* 68-78. http://dx.doi.org/10.14210/ijcthink.v1n1.p68

National Research Council (2010) *Report of a Workshop on the Scope and Nature of Computational Thinking*. Washington, DC: The National Academies Press

OECD (2010). *PISA 2012 field trial: problem solving framework*. Paris: OECD

Ortega-Ruipérez, B. (2018) *Pensamiento computacional y resolución de problemas* (Tesis doctoral). Universidad Autónoma de Madrid, Madrid. https://goo.gl/ortqFs

Park, S. Y., Song, K. S., y Kim, S. (2015). EEG Analysis for Computational Thinking based Education Effect on the Learners'Cognitive Load. *Proceedings of the Applied Computer and Applied Computational Science (ACACOS'15),* 23-25. Kuala Lumpur, Malaysia.

Ritchhart, R. y Perkins, D. N. (2005) Learning to Think: The Challenges of Teaching Thinking. En Holyoak, K. J. y Morrison, R. G. (Eds.) *The Cambridge Handbook of Thinking and Reasoning,* pp. 775-802. Cambridge University Press.

Robles, G., Moreno-León, J., Aivaloglou, E., y Hermans, F. (2017). Software clones in Scratch projects: On the presence of copy-and-paste in Computational Thinking learning. In *Software Clones (IWSC), 2017*

IEEE 11th International Workshop on (pp. 1-7). Austria.

Sáez-López, J. M. y Cózar-Gutiérrez, R. (2017) Pensamiento computacional y programación visual por bloques en el aula de Primaria. *Educar, 53(1)*, 129-146. http://dx.doi.org/10.5565/rev/educar.841

Santacana, J. Asensio, M. y Llonch, N. (Eds.) (2018). *App, arqueología & m-learning. Reconstruir, restituir interpretar y evaluar APP.* Barcelona: Rafael Dalmau.

Selby, C. y Woollard, J. (2013) *Computational thinking: the developing definition.* University of Southampton (E-prints). https://goo.gl/sKdd9G

Shute, V. J., Sun, C., y Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158. https://doi.org/10.1016/j.edurev.2017.09.003

Stahl, G., Koschmann, T. y Suthers, D. (2014). Computer-Supported Collaborative Learning. In: Sawyer, R.K. (Ed.) *The Learning Sciences,* pp. 479-500. N.Y.: Cambridge University Press.

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. *Learning and instruction*, *4*(4), 295-312.

Valverde-Berrocoso, J.; Fernández-Sánchez, M. y Garrido-Arroyo, M. (2015) El pensamiento computacional y las nuevas ecologías del aprendizaje. *Revista de Educación a Distancia*. (46) http://dx.doi.org/10.6018/red/46/3

Wing, J. M. (2006) Computational thinking. *Communications of the ACM, 49*, 33-35. http://dx.doi.org/10.1145/1118178.1118215

Wing, J. M. (2008) Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 366,* 3717-3725. The Royal Society.

Wing, J. M. (2011) Research Notebook: Computational Thinking: What and Why? *The Link.* Pittsburgh, PA: Carneige Mellon. Recuperado de https://www.cs.cmu.edu/link/research-notebook-computational-thinkingwhat-and-why

Wing, J. M. (2014). Computational thinking benefits society. *Social Issues in Computing*. New York: Academic Press.

Wing, J.M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology, 25(2)*, 7-14. http://dx.doi.org/10.17471/2499-4324/922

Zapata-Ros, M. (2015) Pensamiento computacional: Una nueva alfabetización digital. *Revista de Educación a Distancia*. Universidad de Murcia. http://dx.doi.org/10.6018/red/46/4