

SSHE-Vol.4. N1. 011

Modelo pedagógico integrador de pensamiento computacional para la enseñanza y aprendizaje de la programación básica

Integrative pedagogical model of computational thinking for the teaching and learning of basic programming

Autores:

Tania Karina Ponce Torca
Universidad San Francisco Xavier de Chuquisaca
Sucre – Bolivia
ponce.tania@usfx.bo
<https://orcid.org/0009-0009-1158-1668>

Autor de correspondencia: Tania Karina Ponce Torca, ponce.tania@usfx.bo

Enviado: 11-02-2026

Revisado: 10-04-2026

Aceptado: 19-08-2026

Publicado: 21-08-2026



Cómo citar este artículo:

Ponce Torca, T. K. (2026). Modelo pedagógico integrador de pensamiento computacional para la enseñanza y aprendizaje de la programación básica. *Sage Sphere Higher Education*, 4(1), 1-22. <https://doi.org/10.63688/ss155s24>

© 2026; Los autores. Este es un artículo en acceso abierto, distribuido bajo los términos de una licencia Creative Commons (<https://creativecommons.org/licenses/by/4.0>) que permite el uso, distribución y reproducción en cualquier medio, siempre que la obra original sea correctamente citada.



RESUMEN

La enseñanza de la programación básica en educación superior exige atender procesos de razonamiento que anteceden a la escritura de código. El objetivo fue diseñar un Modelo Pedagógico Integrador de Pensamiento Computacional para fortalecer la enseñanza y el aprendizaje de Programación Básica en el Área Tecnológica de la Universidad San Francisco Xavier de Chuquisaca. Se empleó un diseño mixto convergente, descriptivo, propositivo y transversal, con integración de resultados cuantitativos, cualitativos y documentales. Participaron 289 estudiantes, de una población de 741, y ocho docentes. Se aplicó una prueba basada en tareas Bebras, entrevistas semiestructuradas y análisis de registros académicos de 2020-2025. El diagnóstico identificó dificultades en descomposición, reconocimiento de patrones, abstracción y diseño algorítmico, junto con factores pedagógicos, tecnológicos y emocionales. Los registros disponibles mostraron 132 abandonos de 161 estudiantes en 1/2020 y, en 2/2024, 22 aprobados frente a 20 reprobados. La triangulación sustentó un modelo que articula diagnóstico de entrada, pensamiento computacional, mediación pedagógica y tecnológica, programación, retroalimentación y evaluación formativa. Se concluye que la propuesta ofrece una estructura contextualizada para organizar la enseñanza, aunque su eficacia requiere una fase posterior de implementación y evaluación empírica.

Palabras clave: pensamiento computacional; programación básica; educación superior; modelo pedagógico; resolución de problemas.

ABSTRACT

Teaching basic programming in higher education requires attention to reasoning processes that precede code writing. This study aimed to design an Integrative Pedagogical Model of Computational Thinking to strengthen the teaching and learning of Basic Programming in the Technological Area of Universidad San Francisco Xavier de Chuquisaca. A convergent mixed-methods, descriptive, design-oriented, cross-sectional approach was used, integrating quantitative, qualitative, and documentary evidence. Participants were 289 students from a population of 741 and eight faculty members. Data were collected through a Bebras-based task assessment, semi-structured interviews, and analysis of academic records from 2020-2025. The diagnosis identified difficulties in decomposition, pattern recognition, abstraction, and algorithmic design, together with pedagogical, technological, and emotional factors. Available records showed 132 dropouts among 161 students in 1/2020 and, in 2/2024, 22 passes versus 20 failures. Triangulation informed a model integrating entry diagnosis, computational thinking, pedagogical and technological mediation, programming, feedback, and formative assessment. The proposal provides a contextualized structure for organizing programming instruction; however, its effectiveness must be established in a subsequent implementation and empirical evaluation phase.

Keywords: computational thinking; basic programming; higher education; pedagogical model; problem solving.



1. INTRODUCCIÓN

Aprender a programar suele presentarse como una experiencia eminentemente técnica; sin embargo, en el aula universitaria las dificultades no se limitan al aprendizaje de variables, operadores, condicionales o ciclos. Para muchos estudiantes, el reto comienza antes de escribir código: comprender una situación problemática, separar sus partes, reconocer regularidades, identificar la información relevante y convertirla en una secuencia ordenada de acciones.

Esta situación es especialmente relevante en los cursos iniciales de las carreras tecnológicas, donde convergen estudiantes con trayectorias educativas, experiencias digitales y conocimientos previos heterogéneos. Cuando esas diferencias no se reconocen pedagógicamente, quienes poseen experiencia previa pueden avanzar con mayor facilidad, mientras otros requieren apoyos adicionales y pueden experimentar frustración, inseguridad o discontinuidad académica. Desde el aprendizaje significativo, los conocimientos previos condicionan la forma en que se incorpora nueva información (Ausubel, 2000).

El problema, por tanto, no puede reducirse a establecer si el estudiante sabe o no sabe programar. Resulta necesario analizar cómo formula un problema, organiza información, selecciona lo esencial, construye una solución y revisa su estrategia cuando la primera alternativa no funciona. En este escenario adquiere relevancia el pensamiento computacional, entendido como un conjunto de procesos para formular problemas y representar soluciones de manera que puedan ser ejecutadas por un agente de procesamiento de información (Wing, 2006).

Entre sus componentes se reconocen con frecuencia la descomposición, la abstracción, el reconocimiento de patrones y el diseño algorítmico. Las revisiones de George-Reyes (2023), Cossío Acosta (2021) y Shute et al. (2017) muestran que estas capacidades se relacionan con la resolución estructurada de problemas y con la alfabetización digital, aunque también advierten que el pensamiento computacional es un constructo multidimensional que debe evaluarse con criterios explícitos.

La relación entre pensamiento computacional y programación es particularmente significativa en educación superior. Tikva y Tambouris (2021) identificaron, en una revisión sistemática, áreas de conocimiento, evaluación, estrategias de aprendizaje, herramientas y



factores contextuales que deben articularse para enseñar pensamiento computacional mediante programación. De forma complementaria, Rojas-López y García-Peñalvo (2020) mostraron la utilidad de diagnosticar estas capacidades en estudiantes de programación inicial, mientras Pérez (2021) destacó la importancia de conocer las percepciones y condiciones de partida de los universitarios.

La evaluación merece especial atención. Tang et al. (2020) evidenciaron que la investigación sobre pensamiento computacional presenta una gran diversidad de instrumentos y que la evidencia de validez y confiabilidad no siempre se reporta. En el ámbito universitario, Meza et al. (2024) aportaron evidencia específica sobre la validación de una prueba basada en tareas Bebras para estudiantes de primer año, lo que refuerza la necesidad de documentar con precisión la versión, puntuación y propiedades de cualquier adaptación empleada.

Desde esta perspectiva, enseñar programación significa también enseñar a pensar. El algoritmo puede entenderse como una representación del razonamiento; el código, como su materialización; la ejecución, como una oportunidad para comprobarlo; y el error, como información para revisar y mejorar la solución. Esta concepción se vincula con el construccionismo de Papert (1980), en el que el aprendizaje se fortalece al construir objetos significativos que permiten exteriorizar y revisar ideas.

La dimensión social del aprendizaje también resulta relevante. Vygotsky (1978) permite comprender la importancia de la mediación, el diálogo y la interacción. En programación, la explicación de procedimientos, la colaboración entre pares y la retroalimentación docente pueden hacer visible el razonamiento. La evidencia en educación superior muestra que la programación en parejas tiende a producir resultados favorables, aunque su efecto depende del diseño de tareas, de la orientación docente y de la organización de la interacción (Hawlitshchek et al., 2023).

En el Área Tecnológica de la Facultad de Ciencias y Tecnología de la Universidad San Francisco Xavier de Chuquisaca, la Programación Básica constituye un componente formativo central. Los registros institucionales revisados para este estudio muestran dificultades de aprobación, reprobación y abandono en diferentes periodos entre 2020 y 2025. Las entrevistas docentes, a su vez, señalan heterogeneidad en conocimientos previos, ritmos de aprendizaje, disponibilidad tecnológica, pensamiento lógico, práctica y manejo emocional del error.



Por ello, la investigación no se limitó a describir dificultades. Su propósito fue integrar las evidencias cuantitativas, cualitativas y documentales en una propuesta pedagógica común. El objetivo fue diseñar un Modelo Pedagógico Integrador de Pensamiento Computacional para fortalecer la enseñanza y el aprendizaje de la Programación Básica en el Área Tecnológica de la Universidad San Francisco Xavier de Chuquisaca. La pregunta de investigación fue: ¿cómo integrar el pensamiento computacional en un modelo pedagógico que responda a las necesidades cognitivas, pedagógicas, tecnológicas y contextuales identificadas en la Programación Básica?

2. METODOLOGÍA

Enfoque y diseño de investigación

La investigación se desarrolló con un enfoque mixto mediante un diseño convergente paralelo (QUAN + QUAL), de alcance descriptivo y propositivo y corte transversal. Esta clasificación se sustenta en que el componente cuantitativo y el cualitativo fueron recolectados durante la misma fase general del estudio, se analizaron inicialmente por separado y se integraron posteriormente con la evidencia documental para interpretar el problema y construir el modelo pedagógico. El componente cuantitativo examinó el desempeño en tareas asociadas con pensamiento computacional; el cualitativo exploró las experiencias y percepciones de los docentes; y el análisis documental aportó la trayectoria académica de la asignatura entre 2020 y 2025.

La integración se realizó en dos momentos. Primero, se construyó una matriz de convergencia en la que cada hallazgo fue clasificado como convergente, complementario o discrepante según su correspondencia entre la prueba de desempeño, las entrevistas y los registros académicos. Segundo, los hallazgos integrados se transformaron en decisiones de diseño: las dificultades cognitivas orientaron los componentes de pensamiento computacional; la heterogeneidad de entrada justificó el diagnóstico inicial; los problemas de acceso sustentaron la mediación tecnológica; la frustración y el error fundamentaron la retroalimentación formativa; y la fragmentación de estrategias docentes motivó un marco pedagógico común. De este modo, la triangulación no se utilizó solo como contraste, sino como mecanismo explícito para pasar del diagnóstico a la propuesta.

Contexto y participantes



La población estuvo constituida por 741 estudiantes matriculados en carreras del Área Tecnológica y 15 docentes vinculados con asignaturas relacionadas con Programación Básica. La muestra estuvo integrada por 289 estudiantes que participaron en la evaluación basada en tareas Bebras y ocho docentes seleccionados para entrevistas semiestructuradas. La selección estudiantil fue no probabilística por conveniencia y la selección docente fue intencional, atendiendo a su experiencia directa en la enseñanza de programación.

Aunque la selección no fue probabilística, se efectuó una comprobación de suficiencia numérica del tamaño estudiantil mediante la fórmula para población finita. Para $N = 741$, un nivel de confianza de 95 %, máxima variabilidad ($p = q = 0,50$) y un error aproximado de 4,5 %, el tamaño calculado es cercano a 289 participantes. Esta comprobación respalda la suficiencia del número de casos para el análisis descriptivo, pero no elimina las limitaciones de representatividad inherentes al muestreo por conveniencia ni autoriza generalizaciones probabilísticas a toda la población.

Para el componente estudiantil se consideraron los 289 estudiantes matriculados en carreras del Área Tecnológica que participaron voluntariamente en la aplicación de la prueba de desempeño. Para el componente cualitativo se incluyeron ocho docentes vinculados directamente con asignaturas relacionadas con Programación Básica, seleccionados de forma intencional por su experiencia en la enseñanza de programación. La delimitación de ambos grupos se mantuvo coherente con la población definida para el estudio y con el carácter no probabilístico del muestreo.

Técnicas e instrumentos

Para el componente cuantitativo se utilizó una prueba de desempeño basada en tareas Bebras, concebida para aproximarse a capacidades de pensamiento computacional mediante problemas que pueden resolverse sin depender exclusivamente del dominio de un lenguaje de programación. En este estudio, el análisis se organizó alrededor de cuatro procesos: descomposición, reconocimiento de patrones, abstracción y diseño algorítmico. La elección de este tipo de tareas es consistente con la tradición Bebras y con trabajos que han explorado su uso para evaluar pensamiento computacional; sin embargo, la literatura enfatiza que la evidencia psicométrica debe reportarse para cada adaptación y población (Bebras Community, 2025; Tang et al., 2020; Meza et al., 2024).



La prueba de desempeño se estructuró a partir de tareas del enfoque Bebras y se utilizó con finalidad diagnóstica para examinar cuatro procesos: descomposición, reconocimiento de patrones, abstracción y diseño algorítmico. La documentación conservada del estudio no permite identificar una versión estandarizada específica ni reconstruir con precisión el banco de ítems, la escala completa de puntuación o indicadores psicométricos propios de esta aplicación. En consecuencia, los resultados se interpretan de manera descriptiva y no se atribuyen a la prueba propiedades de validez o confiabilidad que no estén demostradas para esta muestra. Esta decisión es consistente con la recomendación de documentar explícitamente las propiedades de cada adaptación de tareas Bebras (Bebras Community, 2025; Tang et al., 2020; Meza et al., 2024).

En el componente cualitativo se aplicaron entrevistas semiestructuradas a ocho docentes. La guía se organizó en torno a conocimientos previos, pensamiento lógico, práctica, diversidad de niveles, condiciones tecnológicas, frustración, estrategias docentes y riesgo de abandono. Asimismo, se realizó análisis documental de registros académicos vinculados con la asignatura de programación durante 2020-2025.

Procedimiento y estrategia de análisis

La investigación se desarrolló en cinco fases. En la primera se revisaron antecedentes teóricos y registros académicos para delimitar el problema y las categorías iniciales. En la segunda se aplicó la prueba de desempeño a los estudiantes y se organizaron las respuestas mediante estadística descriptiva. En la tercera se realizaron las entrevistas semiestructuradas a los docentes. En la cuarta se integraron las tres fuentes mediante la matriz de convergencia y se establecieron las necesidades que debían ser atendidas por el modelo. En la quinta se diseñó la propuesta y se sometió a valoración por expertos.

Para el componente cuantitativo se empleó estadística descriptiva para organizar la información disponible. El análisis cualitativo siguió una codificación temática híbrida: los códigos iniciales se derivaron de los ejes de la guía de entrevista y se refinaron mediante lectura comparativa de las ocho entrevistas hasta agruparse en categorías axiales. Se conservó una matriz de trazabilidad código-categoría-docente y se aplicó comparación constante y triangulación con la prueba de desempeño y los registros académicos. No se estableció saturación teórica como criterio de cierre, debido a que la muestra intencional de ocho docentes fue definida a priori.



Consideraciones éticas

La investigación respetó los principios de participación voluntaria, confidencialidad, anonimización y uso académico de la información. Los docentes fueron identificados mediante códigos (D1-D8) y los resultados estudiantiles se analizaron de forma agregada. La documentación disponible del estudio no consigna un código de aprobación de comité de ética; por ello, no se atribuye una aprobación formal que no esté documentada.

3. RESULTADOS

Comportamiento académico histórico de la programación

El análisis documental de los registros académicos disponibles para 2020-2025 mostró un comportamiento irregular. Entre los valores efectivamente documentados, en 1/2020 se registraron 132 abandonos de 161 estudiantes, equivalente a 82,0 %. Para 2/2024 se consignaron 22 estudiantes aprobados y 20 reprobados. La síntesis se limita a estos datos verificables y evita inferir valores no conservados en los registros incorporados al manuscrito.

Tabla 1.

Indicadores académicos disponibles para el periodo 2020-2025

Periodo	Total matriculados	Aprobados	Reprobados	Abandonos	Indicador verificable
1/2020	161	—	—	132	Abandono = 82,0 %
2/2024	—	22	20	—	Aprobados y reprobados reportados

Nota. La tabla presenta únicamente los valores explícitamente documentados en los registros incorporados al estudio. El porcentaje de abandono de 1/2020 se calculó como 132/161 = 82,0 %. El guion indica que el valor correspondiente no fue reportado en la documentación utilizada.

Resultados de la prueba basada en tareas Bebras

La prueba se utilizó para describir el desempeño estudiantil en procesos vinculados con el pensamiento computacional. Los resultados documentados permiten identificar dificultades en descomposición, reconocimiento de patrones, abstracción y diseño algorítmico en la muestra de 289 estudiantes. Debido a que la matriz individual de puntuaciones y la escala completa de la aplicación no forman parte de la documentación conservada, no se estimaron



medias, desviaciones estándar ni porcentajes por dimensión. La interpretación se mantiene en un nivel descriptivo y se vincula con las necesidades pedagógicas identificadas.

Tabla 2.

Síntesis del desempeño documentado en los procesos evaluados mediante tareas Bebras

Dimensión	Evidencia documentada	Interpretación
Descomposición	Dificultad identificada en el diagnóstico	Se requiere fraccionar los problemas en partes manejables antes de programar.
Reconocimiento de patrones	Dificultad identificada en el diagnóstico	Es necesario fortalecer la identificación de regularidades y relaciones reutilizables.
Abstracción	Dificultad identificada en el diagnóstico	Se requiere distinguir la información relevante de los elementos secundarios del problema.
Diseño algorítmico	Dificultad identificada en el diagnóstico	Se necesita reforzar la organización secuencial y lógica de las soluciones.
Cobertura de la aplicación	289 estudiantes	La prueba se aplicó a la muestra estudiantil definida para el componente cuantitativo.

Nota. La tabla resume únicamente la evidencia disponible para los cuatro procesos analizados y la cobertura de la aplicación; no se incorporan puntuaciones o categorías que no puedan verificarse con la documentación del estudio.

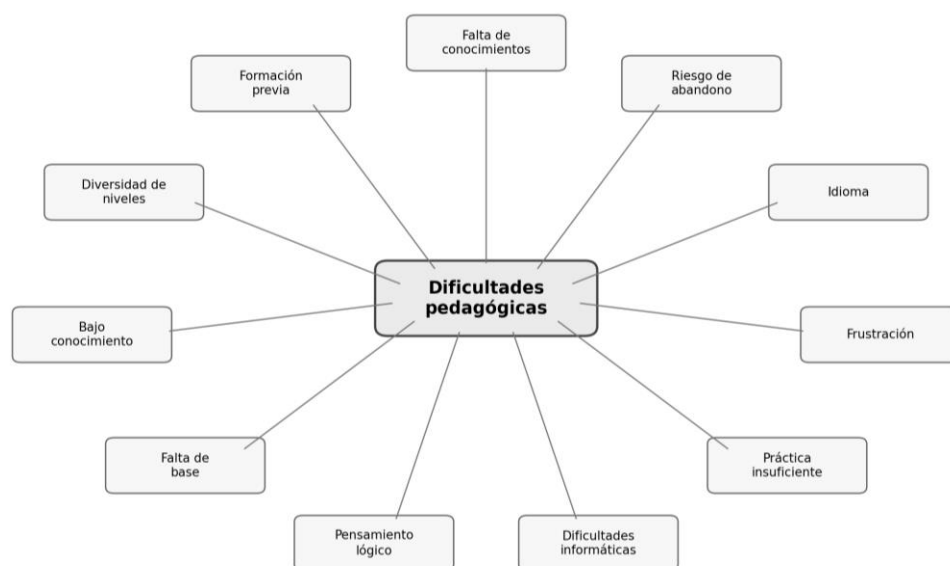
Dificultades pedagógicas identificadas por los docentes

El análisis de las ocho entrevistas produjo una categoría axial denominada dificultades pedagógicas. Los códigos asociados reflejaron heterogeneidad de conocimientos previos y formación, diversidad de niveles, dificultades de pensamiento lógico, práctica insuficiente, limitaciones informáticas, frustración, idioma y riesgo de abandono. La red de categorías muestra que estos elementos no se presentan de manera aislada, sino que interactúan durante el aprendizaje de la programación.

Figura 1.

Red de categorías asociadas con las dificultades pedagógicas identificadas en las entrevistas





Nota. Elaboración propia a partir de la codificación de las ocho entrevistas semiestructuradas. Los docentes describieron aulas en las que estudiantes con experiencia previa en programación comparten el proceso formativo con otros que todavía necesitan familiarizarse con procedimientos informáticos básicos. Esta diversidad obliga a ajustar ritmos, ejemplos y apoyos y puede generar sobrecarga cuando el estudiante debe aprender simultáneamente el uso de herramientas, conceptos abstractos y secuencias lógicas.

Tabla 3.

Trazabilidad cualitativa de categorías y evidencia de entrevista

Categoría	Códigos asociados	Interpretación
Condiciones de entrada	Formación previa; falta de base; bajo conocimiento; diversidad de niveles	Los estudiantes comienzan con experiencias, conocimientos y ritmos diferentes.
Procesos cognitivos	Pensamiento lógico; descomposición; abstracción; algoritmos	Las dificultades aparecen antes o durante la traducción del problema a una solución programable.
Práctica y acompañamiento	Práctica insuficiente; estrategias docentes; apoyo entre pares	La práctica guiada, el acompañamiento y la retroalimentación aparecen como necesidades recurrentes.



Categoría	Códigos asociados	Interpretación
Dimensión emocional	Frustración; comparación; riesgo de abandono	El error puede convertirse en inseguridad cuando no se aborda de manera formativa.
Condiciones tecnológicas	Dificultades informáticas; acceso y funcionamiento de recursos	Las limitaciones de acceso y funcionamiento pueden ampliar las diferencias entre estudiantes.

Nota. Elaboración propia a partir de la codificación temática de las ocho entrevistas semiestructuradas. Se presentan categorías, códigos e interpretaciones respaldadas por el análisis, sin incorporar citas textuales cuya transcripción literal no esté disponible en la documentación del estudio.

Dimensión emocional del aprendizaje

La frustración emergió como un elemento vinculado con las dificultades de aprendizaje, especialmente cuando los estudiantes comparan su avance con el de compañeros que poseen mayor experiencia. En programación, donde el error forma parte del proceso, una interpretación punitiva del fallo puede transformarse en inseguridad. Por esta razón, la propuesta resignifica el error como información para localizar, depurar y mejorar una solución.

Dimensión tecnológica

Las entrevistas también señalaron limitaciones relacionadas con la disponibilidad y el funcionamiento de recursos tecnológicos. Estas condiciones pueden ampliar las diferencias entre quienes disponen de equipos propios y quienes dependen de la infraestructura institucional. En consecuencia, la mediación tecnológica del modelo no se limita a recomendar plataformas o software, sino que exige considerar acceso, soporte y disponibilidad real.

Fragmentación de las prácticas docentes

Otro hallazgo fue la diversidad de contenidos, ritmos, estrategias y formas de acompañamiento entre docentes. Esta variedad refleja respuestas adaptativas a una problemática compleja; no obstante, cuando el apoyo depende exclusivamente de la iniciativa individual, los estudiantes pueden recibir oportunidades diferentes dentro de una misma asignatura. El modelo busca, por ello, ofrecer principios y componentes compartidos sin eliminar la autonomía docente.

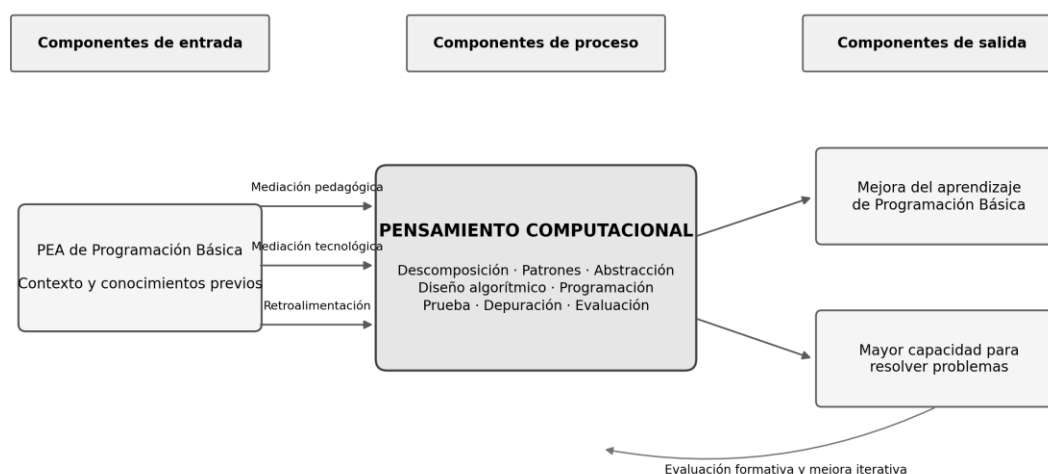


Modelo Pedagógico Integrador de Pensamiento Computacional

A partir de la integración de los resultados se diseñó el Modelo Pedagógico Integrador de Pensamiento Computacional para la enseñanza y el aprendizaje de Programación Básica. El pensamiento computacional constituye el eje articulador y se relaciona con el contexto y los conocimientos previos, la mediación pedagógica, la mediación tecnológica y la retroalimentación. Su lógica general sigue un ciclo de comprensión del problema, descomposición, reconocimiento de patrones, abstracción, diseño algorítmico, programación, prueba, depuración, evaluación y mejora.

Figura 2.

Modelo Pedagógico Integrador de Pensamiento Computacional para Programación Básica



Nota. Elaboración propia. El proceso es iterativo: la evaluación y la depuración permiten regresar a etapas anteriores para revisar y mejorar la solución.

Fundamentos pedagógico, epistemológico, tecnológico y práctico-contextual

El fundamento pedagógico concibe al estudiante como protagonista del aprendizaje y al docente como mediador que plantea desafíos, formula preguntas, ofrece apoyos y ajusta la enseñanza a necesidades de entrada. Esta perspectiva se relaciona con Vygotsky (1978), al reconocer el valor de la mediación social, y con Ausubel (2000), al considerar los conocimientos previos como punto de partida.

Desde el punto de vista epistemológico, el conocimiento se construye progresivamente mediante experiencia, reflexión y comprobación. En programación, un algoritmo o un



programa no representa solo un producto final, sino también el camino seguido para construir una solución. Este planteamiento es coherente con el construccionismo de Papert (1980).

El fundamento tecnológico entiende los entornos de programación, plataformas, simuladores y recursos digitales como medios al servicio del aprendizaje. Su selección debe articular contenido, pedagogía y tecnología, en línea con el marco TPACK de Mishra y Koehler (2006). La inteligencia artificial generativa puede incorporarse como apoyo para explicar conceptos, revisar errores o proponer casos de prueba, siempre que el estudiante conserve la responsabilidad de comprender, verificar y justificar el resultado (Bringula, 2024; Cambaz & Zhang, 2024).

El fundamento práctico-contextual parte de la realidad de la USFX y reconoce la diversidad de conocimientos previos, ritmos, recursos y experiencias. Por ello, el modelo se plantea como una estructura flexible y contextualizada, no como una secuencia rígida o una receta única.

Componentes del modelo

Componente 1. Diagnóstico de entrada. Antes de iniciar la enseñanza se identifican conocimientos previos, experiencia tecnológica y habilidades iniciales de razonamiento. El diagnóstico tiene una finalidad orientadora: permite decidir apoyos, nivel de complejidad y rutas de acompañamiento, sin utilizarse para excluir o etiquetar al estudiante.

Componente 2. Pensamiento computacional y proceso de resolución. Se trabajan de manera explícita la descomposición, el reconocimiento de patrones, la abstracción y el diseño algorítmico, seguidos de programación, prueba, depuración y evaluación. Las actividades deben hacer visible el proceso de razonamiento, no limitarse a la corrección sintáctica del código.

Componente 3. Mediación pedagógica. El docente acompaña el razonamiento mediante preguntas, ejemplos, andamiajes, actividades diferenciadas, trabajo entre pares y retroalimentación. La programación colaborativa se incorpora con roles y criterios de interacción para evitar que la colaboración se reduzca a repartir tareas.

Componente 4. Mediación tecnológica. Las herramientas digitales se seleccionan por su contribución a experimentar, ejecutar, visualizar, comparar y depurar soluciones. Su empleo depende de condiciones reales de acceso y soporte institucional.



Componente 5. Evaluación formativa y mejora. La evaluación considera tanto el producto como el proceso: comprensión del problema, representación algorítmica, justificación de decisiones, funcionamiento del código, depuración y capacidad de mejora. El error se utiliza como evidencia para orientar la retroalimentación.

Componente 6. Resultados formativos esperados. El modelo busca mejorar el aprendizaje de Programación Básica y fortalecer la capacidad para resolver problemas con mayor autonomía, razonamiento lógico, creatividad y capacidad de explicar y revisar soluciones.

Relaciones entre los componentes y validación de la propuesta

El modelo se organiza mediante tres relaciones centrales. La primera vincula fundamentos, pensamiento computacional y estrategias didácticas: los fundamentos establecen la concepción educativa, el pensamiento computacional define las capacidades que se pretenden desarrollar y las estrategias convierten esas capacidades en experiencias concretas. La segunda articula pensamiento computacional, programación y tecnología: el estudiante comprende y representa el problema, construye un algoritmo, lo convierte en código, ejecuta la solución y analiza sus resultados. La tercera integra evaluación, retroalimentación y mejora para convertir el error en una oportunidad de aprendizaje.

La propuesta fue sometida a valoración por expertos mediante una ficha de diez elementos relacionados con pertinencia, fundamentación, estructura, coherencia y aplicabilidad. El registro disponible consigna una valoración global favorable de la propuesta. Como no se dispone de puntuaciones desagregadas ni del perfil individual de los evaluadores, esta fase se informa como una valoración cualitativa de pertinencia y coherencia, sin asignar índices psicométricos de validez de contenido.

Tabla 4.

Síntesis documentada de la valoración por expertos

Aspecto	Información documentada
Instrumento	Ficha de valoración por expertos con 10 elementos.
Aspectos valorados	Pertinencia, fundamentación, estructura, coherencia y aplicabilidad.
Resultado documentado	Valoración global favorable de la propuesta.
Interpretación	La valoración respalda la pertinencia y la coherencia interna percibidas por los especialistas.



Aspecto	Información documentada
Alcance	La valoración por expertos no constituye evidencia de eficacia del modelo sobre el aprendizaje.

Nota. La valoración se informa en el nivel de detalle respaldado por la documentación disponible y no se asignan puntuaciones, porcentajes ni coeficientes no verificables.

La valoración favorable reportada en el manuscrito respalda la pertinencia y coherencia interna percibidas por los especialistas, pero no constituye evidencia de eficacia del modelo. Esa eficacia deberá establecerse mediante una fase de implementación con indicadores de aprendizaje definidos a priori.

4. DISCUSIÓN

Los resultados permiten comprender que las dificultades de la Programación Básica no se explican únicamente por el desconocimiento de un lenguaje. En el contexto analizado convergen razonamiento, conocimientos previos, prácticas docentes, recursos tecnológicos y experiencias emocionales. Esta interpretación coincide con la literatura que concibe el pensamiento computacional como un constructo multidimensional y que reclama evaluaciones capaces de distinguir procesos de resolución y no solo productos finales (Shute et al., 2017; Tang et al., 2020).

La heterogeneidad de entrada observada en las entrevistas justifica incorporar un diagnóstico inicial con finalidad pedagógica. Rojas-López y García-Peñalvo (2020) destacan la pertinencia de evaluar pensamiento computacional en estudiantes que inician programación, y Pérez (2021) muestra que las percepciones y experiencias previas constituyen un marco relevante para interpretar el aprendizaje universitario. El aporte del modelo consiste en convertir el diagnóstico en decisiones de apoyo y progresión, evitando que se utilice como mecanismo de selección.

Las dificultades identificadas en descomposición, reconocimiento de patrones, abstracción y diseño algorítmico son consistentes con las dimensiones recurrentes en la literatura (Belmar, 2022; George-Reyes, 2023). En este estudio, la evidencia disponible permite sostener la presencia de dificultades en los cuatro procesos, aunque no cuantificar su magnitud mediante medias o distribuciones por dimensión. Esta cautela es especialmente importante porque Tang et al. (2020) documentaron heterogeneidad en los instrumentos de pensamiento computacional y una comunicación insuficiente de evidencias psicométricas.



La dimensión emocional agrega un componente pedagógico relevante. En programación, la depuración hace del error una parte esperable del proceso, por lo que una enseñanza centrada solo en respuestas correctas puede ocultar información valiosa sobre el razonamiento. El modelo propone que la retroalimentación analice qué parte de la solución falló, por qué falló y cómo puede mejorarse. Esta orientación desplaza la evaluación desde un criterio binario de funcionamiento hacia una lectura formativa del proceso.

Las estrategias activas deben interpretarse con cautela. Mellado y Cubillos (2024) encontraron efectos favorables de la gamificación en una actividad universitaria de programación, pero también subrayan que el resultado depende del diseño instruccional. Por ello, el modelo no incorpora la gamificación como un recurso automático, sino como una opción subordinada al objetivo cognitivo, al tipo de problema y a la retroalimentación que recibe el estudiante.

La programación colaborativa representa otra vía de apoyo. La revisión de Hawlitschek et al. (2023), basada en estudios empíricos de educación superior, muestra resultados mayoritariamente positivos de la programación en parejas, aunque persisten preguntas sobre guía, conformación de parejas y diseño de tareas. Esto coincide con la lógica del modelo: colaborar debe implicar explicar decisiones, comparar estrategias y revisar soluciones, no simplemente dividir el trabajo.

La mediación tecnológica también requiere una lectura contextual. El marco TPACK plantea que la integración efectiva depende de la relación entre conocimiento disciplinar, pedagógico y tecnológico (Mishra & Koehler, 2006). En el caso estudiado, las limitaciones de infraestructura señaladas por los docentes impiden asumir que una herramienta digital, por sí misma, producirá mejoras. El componente tecnológico del modelo adquiere sentido únicamente cuando existe acceso y cuando la herramienta aporta a comprender, ejecutar, visualizar o depurar.

La inteligencia artificial generativa se incorporó en la actualización curricular de 2026 y no como categoría empírica original de las entrevistas. La literatura reciente muestra posibilidades para generar ejemplos, explicar conceptos, apoyar la depuración y ofrecer tutoría, pero también riesgos de respuestas incorrectas, dependencia y uso acrítico. Bringula (2024) advierte que el código generado requiere verificación y acompañamiento docente, mientras Cambaz y Zhang (2024), en una revisión sistemática, identifican



beneficios potenciales junto con riesgos para estudiantes novatos. En consecuencia, el modelo propone una integración limitada: la IA puede apoyar la explicación y la revisión, pero el estudiante debe comprender, probar, justificar y asumir autoría de la solución.

La triangulación de resultados representa un aporte metodológico del estudio porque vincula diagnóstico cognitivo, experiencia docente y trayectoria académica con componentes concretos de diseño. Tikva y Tambouris (2021) muestran que la enseñanza del pensamiento computacional en educación superior requiere articular conocimiento, evaluación, estrategias, herramientas y factores contextuales. El modelo propuesto recoge esas áreas en una estructura única y las adapta a las necesidades de Programación Básica en la USFX.

Finalmente, la valoración por expertos debe interpretarse como evidencia de validez de contenido o de pertinencia percibida, según el procedimiento que realmente se haya utilizado, y no como prueba de eficacia. La efectividad del modelo solo podrá establecerse después de implementarlo y comparar cambios en desempeño, permanencia y procesos de aprendizaje. Una etapa posterior podría emplear un diseño cuasiexperimental con pretest y posttest y conservar un componente cualitativo que permita comprender cómo estudiantes y docentes experimentan la intervención.

5. CONCLUSIÓN

Las dificultades de Programación Básica identificadas en el estudio son multidimensionales: incluyen procesos de pensamiento computacional, conocimientos previos, diversidad de niveles, condiciones tecnológicas, prácticas docentes y experiencias emocionales. Por tanto, una respuesta pedagógica centrada únicamente en aumentar ejercicios de sintaxis resulta insuficiente.

La evidencia obtenida mediante la prueba basada en tareas Bebras permitió identificar dificultades en descomposición, reconocimiento de patrones, abstracción y diseño algorítmico. Estos procesos deben trabajarse de forma explícita antes y durante la producción de código, mediante actividades que hagan visible el razonamiento y permitan revisar progresivamente las estrategias de solución.

La evidencia cualitativa muestra que el aula es heterogénea y que los apoyos docentes deben responder a distintos puntos de partida. El diagnóstico de entrada adquiere así una función



orientadora, mientras la retroalimentación y la depuración convierten el error en información para aprender.

El Modelo Pedagógico Integrador articula diagnóstico, pensamiento computacional, mediación pedagógica y tecnológica, programación, trabajo colaborativo, retroalimentación y evaluación formativa. Su principal contribución es organizar estos elementos dentro de una lógica común y contextualizada para la Programación Básica.

La valoración por expertos respaldó, en términos cualitativos, la pertinencia y la coherencia interna de la propuesta. Esta valoración constituye evidencia de aceptación de su estructura y aplicabilidad percibida, pero no demuestra efectividad sobre el aprendizaje, la cual deberá examinarse en una fase posterior de implementación.

La eficacia del modelo requiere una fase posterior de implementación y evaluación empírica. Esa etapa deberá medir cambios en pensamiento computacional, desempeño en programación y permanencia, y podría complementarse con evidencia cualitativa sobre la experiencia de estudiantes y docentes.

En términos pedagógicos, aprender a programar significa aprender a comprender, representar, construir, probar, explicar y mejorar soluciones. El valor del modelo radica en convertir el pensamiento computacional en eje organizador de esa experiencia, sin sustituir el razonamiento del estudiante por herramientas o procedimientos automáticos.

REFERENCIAS BIBLIOGRÁFICAS

Ausubel, D. P. (2000). The acquisition and retention of knowledge: A cognitive view. Kluwer Academic Publishers. <https://doi.org/10.1007/978-94-015-9454-7>

Bebras Community. (2025). Bebras international challenge on informatics and computational thinking. <https://www.bebas.org/>

Belmar, H. (2022). Review on the teaching of programming and computational thinking in the world. *Frontiers in Computer Science*, 4, 997222. <https://doi.org/10.3389/fcomp.2022.997222>

Bringula, R. (2024). ChatGPT in a programming course: Benefits and limitations. *Frontiers in Education*, 9, 1248705. <https://doi.org/10.3389/educ.2024.1248705>



- Cambaz, D., & Zhang, X. (2024). Use of AI-driven code generation models in teaching and learning programming: A systematic literature review. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024), Vol. 1 (pp. 172-178). Association for Computing Machinery. <https://doi.org/10.1145/3626252.3630958>
- Cossío Acosta, P. M. (2021). Pensamiento computacional: Habilidades asociadas y recursos didácticos. Una revisión sistemática. *Innovaciones Educativas*, 23(especial), 178-189. <https://doi.org/10.22458/ie.v23iespecial.3693>
- Dong, W., Li, Y., Sun, L., & Liu, Y. (2024). Developing pre-service teachers' computational thinking: A systematic literature review. *International Journal of Technology and Design Education*, 34(1), 191-227. <https://doi.org/10.1007/s10798-023-09811-3>
- George-Reyes, C. E. (2023). Imbricación del pensamiento computacional y la alfabetización digital en la educación. Modelación a partir de una revisión sistemática de la literatura. *Revista Española de Documentación Científica*, 46(1), e345. <https://doi.org/10.3989/redc.2023.1.1922>
- Hawlitshchek, A., Berndt, S., & Schulz, S. (2023). Empirical research on pair programming in higher education: A literature review. *Computer Science Education*, 33(3), 400-428. <https://doi.org/10.1080/08993408.2022.2039504>
- Mellado, R., & Cubillos, C. (2024). Gamification improves learning: Experience in a training activity of computer programming in higher education. *Journal of Computer Assisted Learning*, 40(4), 1959-1973. <https://doi.org/10.1111/jcal.13000>
- Meza, F., Vásquez, A., & San Martín, D. (2024). Validation of a Bebras-based test to assess computational thinking abilities in first-year college students. In Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE 2024), Vol. 2 (pp. 1750-1751). Association for Computing Machinery. <https://doi.org/10.1145/3626253.3635523>
- Mishra, P., & Koehler, M. J. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *Teachers College Record*, 108(6), 1017-1054. <https://doi.org/10.1111/j.1467-9620.2006.00684.x>



- Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic Books.
- Pérez, J. (2021). Percepción de estudiantes universitarios sobre el pensamiento computacional. *REDU. Revista de Docencia Universitaria*, 19(1), 111-127. <https://doi.org/10.4995/redu.2021.15491>
- Rojas-López, A., & García-Peñalvo, F. J. (2020). Evaluación del pensamiento computacional para el aprendizaje de programación de computadoras en educación superior. *Revista de Educación a Distancia*, 20(63). <https://doi.org/10.6018/red.409991>
- Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, 22, 142-158. <https://doi.org/10.1016/j.edurev.2017.09.003>
- Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers & Education*, 148, 103798. <https://doi.org/10.1016/j.compedu.2019.103798>
- Tikva, C., & Tambouris, E. (2021). A systematic mapping study on teaching and learning computational thinking through programming in higher education. *Thinking Skills and Creativity*, 41, 100849. <https://doi.org/10.1016/j.tsc.2021.100849>
- Vygotsky, L. S. (1978). *Mind in society: The development of higher psychological processes*. Harvard University Press.
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35. <https://doi.org/10.1145/1118178.1118215>

CONFLICTO DE INTERESES

No existe.

FINANCIAMIENTO

Ninguno.

CONTRIBUCIÓN DE AUTORÍA



Conceptualización: Tania Karina Ponce Torca.

Metodología: Tania Karina Ponce Torca.

Investigación: Tania Karina Ponce Torca.

Redacción – borrador original: Tania Karina Ponce Torca.

Redacción – revisión y edición: Tania Karina Ponce Torca.

