Revista Colombiana de Computación
Vol. 25, No. 1. January – June 2024, pp. 1-18
e-ISSN: 2539-2115, https://doi.org/10.29375/25392115.4463

REVISTA COLOMBIANA DE COMPUTACIÓN

Programar en la universidad.

Cuadernillo de apoyo cognitivo para el análisis de los procesos

Learning to code in college.

Cognitive support workbook for process analysis

Verónica Sofía D'Angelo[1]

(Recibido: 19 mayo 2022; aceptado: 29 mayo 2024; Publicado en Internet: 30 junio 2024)

Resumen. Este estudio presenta una investigación exploratoria centrada en analizar las primeras impresiones de estudiantes universitarios respecto a un cuadernillo de apoyo cognitivo diseñado para mejorar el proceso de aprendizaje en el contexto de una asignatura de algorítmica básica. El material en cuestión se fundamenta en principios de psicología del aprendizaje y psicología de la programación, con el propósito de abordar los desafíos planteados en la introducción del estudio. Los participantes, quienes están cursando su primer año en una carrera de ciencias de la computación, completaron dos cuestionarios destinados a evaluar su percepción del proceso de aprendizaje y sus opiniones sobre la utilidad del material proporcionado. Los resultados obtenidos sugieren la necesidad de profundizar en el contenido del material de estudio, especialmente en lo referente al análisis de procesos y estructuras de datos. Además, se plantea la posibilidad de ampliar el alcance de este estudio mediante la realización de experimentos controlados que evalúen el impacto del material en el rendimiento y el aprendizaje efectivo de programación.

Palabras clave: algoritmos y estructuras de datos, ciencias de la computación, psicología cognitiva, diagramas Nassi-Shneiderman, programación estructurada.

Abstract. This study presents an exploratory investigation focused on analyzing the initial impressions of university students regarding a cognitive support workbook designed to enhance the learning process in the context of a basic algorithmics course. The material in question is based on principles of learning psychology and programming psychology, aiming to address the challenges outlined in the study's introduction. The participants, who are in their first year of a computer science degree, completed two questionnaires aimed at assessing their perception of the learning process and their opinions on the usefulness of the provided material. The results suggest the need to delve deeper into the content of the study material, especially regarding the analysis of processes and data structures. Furthermore, there is a proposal to expand the scope of this study by conducting controlled experiments to evaluate the impact of the material on performance and effective programming learning.

Keywords: algorithms and data structures, computer science, cognitive psychology, Nassi-Shneiderman diagrams, structured programming

1 Introducción

Si tuviéramos que caracterizar a los humanos como seres pensantes basándonos en las últimas investigaciones en psicología del razonamiento, podría decirse que no son esencialmente deductivos sino más bien inductivos o abductivos. La lógica espontánea depende en gran medida del contexto (Cheng & Holyoak, 1985; Cosmides, 1989; Elqayam & Over, 2013; Gigerenzer & Hug, 1992; Johnson-Laird y otros, 1972; Johnson-Laird & Byrne, 1991; Mercier & Sperber, 2017; Over, 2009). La deducción cobra protagonismo al final de una etapa formativa o laboral, cuando un conjunto importante de categorías generales y reglas adquiridas comienzan a ser aplicadas. Mientras tanto, el camino hacia la abstracción se inicia en el "piso" del contexto e implica realizar cierto esfuerzo de "ascenso" hacia una posición más abstracta o deductiva. Pero este proceso no es general: existen tantas "escaleras" como disciplinas.

Estudios clásicos sobre expertos y novatos han mostrado que los recién llegados a un área de conocimiento no perciben las reglas abstractas de la misma manera que los expertos (Chi y otros,

Categorization and Representation of Physics Problems by Experts and Novices, 1981). Los estudiantes suelen detenerse en los detalles concretos del caso mientras que los expertos captan rápidamente los rasgos funcionales de una situación. ¿Cómo saber en qué peldaño de esa escalera se posiciona cada estudiante respecto de cada área de conocimiento, e incluso respecto de cada concepto particular? Esta pregunta atañe en especial al conocimiento en ciencias de la computación (CC) para el cual, la capacidad de abstracción se considera crítica (Armoni, 2013; Denning, Systems abstractions, 2022; Hazzan & Kramer, The role of abstraction in software engineering, 2008; Kramer, 2007).

1.1 Psicología y Programación

Las asignaturas relacionadas con el aprendizaje de programación en los primeros años de carreras universitarias en CC han tenido tradicionalmente un desempeño académico bajo, un alto nivel de deserción y un rendimiento general considerado pobre. A nivel internacional, la programación de ordenadores es considerada una habilidad de difícil adquisición (Bosse & Gerosa, 2017; Jenkins, 2002). Los factores que afectan su aprendizaje han sido estudiados desde fines de los años 1970 por la psicología de la programación -un área interdisciplinar cuyo objeto de estudio es la investigación sobre la cognición humana en la tarea de programar (Bell, 1995; Hoc y otros, 2014; Sajaniemi, 2008; Soloway & Spohrer, 2013; Weinberg, 1971). Desde los inicios de este movimiento, ya se contaba con cierto desarrollo en la psicología y experiencia en programación como para advertir que los métodos utilizados para programar no sólo deben ser evaluados por su capacidad computacional sino también por sus efectos cognitivos. Sin embargo, los estudios de expertos y novatos antes mencionados fueron posteriores al nacimiento de la programación estructurada e ignorados por la corriente "racionalista" que lideró los métodos de enseñanza. A partir de dichos estudios en varias disciplinas (Chi M. , 1992) se aceptó que los expertos en un área categorizan la información de un modo más abstracto que los recién iniciados (Chi y otros, Categorization and Representation of Physics Problems by Experts and Novices, 1981) y el aprendizaje o el desarrollo del conocimiento dejó de considerarse dependiente únicamente de la edad del sujeto: a partir de ese momento se consideró también su nivel de experiencia en un campo específico. Esto ha permitido comprender por qué los estudiantes que se adentran en un área temática no pueden captar rápidamente los aspectos generales de una situación de la misma manera que lo hacen los expertos (docentes), ni tienden a utilizar un enfoque deductivo ("de arriba hacia abajo") para resolver problemas. Sin embargo, en esta época, la programación estructurada ya era un paradigma consolidado y dominante que, por el contrario, abogaba por el desarrollo deductivo (de arriba hacia abajo) de los programas.

En Latinoamérica, durante el surgimiento de las primeras carreras universitarias de informática (Aguirre & Carnota, 2009), surgió el concepto de "programación científica" que se diferenciaba de la programación amateur (realizada por aficionados) por poseer un método y una serie de principios. Esta programación científica se basaba en el paradigma estructurado (también denominado top-down, o de refinamientos sucesivos) esencialmente deductivo (Dijkstra E. W., 1968; 1970; 1976; Wirth, Program development by stepwise refinement, 1971; 1973). Para programar con el método descendente se sugería a los programadores partir de una especificación general de módulos (cajas negras), luego dividir cada bloque en nuevas partes hasta llegar a los fragmentos elementales de código que podía ser interpretado por un compilador. Si bien esto representaba un avance respecto de la forma de trabajo monolítico (no modular), pudo observarse que los recién iniciados no tenían un nivel de experiencia que les permitiera identificar de manera efectiva los bloques generales más abstractos al principio del proceso de programación para ir refinando sus soluciones hasta llegar al código (Hoc, 2014). Esta preferencia de los recién iniciados por manipular al principio la información más concreta "de nivel bajo" fue explorada por algunos investigadores (Hazzan, 1999; 2003; 2008; Hazzan & Zazkis, 2005).

Dos falsas creencias resultaron muy perjudiciales para la enseñanza de programación estructurada a principiantes: primero, que la programación estructurada necesariamente implica un enfoque descendente, y segundo, que para lograr programas estructurados, los medios deben ser estructurados en sí mismos (Denning, 1975). La confusión reside en "no distinguir el producto del proceso que lo creó" (pág. 214). El mismo Wirth reconoció con posterioridad a sus escritos clásicos que no debe inferirse que "la concepción real del programa procede de una manera tan bien organizada, directa y top-down" (Wirth, 1974, pág. 251).

Desde los primeros estudios experimentales en psicología de la programación se demostró que a los programadores novatos les resulta difícil el método descendente y antes de llegar a captar la estructura general abstracta de un procedimiento o una estructura de control, deben poder explicarse a sí mismos el proceso paso a paso (Hoc, 1983).

Si bien la perspectiva de trabajar con pequeños bloques de código parecía ser más viable desde el paradigma de orientación a objetos (POO), ya que los procesos (métodos) son más específicos y se organizan alrededor de clases (Booch, 1986; Cox, 1986), desde el punto de vista de la enseñanza a novatos, una interpretación ingenua de dicho paradigma tampoco beneficiaba a los aprendices. El paradigma de orientación a objetos es más complejo que el estructurado. Los fragmentos de código, para ser luego utilizados (o reutilizados) por un sistema de mayor envergadura, deben ser modulares y totalmente funcionales. Esto implica tener cierta visión de la evolución y la complejidad de los sistemas, de nociones como la herencia y la encapsulación que no son triviales. Las dificultades se manifiestan en problemas puntuales de aprendizaje (Biju, 2013; Gutiérrez y otros, 2022; Saidova, 2022), ante los cuales se han intentado diversas estrategias, por ejemplo, la gamificación (Jusas y otros, 2022), o el uso de lenguajes multiparadigma como Python que abren la posibilidad de trabajar "sin reglas". En este momento, acaso la enseñanza de programación se haya dificultado -más que simplificado- por el exceso de información que obliga a la toma de decisiones: la discusión sobre qué paradigmas adoptar al inicio (programación estructurada, programación funcional, POO, métodos ágiles), si los lenguajes de bloques visuales facilitan la adquisición de conceptos o son más claros los lenguajes basados en texto, si conviene realizar las pruebas de software desde el principio, entre otras.

Ante este panorama cabe preguntarse si es conveniente seguir abordando la educación en informática desde una "pedagogía tecnológica" que confunde el pensamiento con el cómputo o si se adopta una perspectiva disciplinar profunda que considere sus orígenes y su evolución, ya que todas las disciplinas serias tienen historia (Denning & Tedre, 2021). Desde esta segunda posición, puede advertirse que, con excepción de algunos trabajos, (da Rosa S. , 2015; da Rosa y otros, 2020; da Rosa & Gómez, 2020), la psicología de la programación ha tenido poca participación en la didáctica latinoamericana, y posiblemente eso explique por qué entre tantas disquisiciones tecnológicas de forma, quedó en segundo plano el análisis del proceso como contenido básico. El proceso mismo de programación, tan implícito para un experto, es harto difícil para un ingresante. Implica un desafío de carga cognitiva (Sweller, 2010; 2016) porque deben manipularse mentalmente varios elementos simultáneamente, todos recién adquiridos: entrada y salida de datos, gestión de las variables en memoria, estructuras de datos, estructuras de control, y estructura del proceso que se está escribiendo en respuesta a un problema que se acaba de comprender.

La currícula universitaria mantiene el objetivo inicial de que los alumnos aprendan las estructuras de datos básicas y los procesos elementales de algoritmia sin ataduras a un paradigma específico ni a un lenguaje de programación. Por ello se incluye una asignatura dedicada a algoritmos y estructuras de datos (o designaciones similares), con una orientación estructurada con miras a ser aplicada posteriormente bajo cualquier paradigma.

Si bien las cátedras suelen adherir al enfoque top-down, algunas investigaciones proponen que los ingresantes debieran trabajar sobre pequeños bloques de funcionalidad para ir ascendiendo gradualmente hacia funciones más generales, por ejemplo, (Caspersen & Kölling, 2009). Con este método, se mantiene la estructuración de datos y procesos, aunque no se requiera que el estudiante realice un diseño descendente desde el principio. Este enfoque se volverá más accesible para ellos después de haber comprendido a fondo los procesos básicos.

1.2 Diseño de un material complementario para cátedras con orientación estructurada

Además de los problemas planteados y de la situación actual en la universidad, aquellos estudiantes que se ven obligados a usar un enfoque top-down sin haber adquirido la comprensión pormenorizada de los procesos más básicos, necesitan apoyo para encarar varios frentes que se describen a continuación.

Por un lado, el programador es, ante todo, alguien que resuelve un problema. Pero las personas no suelen ser conscientes de sus propios procesos cognitivos mientras resuelven problemas, por lo que suponen que las soluciones, cuando surgen, lo hacen espontáneamente, por una especie de insight que en realidad no es tal (Simon, 1986). Los alumnos de menor rendimiento, al desconocer la existencia de esos procesos, no tienen oportunidad de producir cambios o mejoras en ellos. Se necesita estimular la toma de consciencia del propio proceso metacognitivo (Prather y otros, 2018; 2020). Una vía prometedora es que la psicología de la programación retome la temática de los modelos mentales cuyas investigaciones han disminuido a lo largo de los años, a pesar de que se reconoce su importancia para el aprendizaje y el desempeño de los programadores (Heinonen y otros, 2022).

Por otro lado, los niveles de abstracción en CC son numerosos. El más elevado es el nivel del problema, el más bajo es la escritura del código, entre ellos hay niveles intermedios como las estructuras de control

de proceso y de datos, los procedimientos y funciones. El estudiante de programación debe "subir" y "bajar" constantemente entre niveles de abstracción. Cuando asocia un caso con una categoría abstracta de proceso o una función, asciende en el nivel de abstracción. Cuando conoce una operación genérica y la aplica a un caso, desciende. Hazzan (1999; 2003; Hazzan & Zazkis, 2005) observó que los alumnos tienden a reducir el nivel de abstracción y a quedarse más tiempo en el nivel de los componentes de código, porque tienen dificultades para trabajar en los niveles más elevados (que él denomina nivel del problema). A medida que los programadores adquieren experiencia, se espera que el nivel de abstracción en un dominio se eleve y que haya una mayor preferencia por la funcionalidad. Pero esto no puede forzarse y no puede saberse de antemano en qué nivel opera cada alumno en el momento de su ingreso a carrera.

Las explicaciones ofrecidas por los profesores durante la resolución de problemas en clase abordan tanto aspectos concretos relacionados con la redacción de instrucciones como cuestiones abstractas sobre la división del problema en bloques. Idealmente, estas explicaciones deberían estar diseñadas para ofrecer múltiples enfoques, favoreciendo así a los estudiantes que puedan requerir diferentes métodos de comprensión. Sin embargo, las explicaciones orales suelen ser extensas, lo que sobrecarga la memoria de trabajo del estudiante y dificulta la asimilación completa de la información, abarcando tanto generalidades como casos específicos. Además, los profesores, en su mayoría programadores con experiencia, tienden a omitir detalles que podrían resultar cruciales para un estudiante novato en la materia.

Otro tipo de explicaciones fructíferas son las auto-explicaciones que se dan los estudiantes al tratar de describir y comprender simultáneamente un proceso. Varios estudios muestran que aquellos alumnos que invierten más tiempo en explicarse a sí mismos aquello que intentan comprender son más eficaces en la resolución de problemas y en la adquisición de nuevos conceptos (Aureliano y otros, 2016; Chi y otros, 1989; Kastens & Liben, 2007; Bai y otros, 2022; Caspersen, 2023; Izu y otros, 2019; Margulieux & Catrambone, 2021; Nathan y otros, 1994; Risha y otros, 2021). Así mismo, otros estudios muestran que esta habilidad metacognitiva para razonar sobre el programa puede realizarse mediante la escritura reflexiva (Alrashidi y otros, 2020; Alt & Raichel, 2020; Anderson & Fincham, 2014; Ch´ng, 2018; Zarestky y otros, 2022).

Con respecto a los apuntes de estudio, no todos los textos redactados por los profesores, o los apuntes que toman los mismos estudiantes, cumplen con principios instruccionales para optimizar su lectura, por lo que resultan confusos. Por ejemplo, cuando se presenta un esquema visual con explicaciones anexadas, se debe tener especial cuidado en la representación de las relaciones parte-todo: los nombres y las descripciones de las partes de un todo deben mostrarse encima de la parte, no alejados (abajo o en otra página) para no perder la percepción general, y el orden de ubicación de dichas partes (Badali y otros, 2020; Merrill, 2002; 2012; 2018; Yorganci, 2020).

En relación con la sobrecarga de información en la memoria del aprendiz, para disminuir la carga cognitiva del proceso de programación, se han implementado diversos abordajes. Por un lado, presentar ejercicios resueltos en vez de pedir la solución a los estudiantes desde el principio. Según la teoría de la carga cognitiva (Sweller, 1988; 1994), y sus aplicaciones al campo de la enseñanza en ciencias de la computación (Duran y otros, 2022; Sweller, 2020). Presentar el análisis de ejercicios resueltos en una primera etapa disminuye la carga y facilita la adquisición de conceptos (Gupta & Zheng, 2020; Knorr, 2020; Lange y otros, 2023; Paas & van Merriënboer, 2020; Saw, 2017; Sentz & Stefaniak, 2019). Otra estrategia es, en el caso de estudiantes sin experiencia previa en programación, utilizar diagramas visuales para ayudar a delimitar el flujo de control. De este modo, la información percibida contiene tanto los detalles concretos del código como las estructuras abstractas que los contienen y pueden ser visualizados en un golpe de vista, incluso el proceso de construcción de los diagramas promueve prestar atención a los aspectos funcionales o estructurales, por ejemplo, se pueden utilizar los diagramas Nassi Shneiderman (1973; Davis, 1998) cuya efectividad para disminuir la carga cognitiva ha sido demostrada incluso experimentalmente (Katona, 2022; 2023). Nótese que se hace referencia a un "proceso de construcción" de diagramas, no de la manipulación digital de bloques visuales prediseñados, por ejemplo Scratch. Los diagramas NSD re realizan generalmente en papel, aunque existen aplicaciones de software para ello: https://structorizer.fisch.lu/ (Structurizer, 2024).

1.3 Objetivos y competencias

El objetivo principal de la investigación es lograr la elaboración de un material de estudio efectivo y práctico que facilite el aprendizaje y manipulación de algoritmos y estructuras de datos básicas en el contexto de la programación. Nuestro enfoque teórico reconoce la importancia de abordar tanto los aspectos cognitivos

como los metacognitivos de dicho proceso. Como se argumentó anteriormente en el nivel teórico, algunos de los problemas que enfrentamos se convierten en objetivos específicos de aprendizaje, a saber:

(1) Facilitar la toma de consciencia del propio proceso de solución: el material debe estar diseñado para ayudar a los estudiantes a reflexionar sobre sus propios procesos de pensamiento mientras resuelven problemas de programación. Esto les permitiría identificar áreas que requieren mejora en su enfoque para resolver problemas. (2) Fomentar la flexibilidad para cambiar de nivel de abstracción: deben proporcionarse ejercicios y actividades que ayuden a los estudiantes a comprender y trabajar con diferentes niveles de abstracción, desde el nivel del problema hasta la escritura de código. (3) Proporcionar explicaciones claras y concisas que reflejen la claridad y concisión de las explicaciones del profesor sobre los conceptos fundamentales de programación reconociendo la dificultad de los estudiantes para registrar todas las explicaciones del profesor. (4) Promover la generación de auto-explicaciones: alentar a los estudiantes a explicarse a sí mismos los conceptos y procesos que están aprendiendo. Esta estrategia metacognitiva ayuda a reforzar la comprensión. (5) Diseñar un material de estudio que optimice el proceso de aprendizaje, utilizando principios instruccionales como evitar la sobrecarga de información para facilitar la comprensión y retención. (6) Proporcionar ejemplos y análisis de problemas resueltos suficientemente detallados con diagramas que estructuren el proceso.

Estos objetivos apuntan a desarrollar diferentes competencias de aprendizaje. Por un lado, la consciencia metacognitiva se promueve mediante la habilidad de reflexionar sobre el propio proceso de pensamiento mientras resuelven problemas de programación. Por otro lado, se busca fomentar la flexibilidad cognitiva en los estudiantes, permitiéndoles comprender y trabajar con diferentes niveles de abstracción. Además, el ejercicio de auto-explicación también incide en el logro de una comunicación efectiva y en la autoregulación del aprendizaje. Asimismo, ayuda a los estudiantes a aplicar los conocimientos teóricos a situaciones prácticas.

2 Método

Se optó por un método cualitativo mixto (Borrego y otros, 2009; Creswell, 2017) para el desarrollo de un estudio de caso exploratorio y descriptivo (Yin, 2009). La justificación de esta decisión es que no abundan antecedentes, como se explicó previamente y que se buscó explorar el contexto para el desarrollo de un nuevo material obteniendo información que orientara dicha producción y que permitiera comprender e interpretar en profundidad la experiencia del estudiante ingresante.

Considerando los desafíos y objetivos establecidos en la introducción, se creó un cuadernillo que ofrecía soluciones a problemas de programación, con un enfoque en la optimización de la presentación de los contenidos, el análisis del código y la promoción de la reflexión por parte de los estudiantes. Este material se concibió como un complemento para los estudiantes que ya estaban cursando la asignatura de Algoritmos y Estructuras de Datos, brindándoles la oportunidad de desarrollar la habilidad de explicar los procesos de programación a sí mismos. Se anticipaba que, al mejorar la comprensión de los conceptos fundamentales, se fortalecería la capacidad deductiva posterior. Aunque el enfoque top-down puede ser desafiante desde el principio, se vuelve más accesible con un entendimiento detallado de los procesos involucrados de menor nivel.

2.1 Materiales

El cuadernillo se inspiró en un material de práctica redactado por primera vez en 2016 para estudiantes de una universidad privada de la provincia de Santa Fe, Argentina. El documento había sido subido a un repositorio y se habían recibido mensajes con consultas de lectores latinoamericanos que buscaban material adicional sobre algoritmos de programación estructurada. Inicialmente, este material consistía en explicaciones detalladas acompañadas de diagramas, que representaban las explicaciones que un profesor proporcionaría oralmente en clase. Aunque la copia original no esté disponible, la esencia del contenido y su evolución se encuentran reflejadas en la versión actual del material, lo cual es el foco central de la presente investigación. Antes de sugerir la incorporación del cuadernillo como material de estudio, se consideró oportuno realizar un sondeo para evaluar la percepción de los estudiantes (El material se encuentra en proceso de revisión para su posterior publicación en línea. Se puede solicitar la dirección para acceder al documento completo enviando un correo electrónico al autor. Actualmente, se encuentra

disponible una vista previa reducida en el siguiente enlace: https://www.aacademica.org/veronica.dangelo/12).

Se invitaron inicialmente setenta y ocho (78) estudiantes universitarios de una universidad pública de la provincia de Santa Fe, de los cuales, sesenta y cuatro (64) aceptaron participar; 15 mujeres, 49 varones, media de edad 18.83 (DE = 1.08). Los investigadores no ejercían como docentes en dicha institución. Los estudiantes cursaban el primero de dos cuatrimestres de una asignatura anual de programación dividida en dos partes: el primer cuatrimestre estaba dedicado a la enseñanza de los algoritmos y estructuras de datos básicos sin implementación en un lenguaje concreto; durante el segundo cuatrimestre se utilizaba el lenguaje Python. La asignatura se dictaba durante el 1° año de Ingeniería en Sistemas de Información.

El cuadernillo constaba de 40 páginas tamaño A4, divididas en tres partes. En la primera parte había (1) una primera sección introductoria donde se explicaba el propósito del material de estudio, (2) una segunda sección dedicada al análisis de situaciones del mundo real en las cuales pueden identificarse estructuras de problemas, y (3) una tercera sección (Figura 1) orientada al pseudocódigo y al análisis de enunciados identificando las abstracciones más importantes: comenzando por las estructuras de datos, ya que éstas orientan la asociación con las estructuras de proceso asociadas.

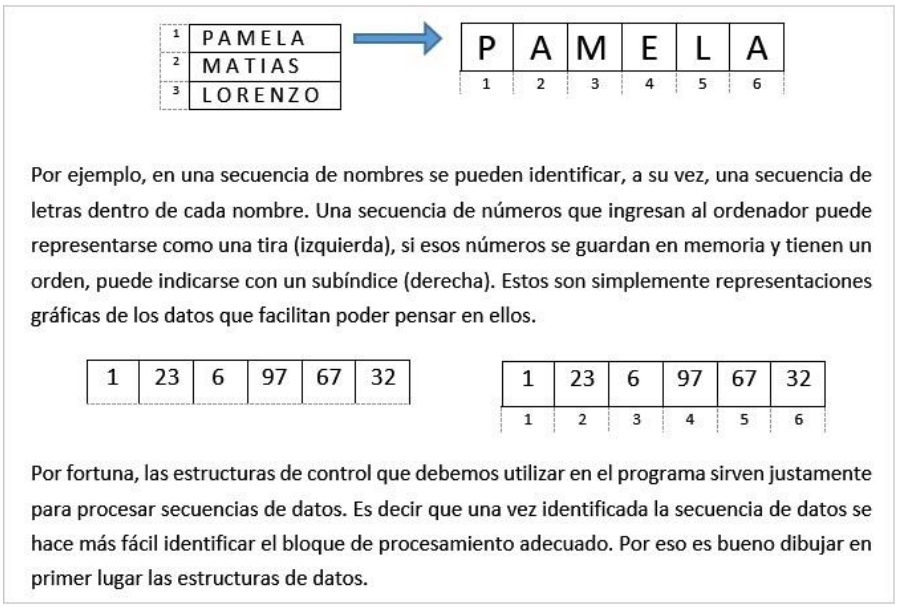

Figura 1. Extracto de la introducción

La segunda parte del material (Figura 2) contenía 24 enunciados de ejercicios de programación. Los enunciados iban acompañados de explicaciones que emulaban las de un profesor y fueron clasificadas en tres tipos (a) explicaciones inductivas: explicaban a qué clase más abstracta pertenece una acción concreta o un conjunto de datos concretos. Por ejemplo, en la figura 1, se explica que un conjunto de 35 números constituye una secuencia de datos, y el cálculo de sus cuadrados es la operación sobre esa secuencia. Las acciones leer y mostrar que se piden en el enunciado se corresponden con funciones que son palabras reservadas del lenguaje.

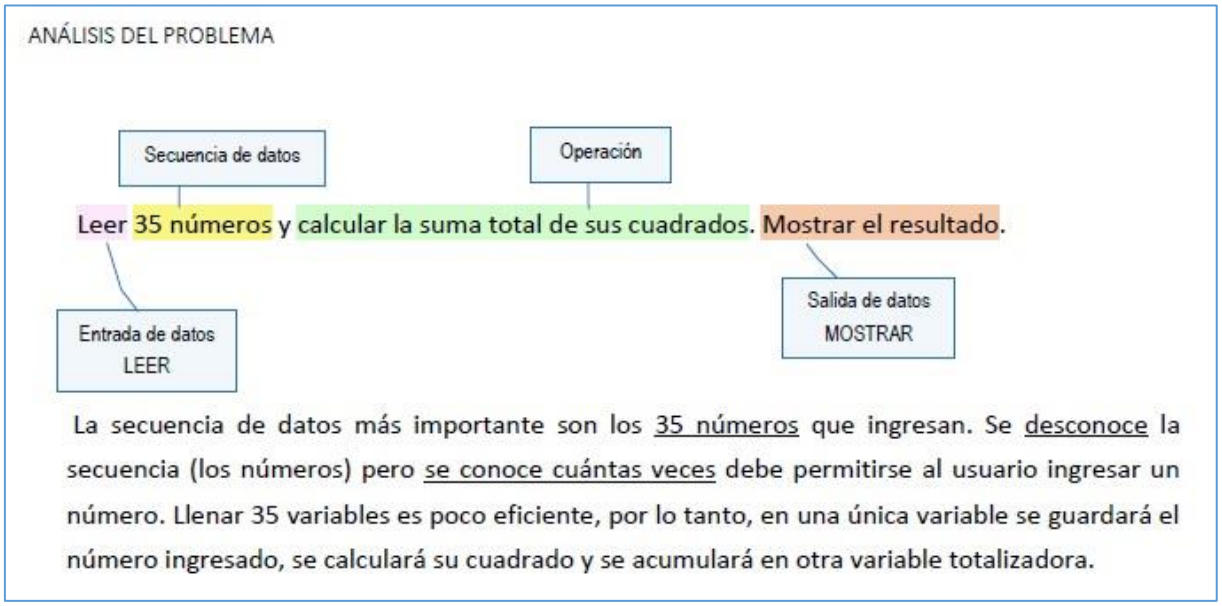

Figura 2. Ejemplo de explicaciones inductivas en el enunciado

Por otro lado, había (b) explicaciones deductivas cuando, conociendo un proceso general, éste se aplicaba a un caso concreto. Por ejemplo, en la figura 3, se activaban conocimientos previamente explicados sobre inicialización, iteración y la palabra reservada LEER. Conociendo cómo deberían funcionar en general, se explicaba cómo funcionaban en el caso concreto.

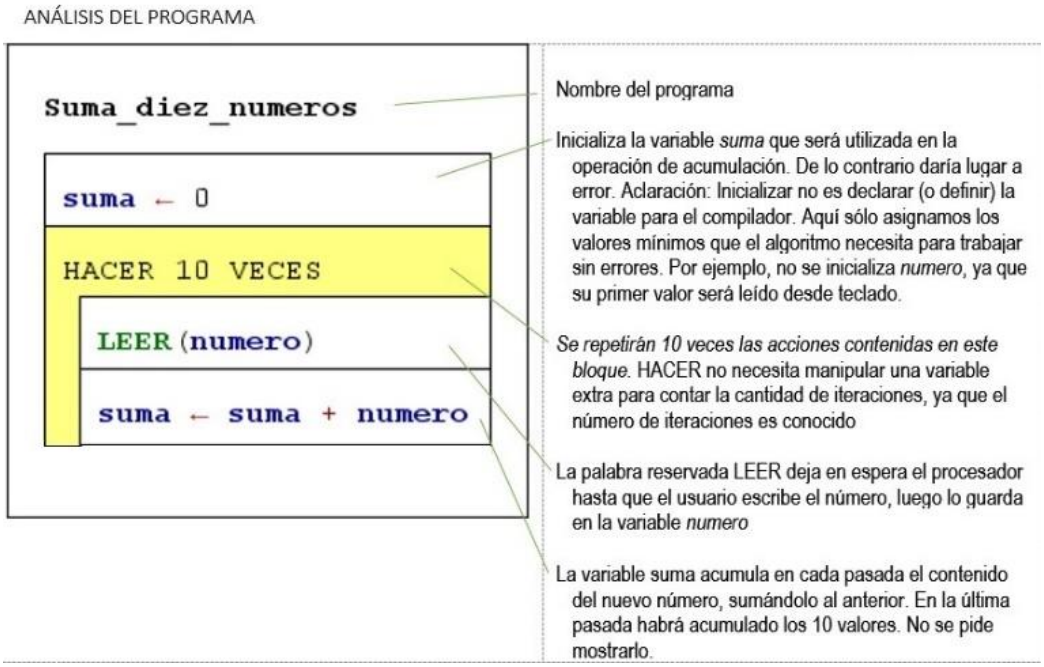

Figura 3. Ejemplo de explicaciones deductivas

Las explicaciones abductivas (c) asociaban un caso con otro caso conocido, permitiendo realizar comparaciones entre casos análogos. Por ejemplo, la comparación entre soluciones de la figura 4.

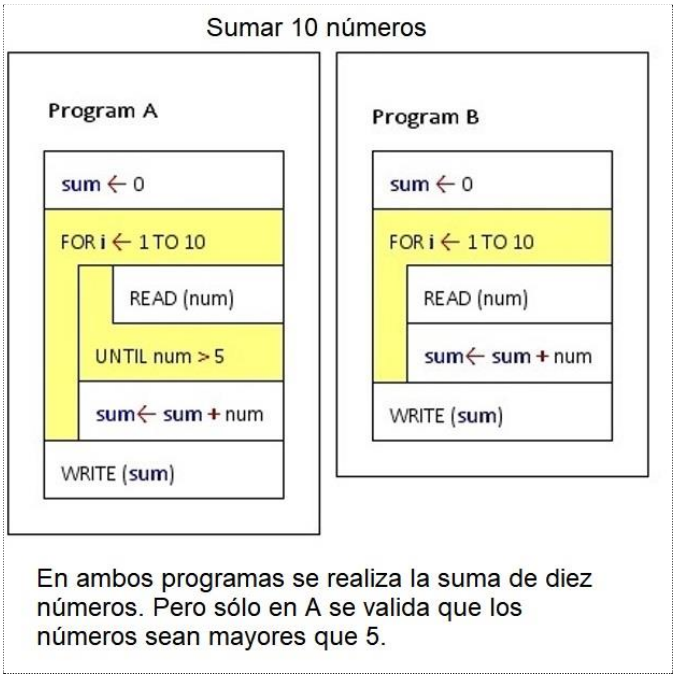

Figura 4. Ejemplo de explicación abductiva

Se esperaba que, a medida que los estudiantes fueran incorporando conceptos como iteración, inicialización, acumulación, entre otros, tenderían a identificar los casos concretos como instancias de esos conceptos, y necesitarán cada vez menos de las explicaciones. Pero hasta que eso no ocurriera, estas explicaciones funcionarían como andamiaje, explicando cómo debería proceder el estudiante explicándose a sí mismo el proceso: deducir ejemplos de las estructuras generales (descender) o bien, dado un caso concreto, pensar a qué tipo de dato o proceso pertenece (ascender).

Debajo de cada enunciado se presentaba una solución en diagrama NSD, a la derecha de la cual se colocaron las explicaciones, lo más cerca posible de la instrucción correspondiente, siguiendo los principios de diseño instruccional anteriormente citados (Merrill, 2002; 2012). Según las recomendaciones sobre carga cognitiva (Sweller, 1988; 1994), en lugar de solicitar al alumno la resolución del problema, se presentaron soluciones analizadas como andamiaje previo. Se esperaba que, si esta forma de análisis era bien recibida por el estudiante, sirviera como modelo para desarrollar su propio análisis de los ejercicios que resuelva en el futuro en otras prácticas de algoritmia. En la tercer parte se presentaron 10 ejercicios con espacios vacíos para que los alumnos participantes de la investigación "imitaran" el mismo análisis (Figura 5).

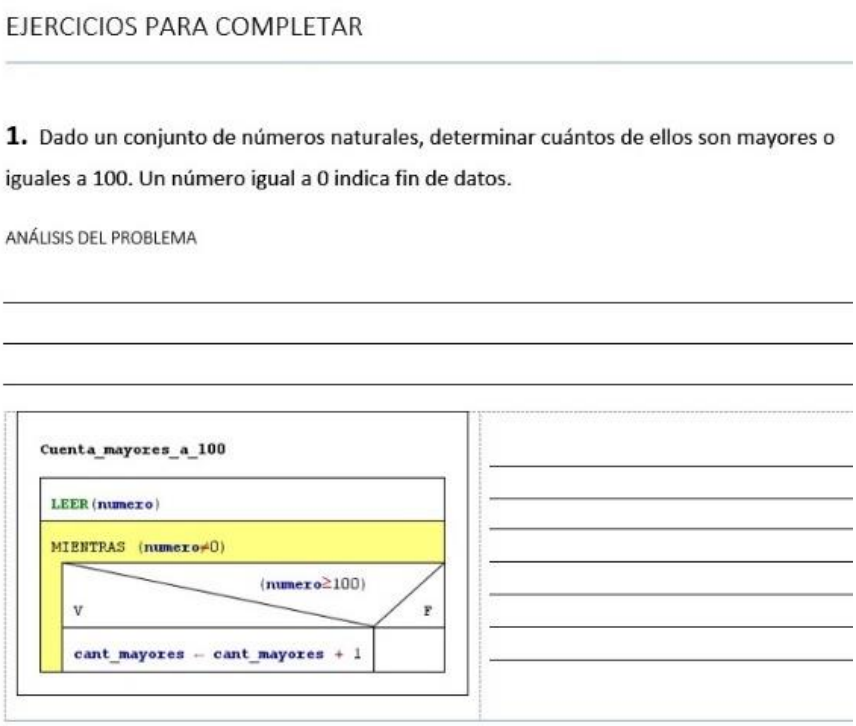

Figura 5. Ejemplo de ejercicio para completar

2.2 Procedimiento

El profesor recomendó el cuadernillo desde el inicio de la asignatura, el cual fue enviado en formato digital como parte de la bibliografía no obligatoria. Se indicó que se trataba de una investigación y que quienes estuvieran interesados en participar podían hacer lo siguiente: leer la parte introductoria del cuadernillo, leer las soluciones de los ejercicios, intentar resolver la tercera parte y responder las preguntas del cuestionario para aportar sus opiniones sobre este material.

Las respuestas se recolectaron a los cuatro meses, al finalizar la parte de la asignatura dedicada a pseudocódigo, para dar tiempo a los estudiantes de incursionar en los conceptos teóricos y la práctica con algoritmos correspondiente a la asignatura.

Se utilizaron dos cuestionarios autoinformados, el primero (Tabla 1) basado en la escala de aprendizaje percibido (PLS) desarrollada por Rovai et al. (2009) para estudiantes de educación superior en diferentes contextos educativos, tanto tradicionales como virtuales. El segundo cuestionario (Tabla 2) constaba de preguntas más centradas en el uso del cuadernillo, diseñadas para recopilar información sobre las opiniones de los estudiantes, así como para explorar temas de interés o aspectos que los estudiantes pudieran aportar y que no hubieran sido considerados previamente por los investigadores. Ambos cuestionarios se presentaron en un único formulario de Google Forms.

Tabla 1. Cuestionario PLS de Aprendizaje Percibido.

Califica las siguientes afirmaciones marcando con una X el grado de acuerdo según la escala de la derecha.	Nada en absoluto						Total-mente
	0	1	2	3	4	5	6
1. Puedo organizar el material del curso siguiendo una estructura lógica.							
2. No puedo producir una guía de estudio del curso para futuros estudiantes.							

Califica las siguientes afirmaciones marcando con una X el grado de acuerdo según la escala de la derecha.	Nada en absoluto						Total-mente
3. Puedo utilizar las habilidades y destrezas aprendidas en este curso fuera de clase.							
4. He cambiado mis actitudes sobre programación como resultado de este curso.							
5. Puedo criticar inteligentemente los materiales del texto utilizados en este curso.							
6. Me siento más autosuficiente como resultado del contenido aprendido en este curso.							
7. No he ampliado mis habilidades y destrezas como resultado de este curso.							
8. Puedo demostrar a otros las habilidades y destrezas aprendidas en este curso.							
9. Siento que soy un razonador más sofisticado como resultado de este curso.							

Tabla 2. Preguntas, opciones y respuestas en el segundo cuestionario.

N°	Pregunta	Opciones	Frecuencia absoluta (N=64)	Frecuencia porcentual
1	¿Cuándo comenzaste a programar?	1. En esta materia.	18	28,13
		2. Hace más de un año	20	31,25
		3. Hace más de 2 años	13	20,31
		4. Hace más de 3 años	13	20,31
2	¿Te resultan útiles los diagramas que se utilizaron en los ejercicios?	1. Extremadamente útil	5	7,81
		2. Muy útil	34	53,13
		3. Bastante útil	16	25,00
		4. Un poco útil	9	14,06
		5. Nada útil	0	0,00
3	¿Habías recurrido antes a las soluciones de los ejercicios en la asignatura de programación?	1.Sí	50	78,13
		2. No	14	21,88
4	¿Te resulta útil poder ver la solución de un ejercicio?	1. Extremadamente útil	23	35,94
		2. Muy útil	37	57,81
		3. Bastante útil	4	6,25
		4. Un poco útil	0	0,00
		5. Nada útil	0	0,00
5	¿Conocés materiales de estudio que incluyan la explicación detallada de cada paso de la solución?	1.Muchísimos	0	0,00
		2.Bastantes	5	7,81
		3.Algunos	10	15,63
		4.Muy pocos	15	23,44
		5. Ninguno	34	53,13
6	¿Te resultan útiles las explicaciones detalladas de cada paso de la solución?	1.Extremadamente útil	34	53,13
		2.Muy útil	12	18,75
		3.Bastante útil	10	15,63
		4.Un poco útil	8	12,50
		5. Nada útil	0	0,00
7	¿Te parece útil poder realizar tu propia explicación del proceso?	1.Extremadamente útil	7	10,94
		2.Muy útil	24	37,50
		3.Bastante útil	22	34,38
		4.Un poco útil	10	15,63

N°	Pregunta	Opciones	Frecuencia absoluta (N=64)	Frecuencia porcentual
		5. Nada útil	1	1,56
9	¿Cómo calificarías este cuadernillo?	1. Extremadamente útil	2	3,13
		2. Muy útil	42	65,63
		3. Bastante útil	18	28,13
		4. Un poco útil	2	3,13
		10. Nada útil	0	0,00
10	¿Qué agregarías al cuadernillo? N=107 (número de sugerencias recopiladas)	1. Más ejercicios	30	28,04
		2. Más explicaciones introductorias	10	9,35
		3. Más explicaciones del proceso	67	62,62
11	Escribí tus comentarios y/o sugerencias. Son importantes para mejorar el material de estudio.			

3 Resultados

3.1 Cuestionario PLS

La evaluación del curso con el cuadernillo de andamiaje mostró una recepción generalmente positiva por parte de los estudiantes. Para calcular las puntuaciones, se aplicaron criterios específicos: los ítems 1, 3, 4, 5, 6, 8 y 9 se puntúan directamente utilizando las respuestas en la escala Likert, mientras que los ítems 2 y 7 se puntúan de forma inversa. La puntuación total del cuestionario, que varió de 0 a 54, fue de 31 puntos, lo que representa un 57.41%.

Además, se calcularon puntuaciones de subescala utilizando el siguiente criterio: para la subescala cognitiva, se sumó la puntuación de los ítems 1, 2 y 5; para la subescala afectiva, se sumó la puntuación de los ítems 4, 6 y 9; para la subescala psicomotora (habilidades), se sumó la puntuación de los ítems 3, 7 y 8. Estas puntuaciones podían variar desde mínimo de 0 hasta un máximo de 18 para cada subescala. Los resultados fueron: 9.67 puntos (53.73%) para la subescala cognitiva, 11.63 puntos (64.58%) para la subescala afectiva, y 9.70 puntos (53.91%) para la subescala psicomotora (habilidades).

Las puntuaciones de las subescalas indican percepciones positivas en todas las áreas evaluadas, con la subescala afectiva obteniendo la puntuación más alta. El análisis detallado revela que, si bien todas las subescalas recibieron puntuaciones superiores a la media, la subescala afectiva mostró la percepción más positiva del aprendizaje, seguida de cerca por la subescala psicomotora. La subescala cognitiva también obtuvo una puntuación por encima del promedio, aunque ligeramente más baja en comparación con las otras dos subescalas.

3.2 Preguntas de exploración

Para la pregunta 1, "Cuándo comenzaste a programar", los resultados indican que el 28% de los estudiantes inicia su aprendizaje de programación en esta asignatura, el 31% comenzó más de un año antes, un 20% hace más de dos años y el mismo porcentaje más de tres años antes.

La mayoría de los estudiantes considera útiles los diagramas y el acceso a las soluciones de los ejercicios. También consideran útiles las explicaciones detalladas, sin embargo, son pocos los materiales de apoyo accedidos que contengan ese tipo de explicaciones. La mayoría considera este cuadernillo muy útil. En la pregunta "Qué agregarías a este cuadernillo" algunos estudiantes seleccionaron más de una opción. Del total de 107 sugerencias, predominan "más explicaciones del proceso" (67), le siguen "más ejercicios" (30) y más explicaciones introductorias (10).

3.3 Análisis de los comentarios y sugerencias

La última pregunta de la encuesta solicitó "comentarios y/o sugerencias para mejorar el material de estudio". Los comentarios tuvieron en general una apreciación positiva sobre la utilidad del cuadernillo como complemento. No hay comentarios que pudieran hacer suponer que las explicaciones eran obvias o ya conocidas, por el contrario, parecen llenar un vacío conceptual, especialmente para quienes están habituados a copiar código: "en la universidad nos hacen pensar en cada paso del programa, me cuesta porque gran parte del código lo busco en GitHub". Se observó una tendencia común hacia la sugerencia de "más ejercicios prácticos y ejemplos" antes de abordar programas extensos. Esta sugerencia implica que la introducción gradual de los conceptos y la práctica progresiva podrían mejorar la comprensión y el rendimiento en las tareas más desafiantes.

Además, se recibieron comentarios acerca de la necesidad de estructuras de programación más claras, lo que sugiere que los estudiantes valoran la conexión entre la comprensión del problema planteado y la selección adecuada de las estructuras de programación. Esto destaca la importancia de desarrollar habilidades de análisis de problemas y de diseño al enseñar programación.

Un estudiante expresó que el dibujo de los diagramas podría ralentizar la escritura del programa y que les resultaba engorroso. Cuatro estudiantes aclararon que ya no necesitan de los diagramas para estructurar el código, lo que sugiere que los diagramas podrían dejar de ser útiles cuando los estudiantes adquieren cierta agilidad en la escritura del código.

4 Discusión

En este estudio, hemos explorado la recepción y percepción de un material de estudio diseñado como apoyo para el aprendizaje de programación en el contexto universitario. Los resultados sugieren que el cuadernillo de andamiaje ha sido bien recibido por los estudiantes y ha contribuido positivamente a su aprendizaje.

Resulta relevante destacar que el grupo de estudiantes que participó en la encuesta está matriculado en una asignatura universitaria que emplea los diagramas NSD. Esto sugiere que el principal valor añadido del cuadernillo estuvo asociado a la información proporcionada en las explicaciones de los algoritmos, en contraste con la utilidad directa de los algoritmos en sí o del uso de los diagramas. Se observó que el conocimiento previo de estos diagramas facilitó la recepción del material de estudio por parte de los estudiantes.

Los resultados de ambos cuestionarios revelan una percepción generalmente positiva de los estudiantes respecto al material de estudio proporcionado, centrado en el cuadernillo de andamiaje. La mayoría de los estudiantes considera útiles los diagramas utilizados en los ejercicios, así como también las soluciones detalladas proporcionadas. Esto sugiere que el material diseñado para apoyar el aprendizaje de programación ha sido bien recibido por los estudiantes.

Es importante destacar que una proporción significativa de estudiantes (28%) comenzó su aprendizaje de programación específicamente en el contexto de esta asignatura, lo que subraya la relevancia del material de andamiaje como punto de partida para aquellos que se inician en la programación. Además, la mayoría de los estudiantes recurrieron a las soluciones de los ejercicios durante el curso, lo que resalta la importancia de proporcionar recursos que faciliten el proceso de aprendizaje y resolución de problemas.

Los comentarios y sugerencias de los estudiantes para mejorar el material también son valiosos. La solicitud de más ejercicios y explicaciones detalladas del proceso indica un interés por contar con recursos adicionales que refuercen la comprensión y práctica de los conceptos. Las observaciones sobre la utilidad de realizar la propia explicación del proceso sugieren un enfoque activo en el aprendizaje, donde los estudiantes se involucran en la construcción de su propio conocimiento. Estos comentarios y sugerencias proporcionan orientación para futuras mejoras en el material, enfatizando la importancia de ofrecer recursos adicionales y opciones flexibles que se adapten a las necesidades individuales de los estudiantes.

Este trabajo destaca la importancia de proporcionar herramientas y recursos efectivos para apoyar el aprendizaje de programación en el nivel universitario. Además, subraya la necesidad de continuar explorando y desarrollando enfoques pedagógicos que faciliten la transición de los estudiantes desde los conceptos básicos hacia niveles más avanzados de competencia en programación.

4.1 Limitaciones y Direcciones Futuras

Es importante tener en cuenta que este trabajo se centró en la percepción de los estudiantes sobre el material de estudio, y no evaluó directamente el impacto en el rendimiento académico o el aprendizaje efectivo de programación. Futuras investigaciones podrían abordar estas limitaciones mediante la implementación de estudios longitudinales o experimentos controlados que evalúen el impacto del material de estudio en el rendimiento y la comprensión de los estudiantes.

Una mejora potencial para facilitar la replicabilidad y la robustez de futuros estudios podría implicar la selección de participantes que ya posean una base de conocimiento en programación, y la aplicación de un diseño pre post para garantizar que puedan proporcionar percepciones y opiniones informadas antes y después de la intervención, lo que permitiría una evaluación más precisa de cualquier cambio o evolución en sus percepciones.

Además, sería beneficioso explorar en mayor profundidad cómo diferentes enfoques pedagógicos y recursos pueden influir en el proceso de aprendizaje de programación, especialmente en relación con la transición desde los bloques visuales hacia la escritura de programas reales.

5 Conclusiones

En este estudio, se investigó un enfoque destinado a guiar a los estudiantes hacia niveles metacognitivos más abstractos a partir de instrucciones básicas, mediante la presentación de explicaciones complejas pero muy detalladas junto a cada instrucción. Se exploró la recepción de un nuevo material diseñado para el aprendizaje de la programación en el ámbito universitario, sugiriendo la viabilidad y el potencial de ampliar este tipo de material. Es importante reconocer las limitaciones inherentes a este trabajo, dado su carácter exploratorio y descriptivo, lo cual implica que los resultados deben interpretarse con ciertas restricciones. Se orienta hacia futuras mejoras y actualizaciones del material, valorando las percepciones de los estudiantes, sin la intención de realizar mediciones cuantitativas precisas del rendimiento del aprendizaje en comparación con otros materiales.

Desde el surgimiento de iniciativas para promover el pensamiento computacional, los bloques visuales han evolucionado hacia objetos más lúdicos y manipulables, desviándose de su función original como estructuradores de procesos. Los diagramas NSD emergen como un potencial facilitador de la transición entre los bloques visuales y la programación real, particularmente para aquellos estudiantes que han aprendido a programar con bloques. Sin embargo, persiste una brecha significativa entre la enseñanza de programación en la educación secundaria y en la universitaria. Mientras que la introducción a la programación en entornos más visuales y lúdicos puede fomentar un aprendizaje autodidacta y poco reflexivo, las cátedras universitarias priorizan el desarrollo de habilidades de pensamiento abstracto, considerando la evolución hacia un futuro laboral en el que la inteligencia artificial podría reducir la oferta de empleos con baja demanda cognitiva.

Se sugiere que, si bien es importante continuar promoviendo el pensamiento computacional desde etapas tempranas, se debe prestar una atención similar a la enseñanza de programación en el nivel superior. Esto implica el desarrollo de estrategias pedagógicas más adecuadas, especialmente pensadas para acompañar a los estudiantes en esta transición. Abogamos por una mayor inmersión en los principios fundamentales de la psicología de la programación y la psicología cognitiva del aprendizaje, como bases sólidas para brindar un apoyo educativo más profundo y efectivo.

Referencias

Aguirre, J., & Carnota, R. (2009). Historia de la Informática en Latinoamérica y el Caribe: investigaciones y testimonios (Primera ed.). Río Cuarto, Argentina: Universidad Nacional de Río Cuarto. https://www.researchgate.net/profile/Marcelo-Carvalho-13/publication/310625262_Historia_de_la_informatica_en_Latinoamerica_y_el_Caribe_investigaciones_y_testimonios/links/58344a2808aef19cb81f797e/Historia-de-la-informatica-en-Latinoamerica-y-el-Caribe-inv

Alrashidi, H., Ullman, T. D., & Joy, M. (4 de Diciembre de 2020). An empirical evaluation of a Reflective Writing Framework (RWF) for Reflective Writing in Computer Science Education. 2020 IEEE Frontiers in Education Conference (FIE) (págs. 1-9). Uppsala, Suecia: IEEE. https://doi.org/10.1109/FIE44824.2020.9273975

Alt, D., & Raichel, N. (2020). Reflective journaling and metacognitive awareness: insights from a longitudinal study in higher education. Reflective Practice, 21(2), 145-158. https://doi.org/10.1080/14623943.2020.1716708

Anderson, J. R., & Fincham, J. M. (Noviembre de 2014). Extending problem-solving procedures. Cognitive Psychology, 74, 1-34. https://doi.org/10.1016/j.cogpsych.2014.06.002

Armoni, M. (Julio de 2013). On Teaching Abstraction in Computer Science to Novices. Journal of Computers in Mathematics and Science Teaching, 32(3), 265-284.

Aureliano, V. C., Tedesco, P., Caspersen, M., & Tedesco, P. C. (28 de Julio de 2016). Learning programming through stepwise self-explanations. 2016 11th Iberian Conference on Information Systems and Technologies (CISTI) (págs. 1-6). Gran Canaria, España: IEEE. https://doi.org/10.1109/CISTI.2016.7521457

Badali, M., Hatami, J., Farrokhnia, M., & Noroozi, O. (Agosto de 2020). The effects of using Merrill's first principles of instruction on learning and satisfaction in MOOC. Innovations in Education and Teaching International, 59(2), 216-225. https://doi.org/10.1080/14703297.2020.1813187

Bai, C., Yang, J., & Tang, Y. (18 de Julio de 2022). Embedding self-explanation prompts to support learning via instructional video. Instructional Science, 50, 681-701. https://doi.org/10.1007/s11251-022-09587-4

Bell, M. A. (Junio de 1995). Review of Psychology of Programming, by J.-M. Hoc, T. R. G. Green, R. Samurçay and D. J. Gilmore. Journal of Visual Languages & Computing, 6(2), 211-212. https://doi.org/10.1006/jvlc.1995.1011

Biju, S. M. (2013). Difficulties in understanding object oriented programming concepts. En K. Elleithy, & T. Sobh (Ed.), Innovations and Advances in Computer, Information, Systems Sciences, and Engineering. Lecture Notes in Electrical Engineering. 152, págs. 319-326. New York, NY: Springer. https://doi.org/10.1007/978-1-4614-3535-8_27

Booch, G. (Febrero de 1986). Object-oriented development. IEEE Transactions on Software Engineering, SE-12(2), 211-221. https://doi.org/10.1109/TSE.1986.6312937

Borrego, M., Douglas, E. P., & Amelink, C. T. (Enero de 2009). Quantitative, Qualitative, and Mixed Research Methods in Engineering Education. Journal of Engineering Education, 98(1), 53-66. https://doi.org/10.1002/j.2168-9830.2009.tb01005.x

Bosse, Y., & Gerosa, M. A. (Enero de 2017). Why is programming so difficult to learn?: Patterns of Difficulties Related to Programming Learning Mid-Stage. ACM SIGSOFT Software Engineering Notes, 41(6), 1-6. https://doi.org/10.1145/3011286.3011301

Caspersen, M. E. (2023). Principles of Programming Education. En S. Sentance, E. Barendsen, N. R. Howard, & C. Schulte (Edits.), Computer Science Education: Perspectives on Teaching and Learning (Segunda ed., págs. 219-236). Bloomsbury Publishing.

Caspersen, M. E., & Kölling, M. (Marzo de 2009). STREAM: A first programming process. ACM Transactions on Computing Education, 9(1), 1-29. https://doi.org/10.1145/1513593.151359

Ch´ng, S. I. (2018). Incorporating reflection into computing classes: models and challenges. 19(3), 358-375. https://doi.org/10.1080/14623943.2018.1479686

Cheng, P. W., & Holyoak, K. J. (Octubre de 1985). Pragmatic reasoning schemas. Cognitive Psychology, 17(4), 391-416. https://doi.org/10.1016/0010-0285(85)90014-3

Chi, M. (1992). Conceptual Change within and across Ontological Categories: Examples from Learning and Discovery in Science. (R. Giere, & H. Feigl, Edits.) University of Minnesota Press, 129-186. https://philpapers.org/rec/CHICCW

Chi, M. T., Bassok, M., Lewis, M., Reimann, P., & Glaser, R. (1989). Self Explanations: How Students Study and Use Examples in Learning to Solve Problems. Cognitive Science, 13(2), 145-182. https://doi.org/10.1207/s15516709cog1302_1

Chi, M. T., Feltovich, P. J., & Glaser, R. (Abril de 1981). Categorization and Representation of Physics Problems by Experts and Novices. Cognitive Science, 5(2), 121-152. https://doi.org/10.1207/s15516709cog0502_2

Cosmides, L. (1989). The logic of social exchange: Has natural selection shaped how humans reason? Studies with the Wason selection task. En Cognition (Vol. 31, págs. 187-276). https://doi.org/10.1016/0010-0277(89)90023-1

Cox, B. J. (1986). Object oriented programming: an evolutionary approach. USA: Addison-Wesley Longman Publishing. https://doi.org/10.5555/16111

Creswell, J. W. (2017). Research design: Qualitative, quantitative, and mixed methods approaches (Tercera ed., Vol. 3). USA: SAGE Publications. https://www.ucg.ac.me/skladiste/blog_609332/objava_105202/fajlovi/Creswell.pdf

da Rosa, S. (2015). The construction of knowledge of basic algorithms and data structures by novice learners. Proceedings of the 26th Annual Psychology of Programming Interest Group Workshop. 7. Bournemouth, UK: Psychology of Programming Interest Group. https://www.fing.edu.uy/~darosa/S.daRosa.pdf

da Rosa, S. R., & Gómez, F. G. (Abril de 2020). A research model in didactics of programming. CLEI Electronic Journal, 23(1, artículo No. 5), 1-16. https://doi.org/10.19153/cleiej.23.1.5

da Rosa, S., Viera, M., & García-Garland, J. (2020). A Case of Teaching Practice Founded on a Theoretical Model. En K. Kori, & M. Laanpere (Ed.), Informatics in Schools: Engaging Learners in Computational Thinking. 13th International Conference, ISSEP 2020, Tallinn, Estonia, November 16–18, 2020, Proceedings. Lecture Notes in Computer Science 12518, págs. 147-157. Springer. https://doi.org/10.1007/978-3-030-63212-0

Davis, W. S. (1998). Nassi-Shneiderman charts. En W. S. Davis, D. C. Yen, W. S. Davis, & D. C. Yan (Edits.), The Information System Consultant's Handbook (Primera ed., pág. 6). CRC Press. https://doi.org/10.1201/9781420049107

Denning, P. J. (1975). Two misconceptions about structured programming. En E. C. Joseph, & J. D. White (Ed.), ACM '75: Proceedings of the 1975 annual conference (págs. 214-215). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/800181.8103

Denning, P. J. (19 de Marzo de 2022). Systems abstractions. Communications of the ACM, 65(4), 22-24. https://doi.org/10.1145/3517218

Denning, P. J., & Tedre, M. (2021). Computational Thinking: A Disciplinary Perspective. Informatics in Education, 20(3), 361-390. https://doi.org/10.15388/infedu.2021.21

Dijkstra, E. W. (Septiembre de 1968). A constructive approach to the problem of program correctness. BIT Numerical Mathematics, 8, 174-186. https://doi.org/10.1007/BF01933419

Dijkstra, E. W. (1970). Notes on structured programming. EUT report. WSK, Technische Hogeschool Eindhoven, Dept. of Mathematics. https://pure.tue.nl/ws/files/2408738/252825.pdf

Dijkstra, E. W. (1976). A Discipline of Programming (Vol. 613924118). Englewood Cliffs, New Jersey, USA: Prentice-Hall. https://seriouscomputerist.atariverse.com/media/pdf/book/Discipline%20of%20Programming.pdf

Duran, R., Zavgorodniaia, A., & Sorva, J. (Diciembre de 2022). Cognitive Load Theory in Computing Education Research: A Review. (A. J. Ko, Ed.) ACM Transactions on Computing Education (TOCE), 22(4, Artículo No. 40), 1-27. https://doi.org/10.1145/3483843

Elqayam, S., & Over, D. E. (2013). New paradigm psychology of reasoning: An introduction to the special issue edited by Elqayam, Bonnefon, and Over. Thinking & Reasoning, 19(3), 249-265. https://doi.org/10.1080/13546783.2013.841591

Gigerenzer, G., & Hug, K. (Mayo de 1992). Domain-specific reasoning: Social contracts, cheating, and perspective change. Cognition, 43(2), 127-171. https://doi.org/10.1016/0010-0277(92)90060-U

Gupta, U., & Zheng, R. Z. (2020). Cognitive Load in Solving Mathematics Problems: Validating the Role of Motivation and the Interaction Among Prior Knowledge, Worked Examples, and Task Difficulty. European Journal of STEM Education, 5(1, Artículo No. 05), 1-14. https://doi.org/10.20897/ejsteme/9252

Gutiérrez, L. E., Guerrero, C. A., & López-Ospina , H. A. (8 de Febrero de 2022). Ranking of problems and solutions in the teaching and learning of object-oriented programming. Education and Information Technologies, 27, 7205–7239. https://doi.org/10.1007/s10639-022-10929-5

Hazzan, O. (Septiembre de 1999). Reducing Abstraction Level When Learning Abstract Algebra Concepts. Educational Studies in Mathematics, 40(1), 71-90. https://doi.org/10.1023/A:1003780613628

Hazzan, O. (2003). How Students Attempt to Reduce Abstraction in the Learning of Mathematics and in the Learning of Computer Science. 13(2), 95-122. https://doi.org/10.1076/csed.13.2.95.14202

Hazzan, O. (2008). Reflections on teaching abstraction and other soft ideas. ACM SIGCSE Bulletin, 40(2), 40-43. https://doi.org/10.1145/1383602.13836

Hazzan, O., & Kramer, J. (10 de Mayo de 2008). The role of abstraction in software engineering. ICSE Companion '08: Companion of the 30th international conference on Software engineering, 1045-1046. https://doi.org/10.1145/1370175.1370239

Hazzan, O., & Zazkis, R. (Enero de 2005). Reducing Abstraction: The Case of School Mathematics. Educational Studies in Mathematics, 58(1), 101-119. https://doi.org/10.1007/s10649-005-3335-x

Heinonen, A., Lehtelä, B., Hellas, A., & Fagerholm, F. (15 de Diciembre de 2022). Synthesizing Research on Programmers' Mental Models of Programs, Tasks and Concepts -- a Systematic Literature Review. arXiv(2212.07763v1 [cs.SE]), 1-66. https://doi.org/10.48550/arXiv.2212.07763

Hoc, J. M. (1983). Analysis of Beginners´ Problem-Solving Strategies in Programming. En Psyschology of Computer Use (págs. 143-158). Academic Press. http://jeanmichelhoc.free.fr/pdf/Hoc%201983d.pdf

Hoc, J. M., Green, T. R., Samurçay, R., & Gilmore, D. J. (Edits.). (2014). Psychology of Programming. Academic Press.

Izu, C., Schulte, C., Aggarwal, A., Cutts, Q., Rodrigo, D., Gutica, M., . . . Weeda, R. (2019). Fostering Program Comprehension in Novice Programmers - Learning Activities and Learning Trajectories. ITiCSE-WGR '19: Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (págs. 27-52). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3344429.33725

Jenkins, T. (23 de Septiembre de 2002). On the Difficulty of Learning to Program. https://www.psy.gla.ac.uk/~steve/localed/jenkins.html

Johnson-Laird, P. N., & Byrne, R. M. (1991). Deduction. Psychology Press. https://philpapers.org/rec/JOHD-9

Johnson-Laird, P. N., Legrenzi, P., & Legrenzi, M. S. (Agosto de 1972). Reasoning and a sense of reality. The British Psychological Society, 63(3), 395-400. https://doi.org/10.1111/j.2044-8295.1972.tb01287.x

Jusas, V., Barisas, D., & Jančiukas, M. (2022). Game Elements towards More Sustainable Learning in Object-Oriented Programming Course. Sustainability, 14(4), 2325. https://doi.org/10.3390/su14042325

Kastens, K. A., & Liben, L. S. (Enero de 2007). Eliciting Self-Explanations Improves Children's Performance on a Field-Based Map Skills Task. Cognition and Instruction, 25(1), 45–74. https://doi.org/10.1080/07370000709336702

Katona, J. (Febrero de 2022). Measuring Cognition Load Using Eye-Tracking Parameters Based on Algorithm Description Tools. Sensors, 22(3, 912). https://doi.org/10.3390/s22030912

Katona, J. (Abril de 2023). An Eye Movement Study in Unconventional Usage of Different Software Tools. Sensors, 23(8, 3823), 1-14. https://doi.org/10.3390/s23083823

Knorr, E. M. (2020). Worked Examples, Cognitive Load, and Exam Assessments in a Senior Database Course. SIGCSE '20: Proceedings of the 51st ACM Technical Symposium on Computer Science Education (págs. 612–618). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3328778.3366915

Kramer, J. (1 de Abril de 2007). Is abstraction the key to computing? Communications of the ACM, 50(4), 36-42. https://doi.org/10.1145/1232743.12327

Lange, C., Gorbunova, A., Shcheglova, I., & Costley, J. (2023). Direct instruction, worked examples and problem solving: The impact of instructional strategies on cognitive load. Innovations in Education and Teaching International, 60(4), 488-500. https://doi.org/10.1080/14703297.2022.2130395

Margulieux, L. E., & Catrambone, R. (2021). Scafolding problem solving with learners' own self explanations of subgoals. Journal of Computing in Higher Education, 33, 499-523. https://doi.org/10.1007/s12528-021-09275-1

Mercier, H., & Sperber, D. (2017). The Enigma of Reason. Harvard University Press. https://www.hup.harvard.edu/books/9780674237827

Merrill, M. D. (Septiembre de 2002). First principles of instruction. Educational Technology Research and Development, 50(3), 43-59. https://doi.org/10.1007/BF02505024

Merrill, M. D. (2012). First Principles of Instruction: Identifying and Designing Effective, Efficient, and Engaging Instruction (Primera ed.). Hoboken, NJ, USA: Pfeiffer (John Wiley & Sons). https://digitalcommons.usu.edu/usufaculty_monographs/100/

Merrill, M. D. (2018). Using the First Principles of Instruction to Make Instruction Effective, Efficient, and Engaging. En R. E. West, Foundations of Learning and Instructional Design Technology: Historical Roots and Current Trends. https://doi.org/10.59668/3

Nassi, I., & Shneiderman, B. (Agosto de 1973). Flowchart techniques for structured programming. ACM SIGPLAN Notices, 8(8), 12 - 26. https://doi.org/10.1145/953349.95335

Nathan, M. J., Mertz, K., & Ryant, R. (Abril de 1994). Learning Through Self-Explanation of Mathematics Examples: Effects of Cognitive Load. the Annual Meeting of the American Educational Research Association (AERA), (pág. 9). New Orleans, LA, USA. https://files.eric.ed.gov/fulltext/ED372095.pdf

Over, D. E. (26 de Octubre de 2009). New paradigm psychology of reasoning. Thinking & Reasoning, 15(4), 431-438. https://doi.org/10.1080/13546780903266188

Paas, F., & van Merriënboer, J. J. (Agosto de 2020). Cognitive-Load Theory: Methods to Manage Working Memory Load in the Learning of Complex Tasks. Current Directions in Psychological Science, 29(4), 394–398. https://doi.org/0963721420922183

Prather, J., Becker, B. A., Craig, M., Denny, P., Loksa, D., & Margulieux, L. (2020). What Do We Think We Think We Are Doing?: Metacognition and Self-Regulation in Programming. ICER '20: Proceedings of the 2020 ACM Conference on International Computing Education Research (págs. 2-13). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3372782.34062

Prather, J., Pettit, R., McMurry, K., Peters, A., Homer, J., & Cohen, M. (2018). Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. ICER '18: Proceedings of the 2018 ACM Conference on International Computing Education Research (págs. 41-50). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3230977.323098

Risha, Z., Barria-Pineda, J., Akhuseyinoglu, K., & Brusilovsky, P. (2021). Stepwise Help and Scaffolding for Java Code Tracing Problems With an Interactive Trace Table. En O. Seppälä, & A. Petersen (Ed.), Koli Calling '21: Proceedings of the 21st Koli Calling International Conference on Computing Education Research (págs. 1-10, Artículo No. 27). New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3488042.3490508

Rovai, A. P., Wighting, M. J., Baker, J. D., & Grooms, L. D. (Enero de 2009). Development of an instrument to measure perceived cognitive, affective, and psychomotor learning in traditional and virtual classroom higher education settings. The Internet and Higher Education, 12(1), 7-13. https://doi.org/10.1016/j.iheduc.2008.10.002

Saidova, D. E. (Junio de 2022). Analysis of the Problems of the Teaching Object-Oriented Programming to Students. International Journal of Social Science Research and Review (IJSSRR), 5(6), 229-234. https://doi.org/10.47814/ijssrr.v5i6.418

Sajaniemi, J. (Mayo de 2008). Psychology of Programming: Looking into Programmers' Heads. Human Technology, 4(1), 4-8. https://doi.org/10.17011/ht/urn.200804151349

Saw, K. G. (Octubre de 2017). Cognitive Load Theory and the Use of Worked Examples as an Instructional Strategy in Physics for Distance Learners: A Preliminary Study. Turkish Online Journal of Distance Education, 18(4, Artículo 11), 142-159. https://doi.org/10.17718/tojde.340405

Sentz, J., & Stefaniak, J. (Marzo de 2019). Instructional Heuristics for the Use of Worked Examples to Manage Instructional Designers' Cognitive Load while Problem-Solving. TechTrends, 63(2), 209 - 225. https://doi.org/10.1007/s11528-018-0348-8

Simon, H. A. (1986). The information processing explanation of Gestalt phenomena. Computers in Human Behavior, 2(4), 241-255. https://doi.org/10.1016/0747-5632(86)90006-3

Soloway, E., & Spohrer, J. C. (Edits.). (2013). Studying the Novice Programmer. New York. https://doi.org/10.4324/9781315808321

Structurizer, 3.32-19. (21 de Marzo de 2024). https://structorizer.fisch.lu/

Sweller, J. (Abril de 1988). Cognitive Load During Problem Solving: Effects on Learning. Cognitive Science, 12(2), 257-285. https://doi.org/10.1207/s15516709cog1202_4

Sweller, J. (1994). Cognitive load theory, learning difficulty, and instructional design. Learning and Instruction, 4(4), 295-312. https://doi.org/10.1016/0959-4752(94)90003-5

Sweller, J. (2010). Cognitive Load Theory: Recent Theoretical Advances. En J. L. Plass, R. Moreno, & R. Brünken (Edits.), Cognitive Load Theory (págs. 29 - 47). Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511844744.004

Sweller, J. (10 de Febrero de 2016). Story of a research program. (S. Tobias, J. D. Fletcher, & D. C. Berliner, Edits.) Education Review, 23. https://doi.org/10.14507/er.v23.2025

Sweller, J. (Febrero de 2020). Cognitive load theory and educational technology. Educational technology research and development, 68(1), 1-16. https://doi.org/10.1007/s11423-019-09701-3

Weinberg, G. M. (1971). The Psychology of Computer Programming. New York: Litton Educational Publishing.

Wirth, N. E. (1 de Abril de 1971). Program development by stepwise refinement. Communications of the ACM, 14(4), 221-227. https://doi.org/10.1145/362575.3625

Wirth, N. E. (1973). Systematic Programming: An Introduction. Upper Saddle River, NJ, USA: Prentice Hall. https://doi.org/10.5555/540371

Wirth, N. E. (Diciembre de 1974). On the Composition of Well-Structured Programs. ACM Computing Survey, 6(4), 247-259. https://doi.org/10.1145/356635.356639

Yin, R. K. (2009). Case Study Research: Design and Methods (Cuarta ed., Vol. 5). Sage. https://books.google.com.co/books?id=FzawIAdilHkC&lpg=PP1&hl=es&pg=PR4#v=onepage&q&f=false

Yorganci, S. (Octubre de 2020). Implementing flipped learning approach based on 'first principles of instruction' in mathematics courses. Journal of Computer Assisted Learning, 36(5), 763-779. https://doi.org/10.1111/jcal.12448

Zarestky, J., Bigler, M., Brazile, M., Lopes, T., & Bangerth, W. (2022). Reflective Writing Supports Metacognition and Self-regulation in Graduate Computational Science and Engineering. Computers and Education Open, 3, 100085. https://doi.org/10.1016/j.caeo.2022.100085