PROCEDIMIENTO METODOLÓGICO BASADO EN EL PENSAMIENTO COMPUTACIONAL PARA EL APRENDIZAJE DE PROGRAMACIÓN ESTRUCTURADA

METHODOLOGICAL PROCEDURE BASED ON COMPUTATIONAL THINKING FOR THE STRUCTURED PROGRAMMING LEARNING PROCESS

https://doi.org/10.5281/zenodo.14805011

AUTORES:

Dennis Fernando Burgasi Cushicagua[1]*

Patricio Medina-Chicaiza[2]

DIRECCIÓN PARA CORRESPONDENCIA: dfburgasi@pucesa.edu.ec – dennis.burgasi96@gmail.com

Fecha de recepción: 06 / 12 / 2024
Fecha de aceptación: 13 / 12 / 2024

RESUMEN

La necesidad surge por la insuficiencia de desarrollo en el pensamiento lógico – matemático, esto dificulta a los estudiantes en la creación de soluciones a diferentes ejercicios; lo que genera una frustración hacia la asignatura de programación. La importancia de la propuesta radica en que, mediante la aplicación de un enfoque metodológico, los estudiantes pueden descomponer, revisar y analizar los ejercicios, esto reduce el tiempo necesario para encontrar soluciones. Este enfoque permite simplificar casos complejos y desarrollar una secuencia de pasos que conducen a la solución mediante el uso de herramientas computacionales.

La presente investigación tiene como objetivo proponer un procedimiento estructurado para el aprendizaje de programación. El estudio adopta un enfoque cuantitativo y utiliza el método cuasiexperimental, con un diagnóstico realizado a través del test de Román González, que mide las habilidades relacionadas con la resolución de problemas. La evaluación se llevará a cabo con 74 estudiantes de primero de bachillerato general unificado de la Unidad Educativa "La Inmaculada" Latacunga-Ecuador. Se tiene como resultado un procedimiento

[1]* Egresado Maestría en Innovación en Educación, Pontificia Universidad Católica, dfburgasi@pucesa.edu.ec , https://orcid.org/0009-0007-5205-3023

[2] Licenciado en Educación, Doctor en Ciencias de la Educación, Pontificia Universidad Católica del Ecuador Sede Ambato; Universidad Técnica de Ambato, pmedina@pucesa.edu.ec ; ricardopmedina@uta.edu.ec

metodológico con 4 fases: diagnóstico, planificación, aplicación y control.***Palabras clave:*** *Herramientas computacionales, pensamiento lógico – matemático, planificación.*

ABSTRACT

The need arises from the insufficient development of logical-mathematical thinking, which makes it difficult for students to create solutions to different exercises; this leads to frustration with the programming subject. The importance of the proposal lies in that, through the application of a methodological approach, students can decompose, review, and analyze exercises, reducing the time required to find solutions. This approach allows for simplifying complex cases and developing a sequence of steps that lead to solutions through the use of computational tools.

The present research aims to propose a structured procedure for learning programming. The study adopts a quantitative approach and uses the quasi-experimental method, with a diagnosis carried out through the Román González test, which measures problem-solving skills. The evaluation will be conducted with 74 students from the first year of Unified General High School at the "La Inmaculada" Educational Unit in Latacunga-Ecuador. The result is a methodological procedure with four phases: diagnosis, planning, application, and control.***Keywords:*** *Computational tools, logical-mathematical thinking, planning.*

INTRODUCCIÓN

El *pensamiento computacional (PC)* es un conjunto de habilidades cognitivas que permite a las personas abordar problemas de manera sistemática y lógica, descomponerlos en partes más manejables, identificar patrones, crear abstracciones y diseñar algoritmos para encontrar soluciones efectivas. Esta forma de pensar se ha reconocido como esencial no solo para la programación, sino también para una amplia gama de disciplinas y actividades cotidianas, lo que la convierte en una competencia transversal valiosa en la educación moderna (Shute et al., 2021). Al incorporar el *(PC)* en el currículo educativo, los estudiantes desarrollan la capacidad de enfrentar problemas complejos y crear soluciones innovadoras mediante enfoques estructurados y algoritmos, una habilidad fundamental en la era digital actual.

Investigaciones recientes han ampliado la comprensión del *(PC)*, al destacar su importancia en la educación temprana y su impacto en el desarrollo de otras habilidades cognitivas, como el razonamiento lógico y la resolución de problemas. Aydemir y Çiftci (2022) subrayan que la enseñanza del *(PC)* desde edades tempranas no solo mejora la competencia en ciencias de la computación, sino que también fortalece habilidades matemáticas y analíticas.

De esta manera, el conocimiento y la aplicación de técnicas computacionales, así como la comprensión de códigos, permiten afianzar destrezas en el estudiante, como procesar datos y resolver cuestiones lógicas. Estas habilidades se convierten en una herramienta cognitiva que guarda una estrecha relación con estructuras y conexiones mentales, esto facilita el aprendizaje relacionado con la programación informática y algorítmica, actividades clave del

pensamiento de orden superior (Wing, 2006). Asimismo, Tang et al. (2022) argumentan que la integración del *(PC)* en diversas disciplinas académicas facilita una comprensión más profunda de los contenidos y promueve un aprendizaje interdisciplinario más efectivo. Estas perspectivas destacan la relevancia y la necesidad de investigar y aplicar el *(PC)* en distintos contextos educativos.

Por otra parte, los diagramas de flujo son herramientas visuales fundamentales en la programación estructurada, utilizados para representar de manera gráfica el flujo de control y las operaciones lógicas de un algoritmo o proceso. Estos diagramas permiten descomponer problemas complejos en pasos secuenciales, facilitando la comprensión y el diseño de soluciones de programación. Cada elemento del diagrama, como los rectángulos para procesos, los rombos para decisiones y los óvalos para inicio y fin, cumple una función específica que ayuda a estructurar el pensamiento lógico necesario para desarrollar código eficiente y libre de errores (Abdullah et al., 2021). La claridad y simplicidad de los diagramas de flujo los convierten en una herramienta educativa efectiva, especialmente para aquellos que están aprendiendo los fundamentos de la programación.

Además, los diagramas de flujo desempeñan un papel crucial en la fase de planificación y diseño de software, ya que permiten a los programadores prever y evitar posibles problemas antes de escribir el código. Su uso no solo facilita la comunicación entre desarrolladores, sino que también sirve como documentación para futuros desarrollos o revisiones del software (Ibrahim et al., 2022). Según López-Pernas et al. (2023), la integración de diagramas de flujo en la enseñanza de la programación estructurada mejora significativamente la capacidad de los estudiantes para conceptualizar y construir algoritmos efectivos. Estos diagramas sirven como puentes entre el pensamiento abstracto y la implementación práctica, lo que los convierte en herramientas esenciales en la educación y el desarrollo de software.

Sin embargo, la falta de desarrollo en el pensamiento lógico-matemático genera frustración en los estudiantes, esto dificulta el logro de objetivos propuestos en el aprendizaje de programación. Por esta razón, el estudio propone un procedimiento metodológico basado en el (PC), orientado al aprendizaje de la programación estructurada. Donde, la motivación del estudio radica en presentar una alternativa en el aprendizaje de programación estructurada a través de ejercicios prácticos donde la tecnología sea el pilar fundamental.

METODOLOGÍA

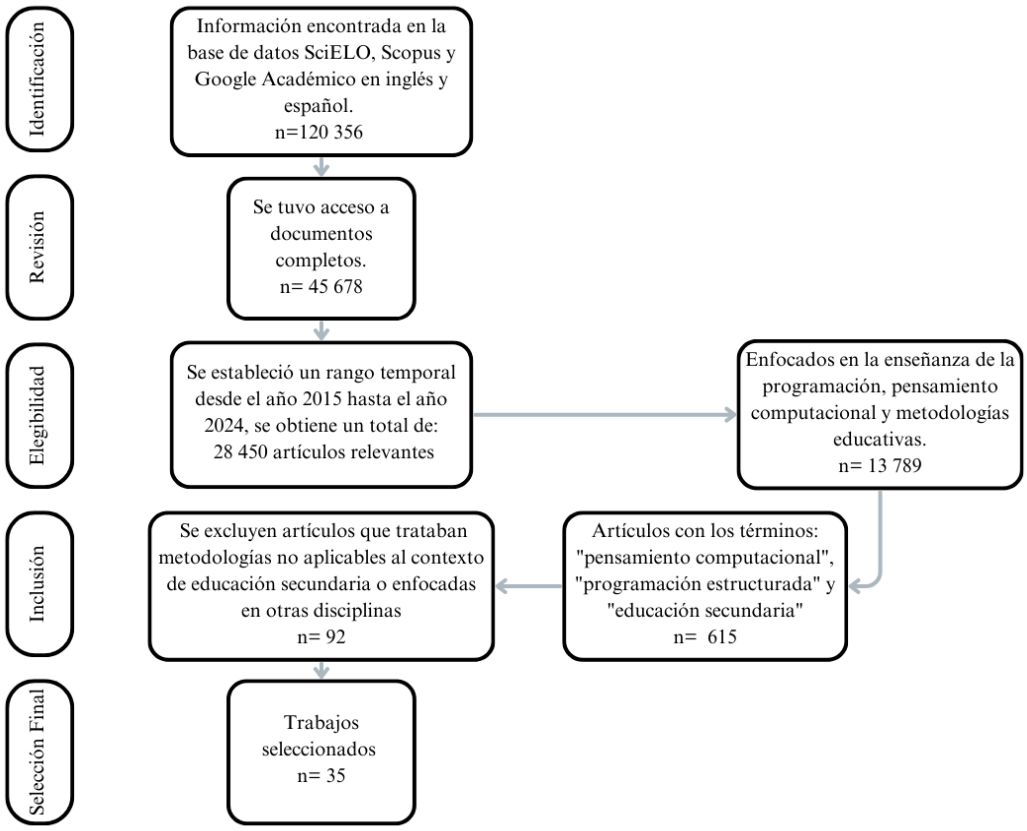
Se llevó a cabo una revisión sistemática en las bases de datos SciELO, IEEE Xplorer, Scopus y Google Académico, conformada por 120 356 artículos científicos, de los cuales se tuvo acceso con texto completo a 45 678. Se estableció un rango temporal desde el año 2015 hasta el año 2024, se obtiene un total de 28 450 artículos relevantes. Posteriormente, se seleccionaron únicamente aquellos que estuvieran enfocados en la enseñanza de la programación, pensamiento computacional y metodologías educativas, lo que redujo la muestra a 13 789 artículos.

Se aplicaron algoritmos de búsqueda con los términos "pensamiento computacional", "programación estructurada" y "educación secundaria" utilizando operadores booleanos "AND" y "NOT" para excluir artículos no relacionados con estudiantes de secundaria o niveles educativos inferiores. Esto arrojó un total de 615 artículos, de los cuales se excluyeron aquellos que trataban metodologías no aplicables al contexto de educación secundaria o enfocadas en otras disciplinas, quedando un total de 92 artículos aptos para su revisión.

Finalmente, tras una revisión detallada del contenido y enfoque de cada artículo, se seleccionaron 35 estudios que proporcionan evidencia relevante sobre la implementación de pensamiento computacional en la educación secundaria, específicamente aplicados a la enseñanza de la programación estructurada.

La Figura 1 ilustra el proceso metodológico utilizado para la identificación de los hallazgos a través de esta revisión sistemática, se ha considerado bases de datos académicos.

Figura 1. *Diagrama de flujo para la selección de información*



Fuente: Elaboración propia

RESULTADOS

Basado en la metodología descrita por Meneses Paucar y Medina-Chicaiza (2021), la temática incluye un procedimiento metodológico presentado en 4 fases: diagnóstico, planificación, aplicación y control dentro del procedimiento metodológico para el aprendizaje de la programación.

Fase 1: Diagnóstico

La fase de diagnóstico se enfoca en identificar el nivel de habilidades y conocimientos previos de los estudiantes en cuanto al (PC), fundamental para abordar la programación estructurada. En este contexto, se emplea el *Test de Pensamiento Computacional* desarrollado por Román-González (2015), una herramienta validada que permite medir competencias clave como la descomposición de problemas, la abstracción y la capacidad de diseñar algoritmos. Esta evaluación proporciona datos cuantitativos sobre el estado actual de las habilidades computacionales de los estudiantes, permitiendo identificar áreas críticas que requieren fortalecimiento.

Adicional, se implementa la observación directa como método cualitativo para detectar dificultades en el aprendizaje. Este enfoque permite identificar problemáticas que podrían no ser evidentes en una prueba estandarizada, como la falta de motivación, frustración o dificultad en la comprensión de conceptos abstractos.

Los resultados del test se determinan de la siguiente manera:

Tabla 1. *Porcentaje Test PC*

	Preguntas	Aciertos	Errores	Sin Respuesta	Total	% Aciertos	Promedio Aciertos	% Errores	Promedio Errores
Descomposición	1	68	3	1	72	94,44	76,79	4,17	22,22
	2	65	7	0	72	90,28		9,72	
	3	61	11	0	72	84,72		15,28	
	4	26	44	2	72	36,11		61,11	
	5	58	14	0	72	80,56		19,44	
	6	63	9	0	72	87,50		12,50	
	7	46	24	2	72	63,89		33,33	
Rec	8	21	48	3	72	29,17	58,33	66,67	40,08

	9	68	4	0	72	94,44		5,56	
	10	46	26	0	72	63,89		36,11	
	11	53	19	0	72	73,61		26,39	
	12	23	47	2	72	31,94		65,28	
	13	45	26	1	72	62,50		36,11	
	14	38	32	2	72	52,78		44,44	
Abstracción	15	25	44	3	72	34,72	44,64	61,11	54,17
	16	22	50	0	72	30,56		69,44	
	17	31	40	1	72	43,06		55,56	
	18	40	31	1	72	55,56		43,06	
	19	42	29	1	72	58,33		40,28	
	20	34	38	0	72	47,22		52,78	
	21	31	41	0	72	43,06		56,94	
Diseño Algoritmos	22	16	56	0	72	22,22	26,59	77,78	72,62
	23	11	61	0	72	15,28		84,72	
	24	32	40	0	72	44,44		55,56	
	25	18	50	4	72	25,00		69,44	
	26	5	67	0	72	6,94		93,06	
	27	10	62	0	72	13,89		86,11	
	28	42	30	0	72	58,33		41,67	

Fuente: Elaboración propia

Los resultados obtenidos revelan un desempeño poco favorable en las temáticas de abstracción y diseño de algoritmos. En promedio, el 54.17% de los estudiantes evaluados no logra desarrollar adecuadamente habilidades de abstracción, lo que implica dificultades para identificar información esencial o representar problemas de manera simplificada. De igual manera el 72.62% de los estudiantes no pueden construir, completar o corregir un algoritmo para resolver una tarea. Aunque el 58.33% de los estudiantes demuestra habilidad para reconocer patrones, el 40.08% no lo consigue. Además, un 1.59% de los evaluados no intentó responder a esta pregunta. De esta manera. los datos obtenidos subrayan la necesidad urgente de revisar los enfoques pedagógicos empleados para la enseñanza de la abstracción y el diseño de algoritmos. Las bajas tasas de éxito en estas temáticas clave sugieren que los estudiantes no internalizan estas habilidades fundamentales, lo cual es preocupante dado su relevancia en el pensamiento computacional. La habilidad de identificar patrones, aunque

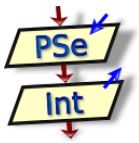
ligeramente más alta, también muestra deficiencias significativas. Estos hallazgos indican la importancia de desarrollar enfoques más personalizados y dinámicos que permitan a los estudiantes no solo memorizar conceptos, sino también aplicarlos de manera efectiva en contextos variados.

Fase 2: Planificación

La fase de planificación se enfoca en el diseño de un programa educativo que integra el (PC) en el aprendizaje de la programación estructurada, adaptado a las necesidades identificadas en el diagnóstico. Esta etapa implica la definición de objetivos de aprendizaje, selección de contenidos, diseño de estrategias didácticas, organización de recursos y planificación temporal.

En este apartado se detallan los participantes y los recursos utilizados. Los participantes incluyen al docente encargado de la asignatura de programación, quien imparte clases en los niveles de básica superior y bachillerato, y a los estudiantes de primero de bachillerato del paralelo “A”, quienes fueron parte del proceso. Los recursos para el diseño del procedimiento metodológico incluyen un computador por estudiante en el laboratorio de computación de la institución, todos ellos equipados con el *software* Pseint previamente instalado.

Figura 2. *Logo Pseint*



Fuente: https://pseint.sourceforge.net/

PSeInt es ampliamente recomendado como una herramienta educativa para el desarrollo del pensamiento computacional, especialmente en el contexto de la enseñanza de la programación estructurada. Esta plataforma permite a los estudiantes aprender algoritmos de manera intuitiva y clara, al trabajar con pseudocódigo y diagramas de flujo. Como menciona Daza-Castellanos et al. (2020), "PSeInt facilita la adquisición de habilidades algorítmicas, brindando un entorno accesible para el desarrollo del pensamiento computacional, lo que resulta clave en los primeros pasos de la formación en programación" (p. 47). Esta herramienta contribuye significativamente a que los estudiantes puedan enfocar su aprendizaje en la lógica y la resolución de problemas sin la complejidad de los lenguajes de programación formales.

A continuación, en la tabla 2 se presenta la planificación de clase destinada al grupo de estudio seleccionado (1 BGU 'A'), la cual se basa en el aprendizaje a través del pensamiento crítico. En esta planificación se consideran las destrezas a desarrollar, los contenidos analizados, y los recursos utilizados. Asimismo, se incluyen los indicadores de evaluación que permiten al docente medir el progreso en el desarrollo de habilidades y el proceso de aprendizaje. Este proceso consta de tres fases: anticipación, construcción y consolidación,

con el objetivo de fomentar un aprendizaje significativo y facilitar la generación de nuevo conocimiento a través de diversas actividades.

Tabla 2. *Planificación Propuesta*

Planificación Micro curricular	
Participantes:	1 BGU "A" (37 estudiantes)
Tiempo:	6 semanas
Lugar:	Laboratorio de computación
Nombre de la Unidad:	**Aprendo programación con Pseint**
Objetivo General:	Desarrollar la habilidad del pensamiento computacional para el aprendizaje de programación estructurada al utilizar el programa Pseint.
Ejes transversales:	• Innovación • Solidaridad • Justicia • Caridad

Destrezas con Criterio de Desempeño	**Indicadores de Evaluación**	**Orientaciones metodológicas**	
		Proceso de aprendizaje	**Actividades en Casa (tareas)**
Descomposición: Divide correctamente un problema complejo en tres o más subproblemas manejables. **Reconocimiento de Patrones**: Identifica al menos dos patrones comunes en problemas similares y reutiliza eficientemente algoritmos. **Abstracción**: Selecciona y modela los aspectos esenciales de un problema, omitiendo detalles irrelevantes. **Diseño Algorítmico**: Diseña al menos tres algoritmos correctos y sus respectivos diagramas de flujo. **Programación Estructurada**: Implementa correctamente soluciones algorítmicas mediante estructuras de control en PSeInt. **Lógica Algorítmica**: Resuelve problemas utilizando estructuras de	Identifica al menos tres componentes básicos en un problema. Identifica patrones recurrentes en problemas. Selecciona correctamente los elementos esenciales de un problema. Diseña tres diagramas de flujo correctos y coherentes. Implementa soluciones funcionales en PSeInt.	**ANTICIPACIÓN:** Se introduce el concepto de descomposición de problemas en la programación mediante un conversatorio. Los estudiantes anticipan cómo dividir problemas complejos para simplificarlos mediante un juego de roles. **CONSTRUCCIÓN:** Identifica y aplica patrones en problemas de programación mediante la reutilización de soluciones previas. Abstrae problemas y se centra en los elementos más importantes para crear modelos simplificados. Diseña algoritmos efectivos y representa sus	- Desarrollo de una consulta sobre los diagramas de flujo o DFD. - Crea una presentación corta sobre cómo descomponer problemas para luego darles solución.

control en al menos el 80% de los ejercicios.	Resuelve problemas algorítmicos aplicando estructuras de control (bucles y condicionales).	soluciones mediante diagramas de flujo que reflejen el proceso lógico de la programación. **CONSOLIDACIÓN:** Implementa los algoritmos diseñados en PSeInt, con un enfoque en la resolución práctica de problemas de programación estructurada. Aplica la lógica matemática y las estructuras de control aprendidas para resolver problemas algorítmicos en un entorno de desarrollo.	

Fuente: Elaboración propia

Selección de contenidos

Los contenidos se eligen al considerar su relevancia para alcanzar los objetivos propuestos. Se incluyen conceptos fundamentales de programación estructurada y (PC), como variables, estructuras de control, funciones y algoritmos básicos; tal cual, se observa en la Tabla 3.

Tabla 3. *Contenido y Recursos*

Contenido	**Descripción**	**Recursos**
Variables	Introducción al uso y definición de variables para almacenar datos.	Manual de Pseint y el libro *Fundamentals of Computer Programming with Algorithms and Data Structures* de Halterman (2012), que ofrece una introducción clara al uso de variables en programación.
Estructuras de Control	Uso de estructuras como condicionales (*si, sino*) y bucles (*mientras, para*).	El libro *Estructuras de datos y algoritmos en Pseint* de Gómez Cárdenas (2019) y tutoriales de plataformas como "Programación Fácil con Pseint".
Funciones	Definición, creación y uso de funciones para modularizar el código.	*Introducción a la Programación con Pseint* de Rodríguez Ramírez (2020), que cubre la creación y

		uso de funciones en programación estructurada.
Algoritmos básicos	Ejercicios prácticos de algoritmos simples (como ordenación, búsqueda y operaciones matemáticas).	*Algorithm Design Manual* de Skiena (2020), que cubre el diseño de algoritmos y su aplicación. También se puede encontrar contenido en libros de texto como *Algoritmos y Programación en Pseint*.
Depuración de errores	Identificación y corrección de errores en el código para mejorar el proceso lógico.	*Software Debugging Techniques* de Binder (2010) ofrece una guía completa sobre la depuración de programas, y la documentación oficial de Pseint contiene ejemplos prácticos sobre depuración.
Resolución de problemas	Uso del pensamiento computacional para descomponer y resolver problemas complejos paso a paso.	*Computational Thinking Education* editado por Kong (2019), que aborda la resolución de problemas desde el enfoque del pensamiento computacional, aplicable a la programación estructurada en Pseint.
Ejecución y pruebas	Probar y ejecutar programas desarrollados para verificar su correcto funcionamiento.	*Test Driven Development: By Example* de Beck (2003), que aborda técnicas para pruebas de programas, y la documentación de Pseint cubre la ejecución y pruebas de código dentro de la plataforma.

Fuente: Elaboración propia.

Fase 3: Aplicación

La fase de aplicación pone en práctica el plan diseñado, se lleva a cabo las actividades pedagógicas que permite a los estudiantes desarrollar habilidades en (PC) aplicadas a la programación estructurada, obsérvese la Tabla 3. En esta etapa, se implementan las estrategias didácticas previamente planificadas, con un enfoque en el aprendizaje activo y la resolución de problemas, se utilizan herramientas tecnológicas y métodos colaborativos.

Tabla 4. *Pasos para las actividades*

Pasos	**Descripción de la Actividad**	**Objetivo**
Introducción al Problema	El docente presenta un problema inicial relacionado con la vida	Fomentar el interés en la resolución de problemas y en la

	cotidiana (como gestionar un inventario o simular un cajero automático).	aplicación práctica de la programación.
Descomposición del Problema	Los estudiantes deben dividir el problema en partes más pequeñas, identificando las operaciones necesarias y los datos requeridos.	Descomponer problemas complejos en partes manejables, facilitando la comprensión y estructuración del problema.
Identificación de Patrones	El docente guía a los estudiantes para reconocer patrones en los datos o en las operaciones que se repiten dentro del problema.	Desarrollar la habilidad de identificar patrones y similitudes en problemas, facilitando la creación de soluciones más eficientes.
Creación de Algoritmos	Los estudiantes diseñan un algoritmo en PSeInt para resolver cada parte del problema, utilizando estructuras de control y variables.	Aplicar el pensamiento lógico y estructurado en la creación de algoritmos, siguiendo un flujo secuencial en la resolución.
Implementación en PSeInt	Los estudiantes implementan el algoritmo diseñado en PSeInt, ejecutando el código y depurando errores para asegurar su correcto funcionamiento.	Introducir a los estudiantes a la codificación y depuración, reforzando el ciclo de prueba y error propio de la programación.
Validación de resultados	Se realizan pruebas con diferentes entradas de datos para verificar que el programa funcione de manera correcta en distintas situaciones.	Desarrollar la capacidad de validar y mejorar soluciones, mediante la comprobación de la funcionalidad y fiabilidad del programa.
Reflexión y retroalimentación	Los estudiantes reflexionan sobre el proceso seguido, discuten dificultades y reciben retroalimentación del docente para mejorar su comprensión.	Fomentar la autoevaluación y el pensamiento crítico sobre el proceso de resolución de problemas y la programación.

Fuente: Elaboración propia.

Desarrollo de actividades prácticas

Las actividades se centran en la resolución de problemas, para ello se utilizan algoritmos y diagramas de flujo (https://acortar.link/swZRh5). Los estudiantes comienzan con ejercicios básicos, como la descomposición de problemas sencillos en pasos lógicos, para progresivamente enfrentarse a desafíos más complejos. Según López-Pernas et al. (2023), la implementación de diagramas de flujo facilita la comprensión de la lógica subyacente en los algoritmos, lo cual es fundamental en la programación estructurada.

Tabla 5. *Ejercicios básicos: Conversión, Condicionales y conteo*

Ejercicios	Descripción	Objetivo	Pasos Lógicos

Convertir grados Celsius a Fahrenheit	Los estudiantes deben crear un algoritmo que convierta una temperatura dada en grados Celsius a Fahrenheit.	Aprender a identificar entradas (temperatura en Celsius), el proceso (fórmula de conversión) y la salida (temperatura en Fahrenheit).	• Solicitar al usuario que ingrese la temperatura en Celsius. • Aplicar la fórmula de conversión: F= (9/5) × C + 32 • Mostrar la temperatura en Fahrenheit como salida.
Calcular el área de un rectángulo	Los estudiantes deben diseñar un algoritmo que calcule el área de un rectángulo dadas sus dimensiones (ancho y alto).	Fomentar la capacidad de descomponer problemas geométricos en pasos lógicos.	• Pedir al usuario que ingrese el valor del ancho y la altura del rectángulo. • Aplicar la fórmula para calcular el área: Área=ancho×altura • Mostrar el área calculada como resultado.
Determinar si un número es par o impar	Se solicita a los estudiantes que diseñen un algoritmo para verificar si un número ingresado es par o impar.	Introducir estructuras de control básicas como la condicional (si-entonces).	• Pedir al usuario que ingrese un número. • Evaluar si el número es divisible entre 2 sin residuo (utilizando la operación módulo). • Si el residuo es 0, mostrar que el número es par; de lo contrario, mostrar que es impar.
Contar hasta un número dado	Los estudiantes diseñan un algoritmo que cuente desde 1 hasta un número dado por el usuario.	Introducir el uso de ciclos o bucles para repetir una acción.	• Solicitar al usuario que ingrese un número. • Iniciar un ciclo que comience desde 1 y se repita hasta alcanzar el número dado. • En cada iteración, mostrar el número actual.

Fuente: Elaboración propia.

Seguimiento y retroalimentación continua

El docente monitorea de cerca el progreso de los estudiantes, brinda retroalimentación en tiempo real sobre sus soluciones a los problemas planteados. De acuerdo con Pérez y Robles

(2021), el *feedback* inmediato y constructivo es esencial para que los estudiantes puedan corregir errores y mejorar su comprensión de los conceptos.

Fase 4: Control

La fase de control se enfoca en la evaluación final y el análisis del impacto del procedimiento metodológico implementado. En esta etapa, se recopilan y analizan los datos obtenidos durante la fase de aplicación, con el objetivo de medir el desarrollo del pensamiento computacional y la competencia en programación estructurada de los estudiantes. Este proceso es clave para garantizar que los objetivos del procedimiento hayan sido alcanzados y para identificar áreas de mejora.

Evaluación sumativa

Para evaluar los resultados de aprendizaje, se aplican pruebas específicas de programación estructurada y nuevamente el *test* de (PC) de Román-González (2015) para comparar el rendimiento con los resultados del diagnóstico inicial. El *test* se compone de preguntas cerradas, diseñadas en formato de opción múltiple, con el fin de evaluar habilidades relacionadas con la abstracción, descomposición, reconocimiento de patrones y diseño de algoritmos. Este formato de preguntas cerradas permite una evaluación objetiva y estandarizada de los estudiantes, sin incluir preguntas abiertas ni ejercicios prácticos en su estructura. El tiempo estimado para completar la evaluación es de 45 minutos, brindando a los participantes un margen adecuado para resolver las 28 preguntas que componen el test. Aunque el test no incluye una rúbrica formal como tal, su sistema de puntuación se basa en la cantidad de respuestas correctas, lo que permite una evaluación directa de las habilidades del estudiante en términos de (PC).

Análisis cualitativo y cuantitativo

Además de los resultados cuantitativos obtenidos mediante las pruebas, se realiza un análisis cualitativo basado en la observación directa y las percepciones del docente. Este análisis permite evaluar el desarrollo del (PC) en contextos prácticos y cómo los estudiantes abordan la resolución de problemas. Tal cuál lo señala López-Pernas et al. (2023), la combinación de ambos tipos de análisis ofrece una visión más integral del impacto de la intervención educativa.

Retroalimentación y ajustes

Los resultados obtenidos se discuten con los estudiantes a través de sesiones de retroalimentación individual y grupal, lo que les permite reflexionar sobre su propio aprendizaje y áreas de mejora. Según Tang et al. (2022), este tipo de retroalimentación fomenta el pensamiento crítico y la autorregulación, elementos fundamentales en el aprendizaje autónomo. Con base en los resultados y la retroalimentación recibida, el docente puede ajustar futuras implementaciones del procedimiento metodológico para optimizar su efectividad.

DISCUSIÓN

El desarrollo de un proceso metodológico basado en el (PC) para el aprendizaje de la programación estructurada ha cobrado relevancia en el ámbito educativo, especialmente en el contexto de los estudiantes de bachillerato. En este sentido, Aydemir y Çiftci (2022) señalan que integrar el (PC) en el currículo de programación no solo fortalece las habilidades lógicas y analíticas de los estudiantes, sino que también mejora su capacidad para resolver problemas de manera estructurada. Este enfoque, como argumenta López-Pernas et al. (2023), resulta crucial para mejorar la retención de conocimientos, ya que la aplicación práctica de conceptos abstractos a través de proyectos facilita la comprensión profunda de la programación estructurada.

El desarrollo del proceso metodológico basado en el pensamiento computacional, que en este estudio se estructura en cuatro fases: diagnóstico, planificación, aplicación y control, contribuye a mejorar el aprendizaje de los estudiantes. Investigaciones como la de Meneses & Salcedo (2021) mencionan que la implementación de estas fases en un proceso metodológico facilita el proceso de enseñanza-aprendizaje, especialmente en temáticas que pueden resultar complejas para los estudiantes. Además, Abdullah et al. (2021) destacan que el uso de diagramas de flujo y pseudocódigo, elementos claves en la enseñanza de la programación estructurada, favorecen la visualización de la lógica subyacente en los algoritmos, lo cual es fundamental para consolidar el aprendizaje.

Por otro lado, la necesidad de adaptar las metodologías a contextos específicos, como el de los estudiantes ecuatorianos, es subrayada por autores como Román-González (2015), quienes enfatizan que los diagnósticos adaptados a las realidades socioeducativas locales permiten una evaluación más precisa de las competencias en el (PC). En este sentido, la observación directa y el uso de pruebas estandarizadas, como el test de Román-González, son esenciales para identificar las principales dificultades de los estudiantes en la programación estructurada.

Finalmente, un proceso metodológico basado en el pensamiento computacional no solo mejora el aprendizaje de la programación estructurada, sino que también contribuye al desarrollo de habilidades cognitivas superiores, como la descomposición, reconocimiento, abstracción y diseño de algoritmos.

CONCLUSIONES

Los resultados obtenidos muestran un progreso significativo en el diseño de algoritmos, reflejado en la mayor precisión y coherencia de los diagramas de flujo creados por los estudiantes. El uso de diagramas como herramienta clave dentro del proceso metodológico fortaleció su capacidad para estructurar soluciones lógicas y eficientes, lo que contribuyó a

un dominio más sólido de los fundamentos de la programación. Este avance destaca la importancia del pensamiento computacional, ya que permitió a los estudiantes descomponer problemas complejos, identificar patrones y abstraer la información esencial, habilidades cruciales para el desarrollo de soluciones algorítmicas efectivas.

La metodología implementada permitió que los estudiantes identificaran patrones recurrentes en problemas de programación, lo que facilitó la reutilización de algoritmos previamente aprendidos. Al final del ciclo de enseñanza, los estudiantes pudieron aplicar conceptos abstractos y generar soluciones eficientes a nuevos problemas, lo que demuestra una comprensión profunda de los principios del (PC).

Los estudiantes alcanzaron un incremento en sus calificaciones en comparación con evaluaciones previas, lo que refleja el impacto positivo de la metodología basada en el (PC). La consolidación del conocimiento a través de ejercicios prácticos en PSeInt, junto con la integración de conceptos algorítmicos, permitió que los alumnos culminaran satisfactoriamente los ejercicios de programación planteados, alcanzando niveles de desempeño superiores a los esperados.

REFERENCIAS BIBLIOGRÁFICAS

Abdullah, M., Karim, A., & Rahim, N. A. (2021). Visual aids in programming education: A review on the effectiveness of flowcharts and pseudocode. *Journal of Computer Science Education*, 25(2), 95-110. https://doi.org/10.1080/10494820.2021.1871346

Aydemir, M., & Çiftci, S. K. (2022). The effect of computational thinking integrated into flipped classroom model on students' programming self-efficacy and problem-solving skills. *Education and Information Technologies, 27*(3), 3643-3667. https://doi.org/10.1007/s10639-021-10727-1

Daza-Castellanos, M. A., Gómez, F. J., & Sánchez, L. M. (2020). *Herramientas tecnológicas para la enseñanza de algoritmos y programación*. Editorial Universidad Nacional de Colombia.

Ibrahim, N., Ariffin, N., & Hamid, N. A. (2022). Enhancing problem-solving skills in programming through flowchart-based learning. *Computers in Education, 130*, 104567. https://doi.org/10.1016/j.compedu.2022.104567

López-Pernas, S., Gordillo, A., & Quemada, J. (2023). The impact of flowchart-based instruction on programming learning outcomes. *Education and Information Technologies, 28*(1), 153-172. https://doi.org/10.1007/s10639-022-11128-8

Meneses Paucar, S., & Medina-Chicaiza, P. (2021). Estrategia metodológica basada en tecnologías de la información y comunicación en expresión oral del idioma inglés. *INNOVA Research Journal*, *6*(1), 111–128. https://doi.org/10.33890/innova.v6.n1.2021.1463

Moreno-León, J., & Robles, G. (2019). Code to learn: Where does it belong in the K-12 curriculum? *Journal of Information Technology Education: Research*, 18, 113-141. https://doi.org/10.28945/4268

Pérez, S., & Robles, G. (2021). Feedback practices in programming education: A systematic literature review. *IEEE Access*, *9*, 123456-123468. https://doi.org/10.1109/ACCESS.2021.123456

Román-González, M., Pérez-González, J.-C., & Jiménez-Fernández, C. (2015). Test de pensamiento computacional: Diseño y psicometría general [Computational Thinking Test: Design & General Psychometry]. https://doi.org/10.13140/RG.2.1.3056.5521

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2021). Demystifying computational thinking for education and learning. *Educational Research Review*, *33*, 100385. https://doi.org/10.1016/j.edurev.2021.100385

Tang, X., Yin, Y., & Lin, Q. (2022). Integrating computational thinking into STEM education: A systematic review. *Computers & Education*, *176*, 104382. https://doi.org/10.1016/j.compedu.2021.104382

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. https://doi.org/10.1145/1118178.1118215